POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Open Set Recognition inspired Predictive Maintenance

Relatore
prof. Tania Cerquitelli

Candidato
Marco Momo

Tutor aziendale
Francesca Cipollini

Anno Accademico 2021-2022

Abstract

In the context of Predictive Maintenance, architectures, systems, and models are developed to monitor industrial processes and detect failures at early stages, by driving maintenance operations only when they are completely necessary. In this thesis, we investigate unsupervised and semi-supervised Anomaly Detection models for Predictive Maintenance that do not rely on the availability of failure data, and we propose a novel approach based on the Open Set Recognition framework. Indeed, Open Set Recognition models can extend the capabilities of Anomaly Detection models by enriching the concept of normality with the addition of semantic classes that characterize the industrial process, while they preserve the capability of detecting abnormal patterns. Moreover, the more complete concept of normality leads the model to more accurate and stable anomaly predictions. Finally, the proposed model leads to the definition of an interpretable condition indicator that can be monitored for driving maintenance decisions. We investigate the models on two benchmark datasets of condition monitoring of rolling bearing, by conducting quantitative as well as qualitative analysis.

Contents

Chapter 1

Introduction

In the industrial context, every unplanned machine downtime can cause a stop in the company's production resulting in significant costs and potential loss in revenues. For this reason, companies are perfecting their maintenance strategies in order to be able not only to detect an already happening failure but to identify its incipient characteristics and prevent it. Due to this necessity, manufacturers perform maintenance activity to keep machines healthy and maximize the plant's reliability and productivity. Historically, maintenance operations were treated in a *time-driven* or *fault-driven* fashion. In fact, in *Reactive Maintenance (RM)* (also found as Corrective Maintenance) actions are performed only when a component has already seen a failure, that is fault-driven. While in *Preventive Maintenance (PM)* checks, services and replacements are carried out at regularly scheduled intervals, or else time-driven. Such plannings are based on mathematical degradation models and statistical analysis typically conducted by the components' manufacturer, and they indicate at which point of the elapsed life the components should theoretically be replaced. A relevant example is the L_{10} bearings rating life [40]: a number that indicates how many millions of revolutions a rolling bearing can perform, with 90% of reliability. Out of this span, the substitution of the rolling bearing is suggested. However, such rules do not often stick to the reality: bearings may be substituted when they could run for more time, and there is a 10% probability of getting into unexpected fault before the designated life.

On the other hand, *RM* maximizes the components' life but it often leads to higher maintenance cost [34]: existing studies show that repairing costs in reactive mode is normally 3 times more than the ones with a scheduled basis [29]. This is not surprising because faulty components could, for example, begin to vibrate and overheat by causing damage also to other connected parts.

Differently, in **Predictive Maintenance (PdM)** operations are conducted only when they are completely necessary, by developing software and systems that continuously monitor the production and predict failure at early stages. Therefore, in this setting maintenance are *status-driven*. For this reason **PdM** would be by far the most reliable and economically convenient maintenance strategy, as shown in Figure 1.1.

Nevertheless, **PdM** has its own challenges, requiring to develop an accurate system able to predict failure events in advance. Since the complexity of industrial machines and

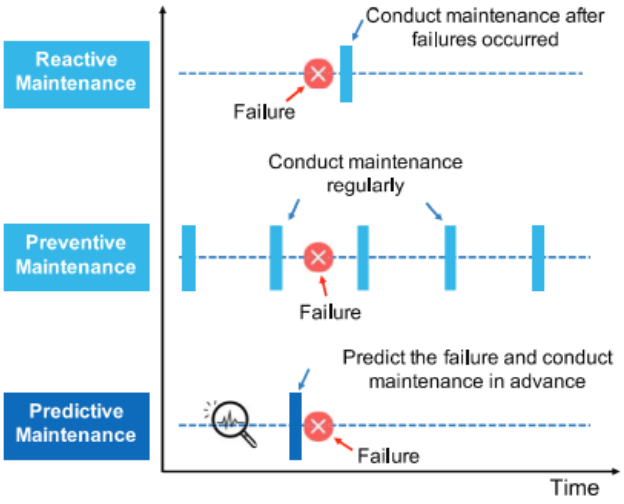

Figure 1.1. Maintenance plans of *RM*, *PM* and *PdM*, from [34].

the non-deterministic behavior of failures make the problem intractable from a purely analytical viewpoint, **data-driven** techniques are considered.

In fact, the concept of *PdM* has existed for many years. However, only recently, the new emerging technologies and computational systems allow manufacturers to tackle many industrial problems by making accessible *Predictive Maintenance* strategies [31]; among them, we mention:

- **IoT sensors** for data acquisition [46]: multiple sensors are mounted on machines or single components and they enable the collection of data from many physical conditions of the production process, by gathering, therefore, a huge volume of data that can be then stored in cloud facilities.

- **Big data techniques for data manipulation**: data from sensors must be cleaned, transformed, merged, and resorted to a suitable form for *ML* models, but the massive amount of information requires using large-scale distributed techniques, like *Spark* [42].

- **Advances in Machine Learning (ML)**: mathematical models that can learn how to tackle a problem from data; for example, they can infer the imminent occurrence of a failure by recognizing hidden schema inside sensors' signals.

There are two main axis for developing a *PdM system*: **fault diagnosis** and **fault prognosis**. The former detects failure at early stages by spotting anomalous behaviors. With "early stages" we intend that a machine is worn but can still work without compromising safety and reliability. The latter predicts the *Remaining Useful Life (RUL)* by learning how the degradation process behaves. Modelling *RUL* is undoubtedly more

appealing, however, it is also a more challenging task because distinct failures can depict very different degradation patterns. For that reason, it is often unfeasible to obtain stable prognostic algorithms. Hence, in this thesis, we will discuss only **fault diagnosis** algorithms.

More in general, *ML* approaches for *PdM* can be divided into two main families: *supervised* and *unsupervised* models; the most relevant difference is that the supervised methods require the presence of failures in the historical data while the unsupervised ones do not. Supervised approaches are by far the most effective when we have data. However, since most manufacturers usually use a *Preventive Maintenance* strategy we have (hopefully) a few failures in our historical data, even with years of recording. Moreover, machines can break with many different fault types that can depict even widely different behaviors: for example, a given fault type could be easily detected by looking at vibration sensors while it turns out invisible to thermal cameras, similarly, another fault could be evident from power consumption whereas the other sensors depict normal patterns. Therefore, to learn strong *supervised* models, we would need not only many failures but also enough of them for each fault type, which is almost always an unfeasible requirement. On these grounds, the employment for *supervised* algorithms is often limited to specific circumstances, while *unsupervised* algorithms are widely considered.

In this thesis, we will discuss **unsupervised anomaly detection** models for *fault detection* in the context of *Predictive Maintenance*, where the scope is developing a system that can raise warnings by spotting abnormal behaviors from sensor data. Such algorithms ingest data from normal working conditions for understanding which patterns characterize the ordinary behavior, and complementary detect when sensors show abnormal conditions. As a result, *unsupervised fault detection* models learn to detect failures without we need to explicitly teach them what a failure is. The advantage is that usually, we can have an almost unlimited amount of data from normal operating conditions because it is the ordinary way of producing goods in industrial plants.

In general, *anomaly detection algorithms* are asked to decide if a new instance does belong to the same distribution of training data, or if it should be considered as different. Usually, normal data form a dense cluster in the feature space while anomalies or outliers result in isolated points. An anomaly detection algorithm somehow defines the boundaries of such clusters and complementary detect outliers; Figure 1.2 reports the idea on a toy example.

The most popular anomaly detection algorithms are *Local Outlier Factor* [4], *One Class SVM* [26], *Isolation Forest* [22] and *Autoencoders* [16]. The first free are variants of classification algorithms, respectively *K-NN*, *SVM* and *Random Forest*. While *Autoencoders* are neural network architecture originally developed for dimensionality reduction that can be easily used for outliers detection.

The common feature of these algorithms is that they work in a *one-vs-all* fashion, they separate what is considered normal from all other possible deviations. The limitation is that a normal point is just normal and the models have no knowledge about semantic concepts that characterize ordinary points. For examples, if normal points belong to three different classes the algorithms [4], [26], [22] and [16] will not recognize such peculiarity.

These considerations give a link to the field of *Open Set Recognition (OSR)* [38] [39] [1], where classifiers are asked not only to predict classes of new samples but also to

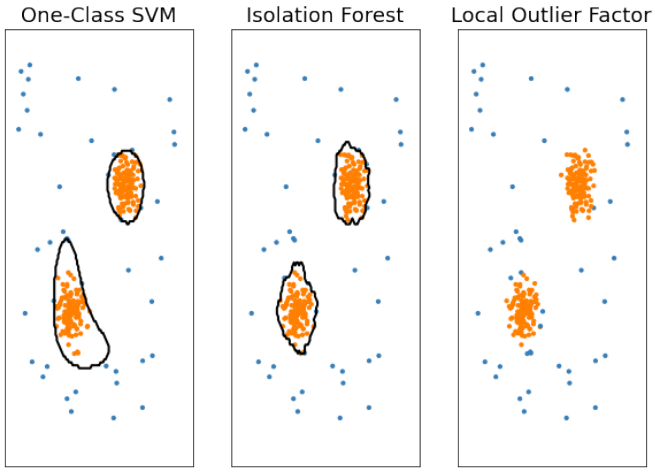

Figure 1.2. A toy example of anomaly detection algorithms on synthetic data: Local Outlier Factor [4], One Class SVM [26], Isolation Forest [22] recognise the dense orange cluster of points as normal while the isolated blue points are anomalies; [26] and [22] are also able of obtaining separation boundaries reported as black curves.

discern if they belong to a concept that is not yet known. From an anomaly detection perspective, they look at normal to the classes discovered during training while they consider anomalies all of the other concepts that are not yet known. It seems a quite strong framework for developing anomaly detection algorithms because the concept of normality can be enriched with further details leading to models that should stick more to reality. These can be exploited in an industrial scenario, where many situations lead to the definition of well-defined classes, for example:

- the same group of sensors monitors the development of a product across different assembly stages: imagine the production of an electronic device where different pieces and circuits are sequentially assembled for building the device;
- the same machine tool can work in different operating modes: think about a cutting blade that performs different cuts depending on the type of material manufactured.

Operating modes or assembly stages could easily define semantic classes and enrich the concept of normality. As a result, we could, for example, detect whenever a blade performs as required a specific kind of cut rather than just understand if the blade is able of cutting well in general.

However, *OSR* is mainly investigated in *Computer Vision* [14], [25]. Therefore, this thesis aims to investigate *Open Set Recognition* models in an industrial scenario, by making a bridge between *PdM* and *OSR*.

With this scope, we propose a novel anomaly detection algorithm build in an *Open Set Recognition* framework and we investigate its performances on two benchmark datasets of condition monitoring of rolling bearings: *PRONOSTIA* [30] and *CWRU* [45]. Indeed, rolling bearings (see figure 1.3) are ubiquitous components in industrial machinery: they

support and guide rotating or oscillating machine elements – such as shafts, axles, or wheels – and transfer loads between machine components, by providing high precision and low friction movements.

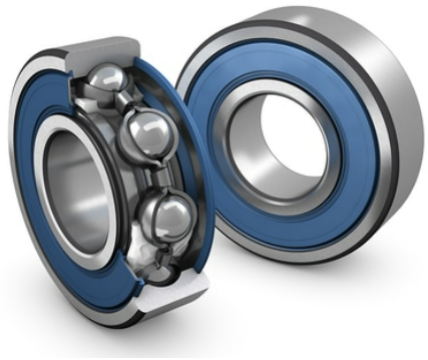

Figure 1.3. A rolling bearing.

According to the literature on this subject, most approaches for rolling bearing fault detection exploit vibration sensors or stator currents. Traditional methods consist of visual analysis by determining handcrafted condition indicators based on physical evidence [18], or by accurately cleaning signals with time-frequency signal processing techniques [33]. The drawback of such kind of methods is that they require relevant domain knowledge for defining critical levels for a condition indicator or understanding when a cleaned signal depicts abnormal patterns. Alternatively, Machine Learning models can be considered to automate the decision process.

Many authors [44] [37] [6] apply feature engineering techniques for determining a good number of predictors from raw signals and then they consider classifiers (SVM and Neural Networks) for building fault detection systems. Others [27] [7] employ Deep Learning algorithms (Autoencoders and Convolutional Neural Networks) directly to raw signals. The main disadvantage of these methods is that they require label datasets that contain anomalies, which are rarely available in real industrial applications. Indeed, since data is typically collected by means of sensors without human intervention, it might not be difficult to obtain a sufficient amount of data for supervised bearing fault detection [35], because labeling requires domain knowledge experts and it is time-consuming and expensive [36] [21]. For that reason, *semi-supervised* and *unsupervised* methods should be considered with more attention for fault detection: *semi-supervised* models require that labels are assigned only to a limited portion of the dataset, while *unsupervised* methods do not need labels at all. For example, [50] proposes a bearing fault detection system with a semi-supervised Variational Autoencoder. While [23] considers an unsupervised Autoencoder for detecting anomalies by controlling the reconstruction error.

On the other hand, the research on *Open Set Recognition* explores the possibility to extend Deep Learning classifiers to the Open Set scenario. [2] proposes a modification in the softmax output layer of a Convolutional Neural Network, based on the information

from the second-last layer, to account also for the unknown class probability. While [25] proposes a further improvement of [2] that consist of extending the neural architecture by considering both classification and reconstruction modules: the last one is devoted to understanding whenever a new sample deviates from the known classes and it plays a crucial role in re-balancing the classification probabilities.

The link between *Fault Detection* and *Open Set Recognition* is that in both situations we can consider reconstruction models, like Autoencoders, for detecting anomalies where in *Fault Detection* an anomaly is a machine fault while in *Open Set Recognition* an anomaly is an unknown class.

The thesis is structured as follows. Chapter 2 present the industrial scenario that we consider, by describing two benchmark datasets: PRONOSTIA 2.1, CWRU 2.2. Next, in Chapter 3 we present exploratory techniques for understanding the relevant aspects of our data, while in Chapter 4 we describe the preprocessing pipeline that extracts features for Anomaly Detection models. After that, in Chapter 5 we start by introducing popular machine learning models, specifically *Autoencoders* 5.1.1 and *XGBoost* 5.1.2, then we propose simple anomaly detection algorithms, and finally we propose a novel approach for fault detection based on the *Open Set* framework. In Chapter 6 we show the results of the proposed methods. Finally, we conclude the work with Chapter 7, by giving further directions to our approach.

Chapter 2

Datasets

In this chapter, we report the datasets considered in this thesis for investigating the performances of anomaly detection models. We examine two benchmark datasets of condition monitoring of rolling bearings: *PRONOSTIA* [30] and *CWRU* [45].

In Section 2.1 we describe *PRONOSTIA*, which consist in a set of *run-to-failure* experiments where bearings are tested and monitored until they show a failure. This dataset allows the investigation of the capability of anomaly detection models in recognizing the failures at early stage.

In Section 2.2 we describe *CWRU*: a dataset in which brand new and defective rolling bearings are tested and monitored for a few seconds. The defects are artificially injected and they can be of different types. The dataset provides labels that indicate which experiments should be considered *normal* or *faulty* together with the fault type, allowing us to objectively evaluate the models' effectiveness in detecting anomalies.

2.1 Pronostia

The *PRONOSTIA* dataset was published for the *IEEE Prognostics and Health Managemen Data Challenge* in 2012 [30], where the participants competed for building the prognostic algorithm in order to predict the Remaining Useful Life (RUL) of rolling bearings. For the competition, the organizers provided a platform to enable testing and to evaluate methods for machine fault diagnosis and prognosis, by making available data of condition monitoring of rolling bearings in *run-to-failure* experiments. Specifically, in each experiment an intact bearing is installed, then the execution is monitored over time by means of vibration and temperature sensors. Moreover, the bearings rotate in 3 distinct operating conditions, in terms of rotating speed and radial load.

During the competition, participants received 6 run to failure experiments, two for each operating condition, and they were asked to learn RUL models on this dataset. Then, the model were evaluated by testing the RUL predictions on 11 new *run-to-failure* experiments in which the monitored history was cut off at some point.

In this thesis, we will not use this dataset for modelling RUL; instead we use it for developing fault detection models and inspect whenever they are able of detecting failures

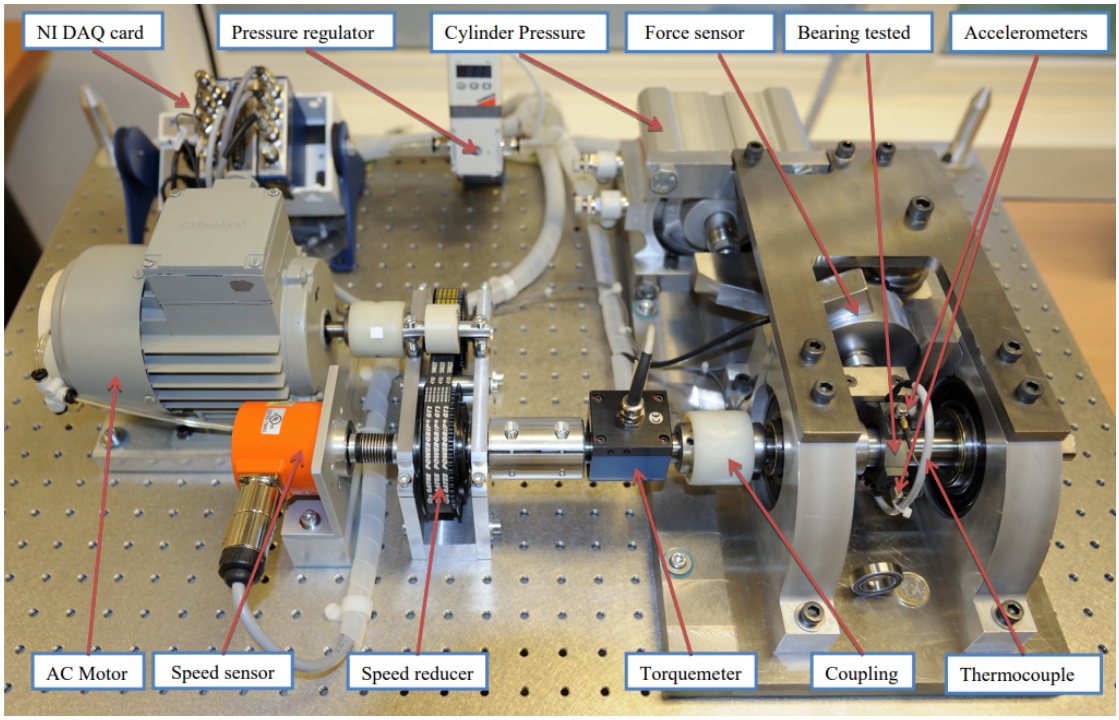

Figure 2.1. Experimental setup of *PRONOSTIA*, from [30].

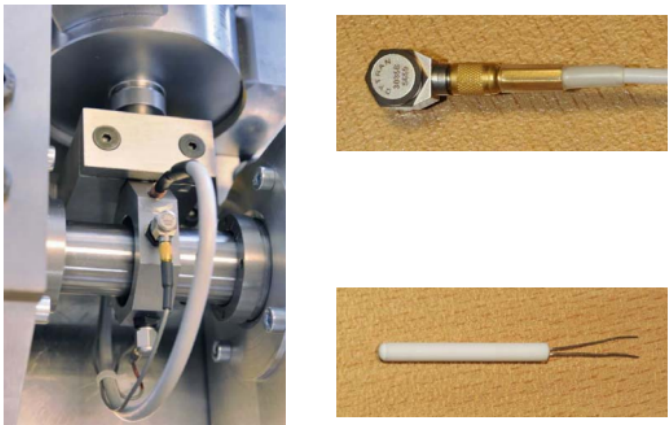

Figure 2.2. Detail of the installed sensors, from [30]. On the right, a vibration and a temperature sensors.

at early stages. Indeed, since during the experiment the bearings from intact become worn, we can analyse the entire degradation process.

Figure 2.1 reports the experimental setup of *PRONOSTIA*: on the left we can see an alternating current (AC) motor; on the right the tested bearing is installed in an

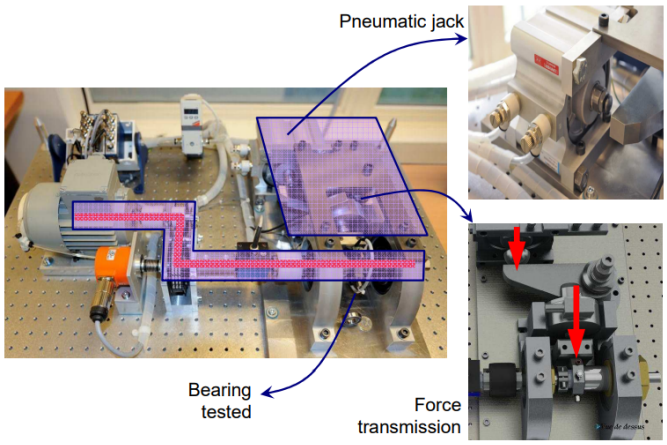

Figure 2.3. Detail of the load transmission part, from [30].

appropriate housing; on the top right corner there is a system that applies the radial load on the bearing housing. The sensors are placed directly on the bearing housing, as Figure 2.2 shows. The two vibration sensors sample at $25.6kHz$ for 0.1 seconds every 10 and they are placed directly on the bearing housing at 12 o'clock and 9 o'clock. The first collect the *vertical* vibrations while the second samples the *horizontal* vibrations. While the temperature sensor continuously sample at $0.1kHz$. The loading transmission system can apply a radial load to the installed bearing, as reported in figure 2.3. This component of the system reduces the bearing's life duration by setting a load value up to the bearing's maximum dynamic load which is $4000N$. As a result, the degradation process is *naturally* accelerated and the experiments last only a few hours. The term *naturally* indicates that the fault are not artificially injected, by they origin from the spontaneous wear of the system.

Bearings can run in 3 different operating conditions :

- **Operating condition 1** : 1800 rpm[1] and $4000N$ load;
- **Operating condition 2** : 1650 rpm and $4200N$ load;
- **Operating condition 3** : 1500 rpm and $5000N$ load.

They differ both for the rotating speed and for degree of stress applied by the radial load. Each experiment was carried out with only one working condition, from the beginning to the end. Table 2.1 list all of the experiments by indicating the available sensor's data, and the corresponding duration. The first number in the experiment's name refer to the operating condition, while the second is a progressive number for the corresponding operating condition. From the table we can see that temperature data is not available for all experiments.

[1] rotation per minute

Experiment	***Sensors***			***Execution time***
	acc_H	***acc_V***	***temp***	
Bearing1_1	x	x	x	7 h47'00"
Bearing1_2	x	x	x	2h25'00"
Bearing1_3	x	x		6h35'30"
Bearing1_4	x	x	x	3h15'19"
Bearing1_5	x	x	x	6h50'19"
Bearing1_6	x	x	x	6h47'50"
Bearing1_7	x	x	x	6h16'21"
Bearing2_1	x	x	x	2h31'40"
Bearing2_2	x	x		2h12'40"
Bearing2_3	x	x		5h26'00"
Bearing2_4	x	x	x	2h5'00"
Bearing2_5	x	x	x	6h25'00"
Bearing2_6	x	x		1h56'40"
Bearing2_7	x	x		38'10"
Bearing3_1	x	x	x	1h25'40"
Bearing3_2	x	x		4h32"40"
Bearing3_3	x	x	x	1h12'10"

Table 2.1. List of PRONOSTIA experiments. The first number in the experiment's name refer to the operating condition.

2.2 Case Western Reserve University dataset (CWRU)

The *Case Western Reserve University*[2] makes publicly available a dataset in which whole and defective rolling bearings are tested and monitored for a few seconds [45]. The test apparatus is reported in Figure 2.4: a motor on the left makes rotating a rolling bearing appropriately installed in its housing.

The defects are artificially injected in different locations: inner race, ball, outer race; moreover, they are engraved with different diameters and depths: Table 2.2 reports all of the tested failure configurations. Moreover, both for whole and defective experiments the bearings rotate at 4 different speeds: 1797, 1772, 1750, and 1730 *rpm* . Finally, vibration data is collected using an accelerometer that samples at $12kHz$ attached to the housing with a magnetic base.

[2] A private university in Cleveland, Ohio.

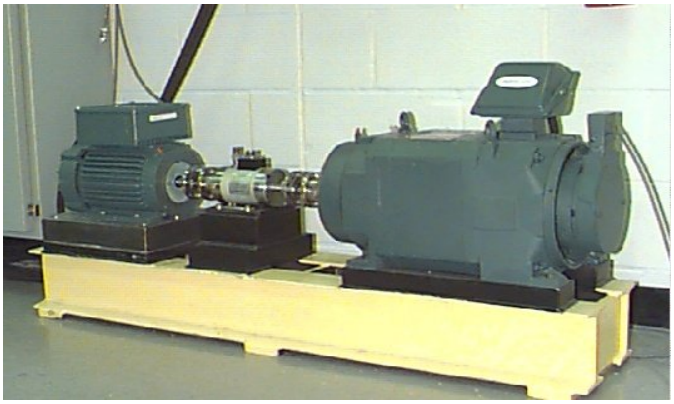

Figure 2.4. *CWRU* test apparatus.

Table 2.3 reports the overall time of monitoring for whole and defective rolling bearings separated by operating conditions.

Location	***Diameter***	***Depth***
Inner Raceway	0.007	0.011
Inner Raceway	0.014	0.011
Inner Raceway	0.021	0.011
Inner Raceway	0.028	0.05
Outer Raceway	0.007	0.011
Outer Raceway	0.014	0.011
Outer Raceway	0.021	0.011
Outer Raceway	0.028	0.05
Ball	0.007	0.011
Ball	0.014	0.011
Ball	0.021	0.011
Ball	0.028	0.15

Table 2.2. Fault configurations. The *Diameter* and *Depth* are expressed in inches.

Operating condition	***Time (s)***	
	Normal	***Faulty***
1797 rpm	20.33	152.3
17702 rpm	40.33	152.3
1750 rpm	40.33	152.3
1730 rpm	40.47	152.49

Table 2.3. The overall time of monitoring separating for operating conditions.

Chapter 3

Exploratory Data Analysis

In this chapter we explore signals of the datasets *PRONOSTIA* 2.1 and *CWRU* 2.2 with the aim of gaining insights for developing effective anomaly detection models. In Section 3.1 we introduce mathematical concepts for managing **IoT** sensor's data. Then, in Section 3.2 we apply exploratory techniques to the two datasets.

3.1 Techniques

In our setting, the health condition of rolling bearings is monitored by one or more **IoT sensor**. They can continuously sample information from which we can understand the quality of the bearing's rotation. More in general, IoT sensors are placed in every industrial machine for monitoring the processes, and the more the machine is complex the more sensors we need for correctly observing the operations.

In Section 2.2 we have presented the dataset *CWRU* where bearings are continuously monitored for a few seconds, using a single vibration sensor. While in Section 2.1 we have introduced the dataset *PRONOSTIA* where two vibration sensors sample information in batches of 0.1 seconds every 10: the monitoring is not continuous but consists of sampled intervals alternated by breaks. This last approach is common in many industrial scenarios, because manufacturers have numerous machines and often multiple plants, therefore, the continuous monitoring of every component will lead to an unmanageable quantity of data. Indeed, sensors are often characterized by a high sampling frequency. For example, the ones of *PRONOSTIA* samples at $25kHz$, that is 25600 samples every second. With an hour of monitoring without pauses, we get around 92 million of samples. On the other hand, if we sample for 0.1 seconds every 10 we get only a total of

$$25600 \times \frac{3600}{10} \times 0.1 = 921600$$

records every hour, which is a great reduction.

In general, from an *IoT sensor* we obtain a sequence of signals that brings snapshots of the process that we are monitoring. Therefore, we need to formalize the signals in mathematical terms and introduce techniques that can extract useful features from them.

Specifically, in Section 3.1.1 we introduce the **time series**; while in Section 3.1.2 we present **Fourier analysis**, a set of techniques to extract features in the frequency domain; finally, in Section 3.1.3 we propose a method for dimensionality reduction.

3.1.1 Time series

A **time series** is a sequence of numbers

$$x_t \in \mathbb{R}, \quad t = 0,1,2,...,n-1,$$

where n is the temporal length of the series, and the set of indexes $t = 0,1,\dots,n-1$ indicates the temporal order of the sequence.

The sequence can be compactly written as a vector

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \in \mathbb{R}^n.$$

While a **multivariate time series** is a sequence of vectors

$$x_t \in \mathbb{R}^l, \quad t = 0,1,2,\dots,T-1,$$

where l is the number of channels, or else the number of sensors, and it can be written as a matrix

$$X = \begin{bmatrix} x_{0,0} & x_{0,1} & \dots & x_{0,l} \\ x_{1,0} & x_{1,1} & \dots & x_{1,l} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,0} & x_{n,1} & \dots & x_{n,l} \end{bmatrix} \in \mathbb{R}^{n \times l}.$$

Time series are suitable when we consider just one sensor, while *multivariate time series* are appropriate when we have multiple ones.

Moreover, since *IoT sensors* sample at a constant sampling rate, f, from the time series indexes it is possible to obtain the exact temporal location of the samples. Indeed, the inverse of the sampling frequency,

$$\Delta t = \frac{1}{f},$$

indicates the elapsed time from one sample to the next one. Therefore, for every time index t we can obtain the corresponding temporal location by computing $t \cdot \Delta t$.

Finally, we point out that in the context of *Predictive Maintenance*, time series, rather than single sampled points, should be considered as instances of a dataset. Indeed, since we are interested in understanding whether a machine performs the process as it should, we have to assess whether the whole signal is anomalous or not.

3.1.2 Fourier analysis

In this section, we restrict our attention to time series and we introduce **Fourier analysis**: a set of signal processing techniques for extracting features related to the frequency domain. Indeed, the rotational behavior of the ball bearings suggests that most of the important characteristics are closely related to frequencies. Moreover, considering only time series is not restrictive: every multivariate time series can be seen as a set of time series, one for each channel.

In brief, *Fourier Analysis* consists in changing the basis on which we express time series as vectors, bringing out aspects that are not evident over time. In fact, time series are essentially vectors in which the components indicate the precise temporal order.

From basic linear algebra, we know that every vector can be written in terms of a chosen basis of the vector space. Observing a time series over time is equivalent to considering the corresponding vector in the *canonical basis*[1] because the coefficients with respect to the basis are the elements of the sequence themselves. On the other hand, time series from condition monitoring of rolling bearings may show interesting aspects in the frequency domain, because the rotating behavior shows patterns that are repeated over time. However, these aspects are often hidden in the time series and a suitable transformation should be performed for revealing them. To this extent, we introduce **Fourier Analysis**: a set of techniques for extracting features in the frequency domain. An extended description of this argument is outside the scope of the thesis (see [47] for further information) and we introduce only the tools that are useful for our discussions.

Since *Fourier Analysis* involves complex numbers and linear algebra we start by introducing preliminary concepts.

Let $\mathbb{C}$ be the set of complex numbers, that is

$$\mathbb{C} = \{u + \mathrm{j}v, \quad u, v \in \mathbb{R}\},$$

where j is the imaginary unit, u is the real part, and v is the imaginary part. The conjugate of a complex number, $z = u + \mathrm{j}v$, is another complex number with the same real part and the imaginary part flipped in sign,

$$\overline{z} = u - \mathrm{j}v, \tag{3.1}$$

where $\bar{\cdot}$ is the conjugation symbol. While, the product between two complex numbers $z = u + \mathrm{j}v$ and $z' = u' + \mathrm{j}v'$ is given by

$$z \cdot z' = uu' - vv' + \mathrm{j}(uv' + u'v) \tag{3.2}$$

[1] The canonical basis of $\mathbb{R}^n$ is a set of vectors $\mathcal{L} = \{e^{(0)}, e^{(1)}, \ldots, e^{(n-1)}\}$ so that

$$e_i^{(k)} = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases}, \quad \forall k \in \{0,1,\ldots,n-1\},$$

Let $\mathbb{C}^n$ be a complex n-dimensional vector space. A basis of $\mathbb{C}^n$ is a set of n vectors,

$$\mathcal{L} = \{f^{(k)} \in \mathbb{C}^n, \quad k = 0,1,\dots n-1\},$$

so that for every $x \in \mathbb{C}^n$

$$x = \sum_{k=0}^{n-1} \alpha_k f^{(k)},$$

for some set of *coefficients* $\{\alpha_k \in \mathbb{C}\}_{k=0}^{n-1}$, and

$$\sum_{k=0}^{n-1} \alpha_k f^{(k)} = 0 \iff \alpha_k = 0 \quad \forall k \in \{0,1,\dots,n-1\}.$$

Moreover, a base $\mathcal{L} = \{f^{(0)}, f^{(1)}, \dots, f^{(n-1)}\}$ is *orthonormal* if for every couple of basic vector $f^{(k)}, f^{(h)}$ the following equation holds

$$\left\langle f^{(k)}, f^{(h)} \right\rangle = \sum_{i=0}^{n-1} f_i^{(k)} \overline{f_i^{(h)}} = \begin{cases} 1 & h = k \\ 0 & h \neq k \end{cases}$$

If a basis, $\mathcal{L}$, is *orthonormal* it can be shown (see [47] at Section 2.5.2) that for every vector $x \in \mathbb{C}^n$ the coefficients $\{\alpha_k \in \mathbb{C}\}_{k=0}^{n-1}$, can be computed as the inner product between the vector the corresponding basic vector, that is

$$\alpha_k = \left\langle x, f^{(k)} \right\rangle, \quad \forall k \in \{0,1,\dots,n-1\}.$$

For example, the canonical basis, that is the set of vectors $\mathcal{L} = \{e^{(0)}, e^{(1)}, \dots, e^{(N-1)}\}$ so that

$$e_i^{(k)} = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases}, \quad \forall k \in \{0,1,\dots,n-1\},$$

is an *orthonormal* basis, and the *coefficients* are the vector entries themselves.

While, the **Fourier Basis** of $\mathbb{C}^n$ is set of vectors $\{f^{(k)}\}_{k=0}^{n-1}$, so that for every k

$$f_t^{(k)} = \frac{1}{\sqrt{n}} e^{\mathrm{j}\frac{2\pi k}{n}t}, \quad t = 0,1,\dots,N-1.$$

For understanding more their formulation we can use the *Euler formula*,

$$e^{\mathrm{j}\theta} = \cos(\theta) + \mathrm{j}\sin(\theta). \tag{3.3}$$

Specifically, we can rewrite the basis vectors as follow, for every k

$$f_t^{(k)} = \frac{1}{\sqrt{n}} \left(\cos(\frac{2\pi k}{n} t) + \mathrm{j}\sin(\frac{2\pi k}{n} t) \right).$$

That is, every basic vector $f^{(k)}$ consist of two harmonic waves in the real and imaginary part, with oscillating frequency $\omega_k = \frac{2\pi k}{n}$. Moreover, we can show that the *Fourier Basis* is *orthonormal*, see [41]

The **Discrete Fourier Transform** (**DFT**) of a signal is the vector of *coefficients* in the Fourier Basis. Formally, the *DFT* is defined as an operator

$$\mathcal{F} : \mathbb{C}^T \longrightarrow \mathbb{C}^T,$$

that for every signal $x \in \mathbb{C}^T$ it gives as output the coefficients in the Fourier Basis,

$$\mathcal{F}(x) = \begin{bmatrix} \left\langle x, f^{(0)} \right\rangle \\ \left\langle x, f^{(1)} \right\rangle \\ \vdots \\ \left\langle x, f^{(n-1)} \right\rangle \end{bmatrix} \in \mathbb{R}^n.$$

Each coefficient is given by the inner product between the signal vector x and the corresponding basis vector $f^{(k)}$, that is

$$[\mathcal{F}(x)]_k = \left\langle x, f^{(k)} \right\rangle = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cdot \overline{e^{j\frac{2\pi k}{n}t}} = \frac{1}{\sqrt{N}} \sum_{t=0}^{n-1} x_t \cdot e^{-j\frac{2\pi k}{n}t}, \tag{3.4}$$

where, for obtaining the last equality, we use the definition of complex conjugate 3.1 together with formula 3.3. Moreover, it is worth noting that the product inside the summation is a product between complex numbers, defined by the equation 3.2.

In this thesis, we will apply the *Discrete Fourier Transform* only to real-valued signals, so we can further simplify coefficients' formula 3.4. By using the equation 3.2 we obtain

$$\begin{aligned} [\mathcal{F}(x)]_k &= \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cos(\frac{2\pi k}{n}t) - i x_t \sin(\frac{2\pi k}{n}t) \\ &= \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t (\cos(\frac{2\pi k}{n}t) - i \sin(\frac{2\pi k}{n}t)). \end{aligned}$$

Next, by considering the *Discrete Fourier Transform* we can obtain the **Power Spectrum** of a signal: the vector whose components are the squared modulus of the coefficients. Formally, for every signal $x \in \mathbb{R}^n$ the *power spectrum* is the vector $\mathcal{S}(x)$ whose components are computed as follow

$$[\mathcal{S}(x)]_k = \mathfrak{Re}([\mathcal{F}(x)]_k)^2 + \mathfrak{Im}([\mathcal{F}(x)]_k)^2, \quad k \in \{0,1,2,\ldots,n-1\}, \tag{3.5}$$

where $\mathfrak{Re}(\cdot)$ and $\mathfrak{Im}(\cdot)$ are respectively the real and imaginary part of a complex number.

Moreover, we can show (see [41]) that the spectrum for the real-valued signals is symmetric with respect to the middle element of the vector, that is

$$[\mathcal{S}(x)]_k = [\mathcal{S}(x)]_{n-k}, \quad \forall k \in \{0,1,2,\ldots,N\},$$

where $N = \frac{n}{2}$ if n is even and $N = \frac{n-1}{2}$ if n is odd.

As a result, we can consider the first $N+1$ component of the *power spectrum* without losing information. In the following, when we say "we compute the *Power Spectrum*" we

implicitly intend that we compute the power spectrum and we select only the first $N+1$ values.

Moreover, since signals are sampled from *IoT sensors* with a constant sampling rate we can relate the *Power Spectrum* coefficients with **sample frequencies** [24]. Specifically, if f is the sampling rate of our sensor from which we obtain signals, the *sample frequency* of the coefficient with index $k \in \{0,1,\dots,N\}$ is given by

$$\frac{f}{n} \cdot k, \tag{3.6}$$

and it is expressed in Hertz (Hz).

As an example, if $f = 25.6kHz$, $n = 2560$, every index $k \in \{0,1,\dots,1280\}$ is related to the sample frequency $\frac{f}{n} \cdot k = 10 \cdot k$. As a result, the sampling frequency ranges in the set $k \in \{0,10,20,\dots,12800\}$.

3.1.3 Principal Component Analysis

In this subsection we introduce the **Principal Component Analysis** (**PCA**): a technique for dimensionality reduction. We will use it in the next sections for visualization purposes. Indeed, *IoT sensors* lead to high dimensional data, therefore it is difficult to understand the high-level property that characterizes the signals. However, the possibility to summarize the data by means of a few features may allow gaining useful insights.

We consider a dataset arranged in a matrix $X \in \mathbb{R}^{m \times n}$, where m is the number of instances and n is the number of features. We can see the matrix as a set of row vectors $x_1, x_2, \dots, x_m \in \mathbb{R}^n$. We would like the reduce the dimensionality of the vectors using a linear transformation. Then, we can consider a matrix $W \in \mathbb{R}^{d \times n}$, with $d < n$, that defines the linear mapping

$$x \in \mathbb{R}^n \longrightarrow Wx \in \mathbb{R}^d$$

that reduces the dimensionality from n to d. Moreover, we want that the lower dimensional representation summarizes most of the information of the original vector. That is, we would like to restore the original input from $h = Wx$. Then, we consider another matrix $U \in \mathbb{R}^{n \times d}$that defines the linear mapping

$$h \in \mathbb{R}^d \longrightarrow Uh \in \mathbb{R}^n,$$

that from the compressed form recover the original dimensionality.

Intuitively, the transformation induced by W is able of decreasing the dimensionality and preserving most of the information if

$$x_i \approx UWx_i, \quad i = 1,\dots,m.$$

In *PCA*, the compression matrix W and the recovering matrix U are obtained by minimizing the distance between x_i and UWx_i in *euclidean norm*,

$$\underset{W \in \mathbb{R}^{d \times n}, U \in \mathbb{R}^{n \times d}}{argmin} \sum_{i=1}^{m} ||x_i - UWx_i||_2^2. \tag{3.7}$$

As a result, by solving 3.7, we can use W to decrease the dimensionality.

Interestingly, the numerical routines that solve 3.7 gives us the proportion of variance explained by each principal component (see [9] Section 10.2.2).

Moreover, before solving 3.7 is a common practice to standardize input data [9]. That is, we scale each column of the matrix X so that they have zero mean and unit variance.

Formally, we compute the mean vector

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix},$$

where,

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} X_{i,j}, \quad j = 1, \ldots, n.$$

Then, we define a diagonal variance-covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix},$$

where,

$$\sigma_j = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} (X_{i,j} - \mu_j)^2}, \quad j = 1, \ldots, n.$$

Finally, we standardize every row $x_i = \{X_{i,j}\}_{j=1}^{n}$ as

$$\Sigma^{-1} (x_i - \mu)$$

3.2 Data Exploration

3.2.1 PRONOSTIA

In this dataset, we have signals from condition monitoring of rolling bearings from 17 run-to-failure experiments. The signals are sampled from two vibration sensors and a temperature sensor. However, temperature values are not available for all experiments, therefore we decided to not take them into account during our analysis, while the vibration data is always available and will be considered in the development of our models. The sensors are installed directly in the bearing housing and they sample information in different locations. Namely, the vertical accelerometer (acc_V) is placed at 12 o'clock while the horizontal accelerometer (acc_H) is placed at 3 o'clock.

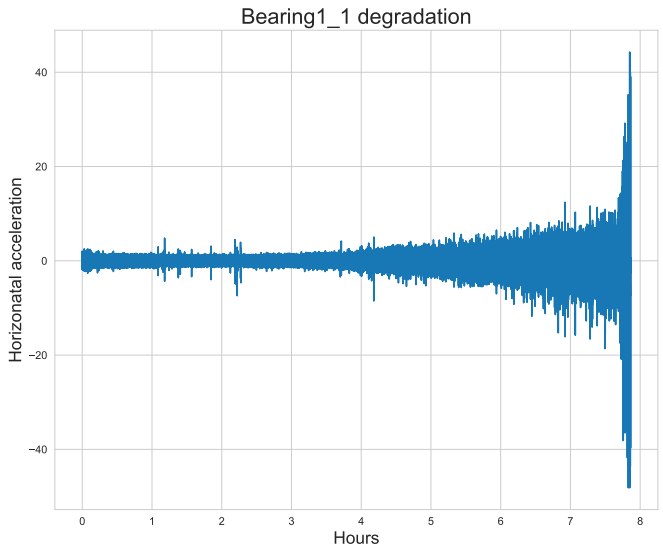

Figure 3.1. The vibration raw signal monitored by the horizontal accelerometer in the first experiment with operating condition 1.

The experiments' duration ranges from 1 hour to 8, and the degradation patterns depict different behaviors.

As an example, we can look at Figure 3.1 where we can see a slow and progressive degradation captured by the *acc_H* sensor: after 4 hours of operation we can recognize the beginning of the degradation process, where the amplitude of the signal starts to increase, and the more time runs the more the amplitude increases until the failure occurs. This is a perfect example of the standard degradation process that we can expect in rolling bearings and more in general industrial machines: the failure is not sudden but it shows early warning signs. Data-driven approaches for *Predictive Maintenance* must recognize these warnings, detecting component degradation before the failure occurrence.

On the other hand, Figure 3.2 shows a sudden failure: the raw signal maintains the same behavior from the beginning of the experiment until a failure unexpectedly occurs.

Of course from a signal similar to the one presented in *Figure* 3.1 the degradation can be easily detected by monitoring simple summary statistics of the signal. For example, the standard deviation can be a good indicator because the amplitude of the signal symmetrically increases both in the positive and the negative directions, and then we can determine when a maintenance operation should be performed by considering a threshold on the statistic's value above which we should stop the execution.

However, this approach has many disadvantages. First, it is a reasoning that works in retrospect: we have no guarantees that degradation can be detected by the standard deviation; maybe the signal's behavior becomes asymmetric and can be seen for example with the skewness, or even the abnormal behavior does not intact significantly the values

of any summary statistics, but it is only visible with the use of different kind of features. Second, even if we know in advance that the degradation will affect exactly the standard deviation of the signal, is not trivial to define a suitable upper bound to the allowed values for the statistic.

Therefore, in order to develop a more robust anomaly detection system, other techniques have to be imagined. Data-driven approaches together with suitable pre-processing pipelines can detect hidden behaviors inside apparently normal raw signals, by emerging the incipient of anomalous patterns. We have explored the presented experiments by the

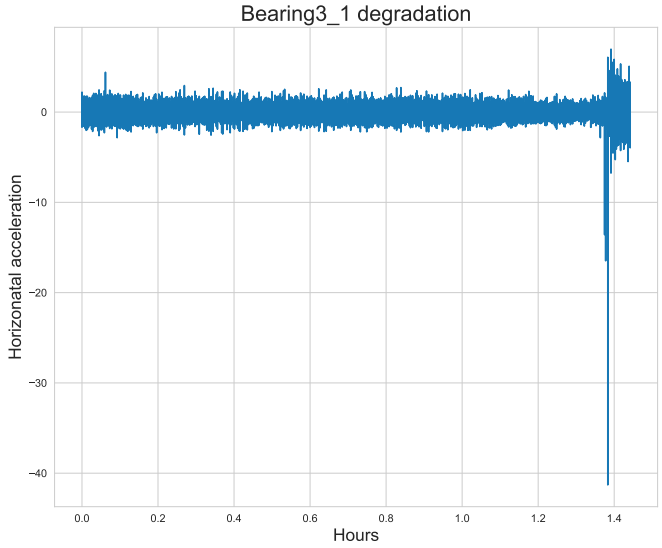

Figure 3.2. The vibration raw signal monitored by the horizontal accelerometer in the first experiment with operating condition 3.

use of one sensor only. It is also interesting to investigate the signals from the two sensors jointly and analyze whether they bring concordant information or not. Indeed, since they are placed at different locations they give a distinct point of view of the system: one sensor could show abnormal behaviors while the other does not. Imagine a situation where a little scratch originated exactly in the location where the *acc_H* sensor points: every time the bearing's balls pass over the scratch the sensor collects data that contains some aspect that is not normal; on the other hand, the scratch is so small that the *acc_V* sensor completely ignores the incipient failure.

Figure 3.3 reports 3 experiments by showing the signals from both sensors. In Figure 3.3 *(b)* the two sensors show the same behavior during the degradation process: after 30 minutes of operation, we can observe a change in the signal shape that proceed in the same manner for both sensors until the fault. While in Figure 3.3 *(a)* and *(c)* the two sensor often reveal different information. In Figure 3.3 *(a)* the horizontal acceleration depicts a smooth degradation while the vertical sensor shows a sequence of spikes spaced out over

time. Similarly, in Figure 3.3 *(c)* we can observe spikes in terms of vertical acceleration when the horizontal does not.

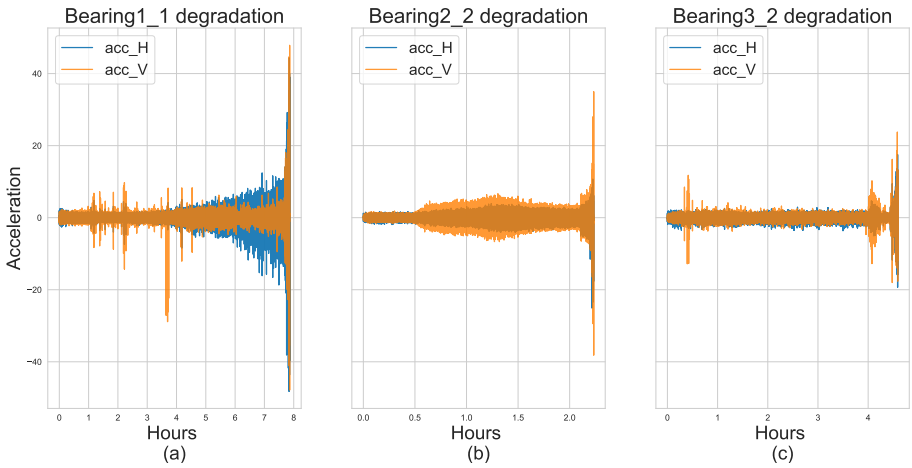

Figure 3.3. Condition monitoring for 3 different experiment, one for each kind of operating condition.

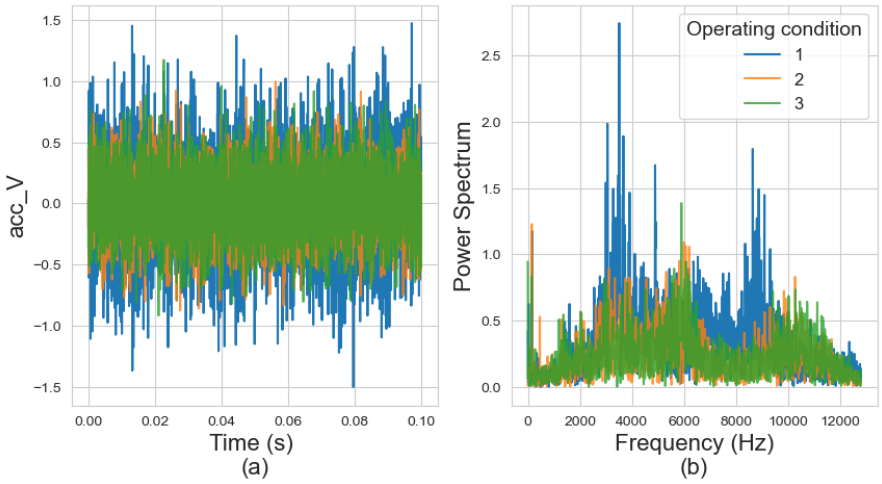

Figure 3.4. Comparison between signals from *acc_V* sensor with different operating conditions, in terms of: (a) raw vibrations, (b) Power Spectrum.

The bearings work in three different operating conditions:

- Operating condition 1 : 1800 rpm and 4000 N radial load;
- Operating condition 2 : 1650 rpm and 4200 N radial load;

- Operating condition 3 : 1500 rpm and 5000 N radial load.

We could expect that different operating conditions may depict signals with different shapes. However, from Figure 3.4 *(a)* it is not trivial to distinguish them. Indeed, the high sampling rate brings 2560 points for each signal, and the samples draw an oscillating behavior making visual analysis hard.

By considering the **Power Spectrum** of the same signals the comparison become clearer as shown in Figure Figure 3.4 *(b)*. We can see that the energy of the distinct signals is focused in different locations of the frequency spectrum. Specifically, the signal of operating condition 1 deeply differs from the others, while signals from operating conditions 2 and 3 are similar to each other.

Thus, the *Power Spectrum* seems promising for extracting features from signals for condition monitoring. Indeed, discriminating the operating condition in which bearings work should be a simpler task than detecting anomalies. So, determining features that should be able to discriminate this aspect should be a first step toward building a promising anomaly detection model.

For that reason, in the following, we further analyze the power spectrum of the signals. Specifically, we compute the *Power Spectrum* of each signal of every experiment and we inspect if the coefficients' values change significantly during the degradation process. As an example, Figure 3.5 reports a heatmap that shows the distribution of the coefficients over time. Precisely, we consider the signals from the *acc_H* sensor of the experiment *Bearing3_2* and we compute the normalized *Power Spectrum* for all signals. That is, for every signal x we compute the *power spectrum* $\mathcal{S}(x)$ and then we normalize it as

$$\tilde{\mathcal{S}}(x) = \frac{\mathcal{S}(x)}{\sum_{k=0}^{N-1}[\mathcal{S}(x)]_k},$$

where N is the signal length. We note that $\tilde{\mathcal{S}}(x)$ is a vector that sums up to one and it has negative entries. As a result, it can be seen as a probability distribution, where the density indicates the energy of the signal allocated to a specific harmonic frequency.

The heatmap of Figure 3.5 is a matrix where the columns correspond to data batches and the rows to frequency, and the entries are the normalized *Power Spectrum*, that is

$$H_{k,i} = [\tilde{\mathcal{S}}(x^{(i)})]_k,$$

where $x^{(i)}$ is the signal of batch i of the experiment *Bearing3_2* from the *acc_H* sensor.

By analyzing Figure 3.5, we can recognize that the coefficients' distribution significantly changes over time by drawing 3 phases:

1. *Phase 1*: approximately before batch 100 the energy distribution is mostly concentrated between 3 and 4 kHz.

2. *Phase 2*: approximately from batch 100 to 1400 the spectrum become mostly located between 1 and 2 kHz.

3. *Phase 3*: from batch 1400 on the frequency spectrum is completely changed.

A possible interpretation is the following. During *Phase 1* the bearing is brand new, it rotates perfectly and shows a spectrum that is proper for its operating condition. While in *Phase 2* the rotational behavior slightly changes. This can be caused by the appearance of small defects, for example when the bearing surface shows natural abrasions. Finally, in *Phase 3* the defects degenerate and the spectrum completely changes by shifting most of the energy to the high frequencies. This kind of behavior is known as **data drift** or

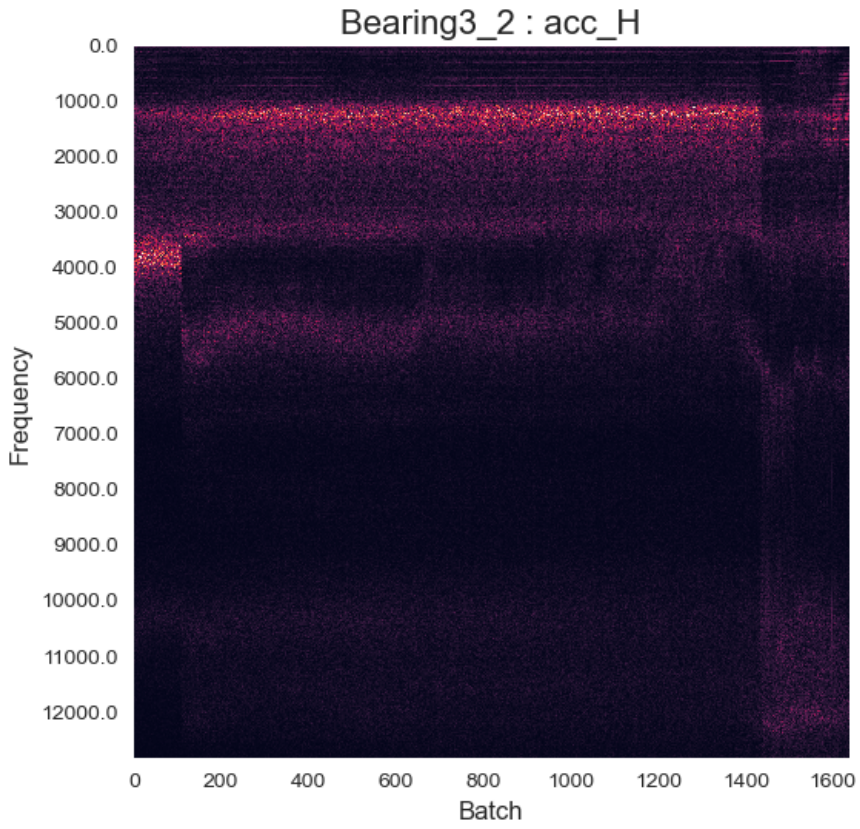

Figure 3.5. The heatmap shows the distribution on the frequency spectrum of the energy of the signals across the different data batches of experiment 3_2.

data shift (for example see [17], [10], [48]): the underling distribution of the data is not stationary but it changes overtime. However, it should not be surprising to discover *data drift* in the context of *Predictive Maintenance*: the rolling bearings degraded over time and the imperfection in the physical system are naturally transferred to a change in the data distribution. In Sections 5.2.2, 5.2.3 we will extensively exploit this behavior for building Anomaly Detection models. Indeed, since the data distribution changes the model should detect the data drift and consequently detect abnormal behavior.

Finally, we would get an overall view of the dataset. Since raw signals as well as power spectrums are high dimensional data, we have to perform dimensionality reduction techniques for observing the whole data by means of a few features. First, we arrange power

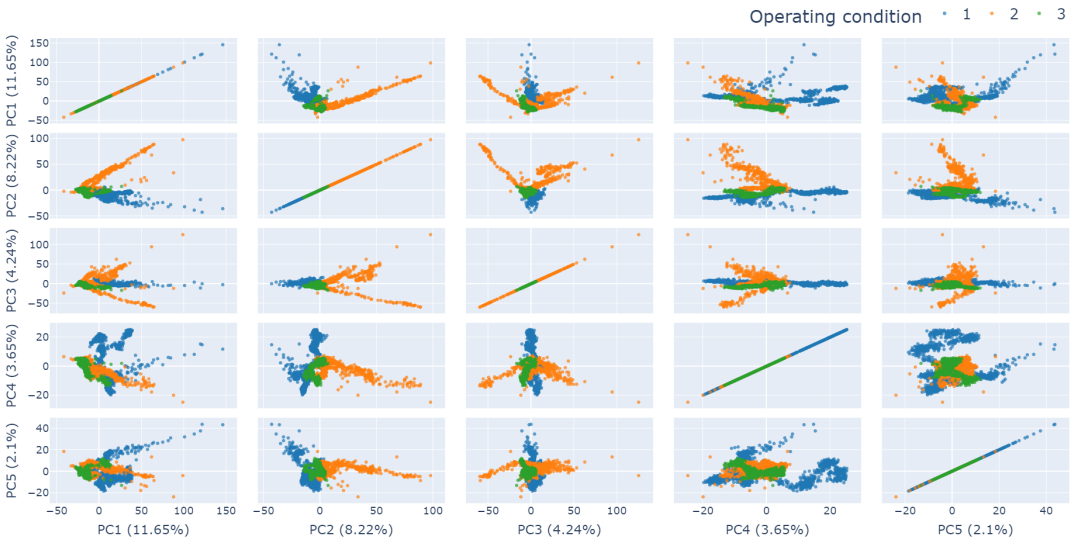

Figure 3.6. Result of *PCA* on *Power Spectrum* data for the first 30 minutes of monitoring from each experiments. The colors refer to the different operating conditions.

spectrum data from the two signals in a single matrix and then we perform **Principal Component Analysis** (**PCA**) by obtaining a 5-dimensional representation of the data. Let X be a data matrix, whose rows are the batches of the experiments and the columns are the power spectrum coefficients of the signals from the two sensors, stacked one after the other, that is

$$X_{i,\cdot} = \left[\mathcal{S}(x_V^{(i)}), \mathcal{S}(x_H^{(i)})\right],$$

where $x_V^{(i)}$ and $x_H^{(i)}$ are the signals of respectively the vertical and horizontal accelerometers for a given batch of some experiment.

First, we build such a matrix by considering only the first 30 minutes of monitoring from each experiment and we apply **PCA**. Figure 3.6 shows the result, where different colors indicate distinct operating conditions. Overall, we can see that the operating conditions are organized into distinct clusters, showing that the different operating conditions correspond to distinct patterns in the signals.

Next, we build the data matrix by considering the whole data that refer to experiment *Bearing*3_2 and we apply the same procedure. Figure 3.7 shows the result, where the colors reflect the monitoring time. Overall, we can see that the first 200 batches form a dense red cluster; the next batches until 1400 form stretched profiles and finally the last batches draw isolated points. A possible interpretation is that the stretched profile refers to the fault incipient as a slow and progressive degradation, while the isolated points correspond to the fault. Moreover, we observe that the phases that we have highlighted are the same individuated in Figure 3.5.

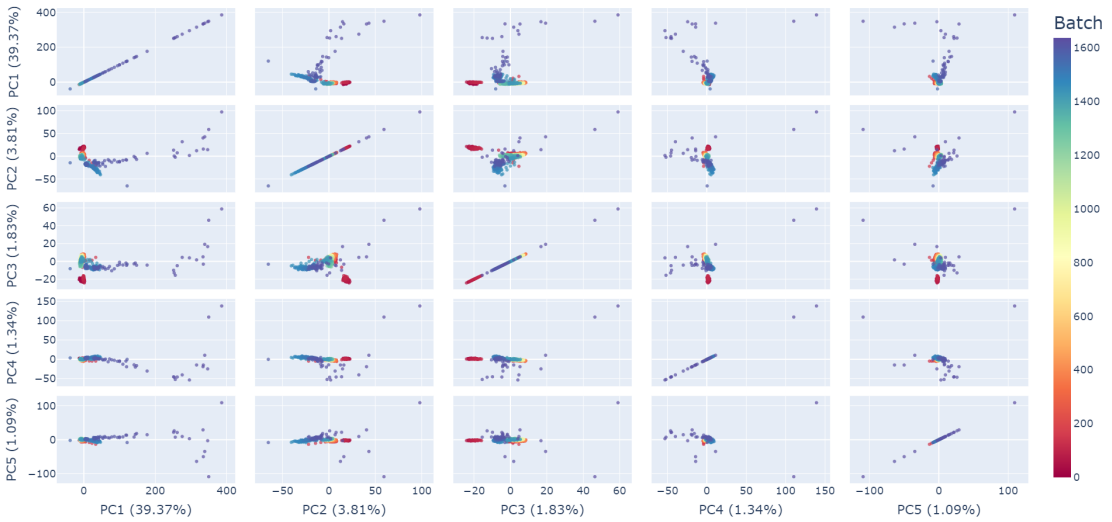

Figure 3.7. Results of *PCA* on *Power Spectrum* data for the experiment *Bearing*3_2. The colors refer to the monitoring time, in terms of the number of the corresponding data acquisition batch. We recall that from a data acquisition batch it elapse 10 seconds.

3.2.2 CWRU

CWRU dataset contains signals from condition monitoring of brand new as well as defective rolling bearings, monitored by a single vibration sensor. The defects are artificially originated by damaging the bearings in specific locations, resulting in different fault types. The experiments' duration ranges from 20 seconds to 152. Moreover, the bearings are tested in four different operating conditions, in terms of rotating speed.

In order to analyse the signals in a similar manner with the previous section, we chunk the signal in windows of 2560 samples. Since the sampling rate of the sensor is $12kHz$ the seconds elapsed in every window are

$$2560 \cdot \frac{1}{12000Hz} \approx 0.21s.$$

Next, as in the previous section we can build a data matrix where the rows correspond to windows and the columns to the *power spectrum*. Then, upon standardization with equation 3.1.3 we can apply *PCA*.

First, we apply the approach by considering only normal data, as Figure 3.8 shows: we can draw two important considerations:

- We are not able of finding points of operating condition 1. Indeed, a more careful analysis shows that the normal data of operating condition 1 are the duplicates of data from class 2: as a result, the points overlap and we can see only one class. Since in Sections 5.2.2 and 5.2.3 we will extensively use the fact of having signals from the

Figure 3.8. Results of PCA on Power Spectrum data for the experiments of *CWRU* dataset that refer to healthy rolling bearings. The colors refer to the different operating conditions.

different operating conditions, we must select only the points from one of the two classes, also for anomaly data. We decide to keep only class 1.

- We can observe a clear separation between the classes: see for example $PC4$ vs $PC2$ and $PC3$. As a result, a classifier can easily distinguish between operating conditions.

Next, we consider only data from *operating condition* 3 and we perform *PCA*. Figure 3.9 shows the result. We can see that the points from *normal* signals form a tight cluster while the *anomalies* are more scattered. Moreover, we note that *normal* data cluster is quite distant from the one of the *inner race* fault, while the difference is less stressed for *ball* and *outer race* faults. From this consideration, we may expect that some *inner race* anomalies will be easy to detect while catching the others may be more difficult.

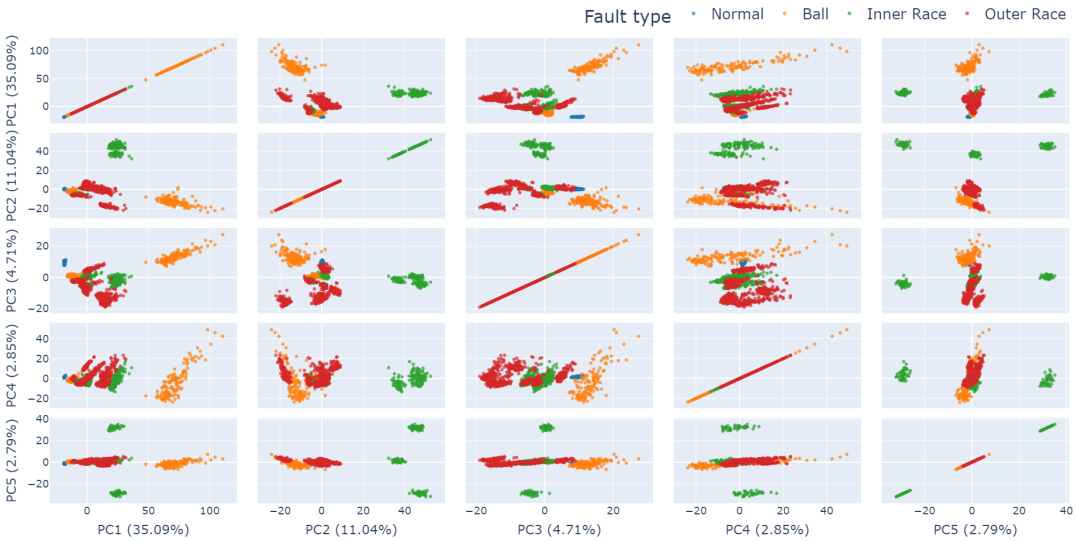

Figure 3.9. Result of PCA on Power Spectrum data for the experiments of *CWRU* dataset. The colors refer to the different failure types.

Chapter 4

Preprocessing

In this Chapter, we describe the *preprocessing operations* that we perform to datasets *PRONOSTIA* 2.1 and *CWRU* 2.2 for preparing the data for the modelling phase described in the next chapter. Specifically, in Section 4.1 we present the algorithm that maps raw signals into several salient features, by computing time-domain as well as frequency-domain aggregations. The *feature extraction algorithm* is in common for both datasets and will be used as a part of the *preprocessing pipelines* in Sections 4.2, and 4.3. In addition, in Sections 4.2, and 4.3 we will detail all of the operations that we perform for transforming raw data into well-organized datasets, which can be then used for training and evaluating *Anomaly Detection* models.

4.1 Feature extraction algorithm

Let we assume that we have a set of signal $\{x^{(i)}\}_{i\in\mathcal{I}}$, where $\mathcal{I}$ is a set of indexes, and every signal is a time series with n samples, $x^{(i)} \in \mathbb{R}^n$. Since n is a large number (we will see that in this thesis $n = 2560$), we would extract a few features that can properly summarize the time series. To this extent, we compute 9 summary statistics of the signal, by gaining time domain aggregations. Next, we consider the signal's *Power Spectrum* and we aggregate its coefficients in frequency bands, by obtaining frequency-domain features.

Formally, let us consider a signal $x = x^{(i)}$ for some $i \in \mathcal{I}$; we compute the summary statistics:

1. *Mean*,
$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j \ ;$$

2. *Standard deviation*,
$$\sigma = \frac{1}{n-1}\sqrt{\sum_{j=1}^{n}(x_j - \mu)^2} \ ;$$

3. *Skewness*,
$$\gamma = \frac{\frac{1}{n}\sum_{j=1}^{n}(x_j - \mu)^3}{\sigma^3} \ ;$$

4. *Kurtosis*,
$$\kappa = \frac{\frac{1}{n}\sum_{j=1}^{n}(x_j - \mu)^4}{\sigma^4} - 3 \ ;$$

5. *Maximum*
$$\max_{j=1,\dots,n}(x_j) \ ;$$

6. *Minimum*,
$$\min_{j=1,\dots,n}(x_j) \ ;$$

7. *Median*,
$$quantile(x, 0.5);$$

8. 1% *quantile*,
$$quantile(x, 0.1);$$

9. 99% *quantile*,
$$quantile(x, 0.99);$$

Where the *quantile* at the level $\alpha \in [0,1]$ is defined as

$$quantile(x, \alpha) = \inf\{u \in \mathbb{R} : \mathbb{P}(\mathcal{X} \leq u) \geq \alpha\} \tag{4.1}$$

where $\mathcal{X}$ is the probability distribution of x. However, this definition requires the analytical knowledge of the probability distribution of our signals, which we do not have. Therefore, the equation 4.1 is estimated using numerical routines. In particular, in this thesis, we use the one available in the *Python* package *Numpy* [32].

Next, we compute the frequency-domain aggregations. First, from every signal x we compute the *Power Spectrum*, by using formula 3.5,

$$s = \mathcal{S}(x) \in \mathbb{R}^{\frac{n}{2}},$$

where the vector s has $\frac{n}{2}$ real components because with our data n is always an even number.

Moreover, we recall that the *Power Spectrum* coefficients can be related to sample frequency with the relation 3.6. Namely, let f be the sampling rate of the sensor from which we obtain the signals, then the *Power Spectrum* coefficient at index k is related to the *sample frequency* $f_k = k \cdot \frac{f}{n}$.

Next, we select a *band width* of $\omega = 100Hz$ and we split the sample frequencies into several contiguous and non-overlapping windows of length given by the *band width*. Then, for each window, we obtain a feature by summing up all the *Power Spectrum* coefficients whose corresponding sample frequencies fall into the window.

Formally, for every $b \in \{1, \dots, \frac{f}{2\omega}\}$ we define the frequency window as the interval

$$\mathcal{J}(b) = \begin{cases} [\omega b - \omega, \omega b) & b < \frac{f}{2\omega} \\ [\omega b - \omega, \omega b] & b = \frac{f}{2\omega} \end{cases} .$$

We observe that the window with index $b = \frac{f}{2\omega}$ is a closed interval because the upper extreme coincides with the largest available frequency.

Finally, we compute the features

$$\phi(b) = \sum_{k: f_k \in \mathcal{J}(b)} s_k, \quad \forall b \in \{1, \dots, \frac{f}{2\omega}\}.$$

As a result, by considering both time-domain and frequency-domain aggregations, for every signal we obtain a total of

$$d = 9 + \frac{f}{200}$$

features. We can then summarize the whole **feature extraction algorithm** in a function

$$\Psi : \mathbb{R}^n \longrightarrow \mathbb{R}^d,$$

that for every signals x gives the vector of aggregations $\Psi(x)$.

4.2 Preprocessing pipeline PRONOSTIA

In *PRONOSTIA* we have data from condition monitoring of rolling bearings through two vibration sensors in 17 run-to-failure experiments, where the bearings are tested in different *operating conditions*, in terms of rotational speed and radial load. In each experiment, we have a sequence of data acquisition batches where the sensors collect information for 0.1 seconds, and since the sensors have a sampling rate of $f = 25.6kHz$, for any data batch we have two signals of $n = 2560$ samples each.

In this thesis, we are interested in investigating *Anomaly Detection* models for rolling bearings fault detection, whose task is detecting failures at early stages by spotting abnormal patterns in the signals. To this extent, we transform the raw data into a well-organized dataset whose rows correspond to a data batch of a given experiment, while the columns are the extracted features from the signals of the two sensors. Indeed, in this context, we can look at the data batches as snapshots of the bearing's health state from the point of view of the two sensors. In addition, among the columns, we consider also metadata information, namely: the *operating condition*, the experiment identifier, and the batch number. We will see that the *operating condition* will be a crucial information for the development of the anomaly detection models of Sections 5.2.2, and 5.2.3. While the experiment identifier is just a piece of information that indicates which experiment corresponds to a given data batch. Finally, the batch number is a progressive number that indicates the order of the data batch in a given experiment.

Formally, for every experiment $e \in E$, where E is the set of experiments, and for every batch $i \in \{1,2,\dots,N_e\}$, where N_e is the number of batches of experiment e, a record of the dataset is given by

$$\left[\; \Psi(x_V^{(e,i)}) \; \vdots \; \Psi(x_H^{(e,i)}) \; \vdots \; metadata \; \right], \tag{4.2}$$

where $x_V^{(e,i)}$ and $x_H^{(e,i)}$ are the signals from the vertical and horizontal accelerometer of experiment e and data batch i, while the *metadata* consist in the experiment identifier, e, the batch number, i, and the operating condition, c. On the other hand, the numerical columns consist of the set of extracted features from the signals of the two accelerometers. Overall, since the sampling rate is $f = 25.6kHz$ we obtain a total of

$$2 \cdot d = 2 \; \cdot \left(9 + \frac{f}{200}\right) = 2 \; \cdot (9 + 128) = 274.$$

By repeating the procedure for all batches and all experiments we obtain a dataset $\mathcal{D}$ of 277 columns and 24018 rows.

4.3 Preprocessing pipeline CWRU

In *CWRU* dataset we have a set of experiments where intact and defective rolling bearings are tested under different operating conditions, in terms of rotational speed. A vibration sensor monitors the bearings' execution, with a sampling rate of $12kHz$. Since the experiments' duration ranges from 20 seconds to 152, every signal contains a huge number of samples. Therefore, we decide to split each signal into time windows of 2560 samples each, which correspond to around 0.213 seconds of monitoring.

In this section, we describe how the raw signals from *CWRU* are transformed into a well-organized dataset, whose row corresponds to the time windows of the experiments, while the columns contain the extracted features as well as metadata information.

Formally, for every experiment $e \in E$, where E is the set of experiments, and for time window $i \in \{1,2,\dots,N_e\}$, where N_e is the number of time windows in experiment e, we obtain a record of the dataset is given by

$$\left[\; \Psi(x^{(e,i)}) \; \vdots \; metadata \; \right], \tag{4.3}$$

where $x^{(e,i)}$ is the signal from the vibration sensor of experiment e and data batch i, while the *metadata* consists of the operating condition, the anomaly label, and the failure type. The anomaly label is a qualitative indicator that discriminates whether the current record refers to an intact or a defected bearing. On the other hand, since the sampling rate is $f = 12kHz$ the feature extraction algorithm leads to a total of

$$d = \left(9 + \frac{f}{200}\right) = (9 + 60) = 69.$$

By repeating the procedure for all time windows and all experiments we obtain a dataset $\mathcal{D}$ of 72 columns and 2589 rows.

Chapter 5

Models

In the context of *Predictive Maintenance*, **Anomaly Detection** models can be used to continuously assist the production: they monitor the signals sampled from *IoT sensors*, and raise warnings when they find abnormal patterns. To this extent, we have to train a Machine Learning model, so that it recognizes ordinary as well as anomalous conditions. However, supervised methods are typically unfeasible, due to the extreme rarity of anomalies in historical data, and therefore, *unsupervised* methods become often the only viable strategy.

In this chapter, we discuss **unsupervised Anomaly Detection** models for fault detection. Such algorithms leverage data from ordinary working conditions for understanding which are the schema and the patterns that characterize the monitoring of ordinary production, and complementary they detect anomalies.

To this extent, in Section 5.1, we introduce methods and algorithms that will be part of the proposed Anomaly Detection models. Afterward, in Section 5.2 we present three different *Anomaly Detection* models. Specifically, in Section 5.2.1, we propose an Anomaly Detection algorithm based on reconstruction errors of an Autoencoder. Then, in Section 5.2.2, we link *Anomaly Detection* with the *Open set Recognition* framework by proposing a method that exploits classification probabilities for detecting anomalies. Finally, in Section 5.2.3, we proposed a novel approach based on more sophisticated *Open Set Recognition* techniques taken from [49], that leads to the definition of an interpretable condition indicator and overcomes the necessity of computing thresholds for predicting anomalies.

The *Open Set Recognition* inspired methods are developed by considering a scenario where the machine can work under different operating conditions, which is quite common in the industrial context. In particular, in the thesis, we have investigated the algorithms in condition monitoring of rolling bearings that can work with different rotating speeds (See Chapter 2). However, we can apply such algorithms in general for every kind of machine where we can recognize distinct operating modes or working stages. For example, think about a machine that can produce different kinds of pieces by changing its operating mode, or a when production line follows a sequence of steps where distinct operations are carried out.

5.1 Preliminary concepts

5.1.1 Neural Networks

A **Neural Network** [15] is a network of connected units, called *neurons*; each neuron is an elementary calculation unit, which is able to send its results to the other cells according to a precise graph structure. Every neuron and every connection have free parameters, called *weights*, that can be fitted in order to tackle a given task. A set of neurons constitutes the input of the network, and some other neurons indicate the output. The remaining layers, instead, form the computational core of the neural network, that is, the ensemble of the mathematical operations that transform the input into output.

Neurons can perform different kinds of computations, and they send information to other neurons by means of weighted links. The outgoing links indicate where the computations are feed, while incoming links indicates the neuron inputs. The weights are object of inference and are determined by model training.

The simplest Neural Network architecture is the Feed Forward Neural Network (see Chapter 6 of [15]), where neurons are arranged in a set of layers and there are connections only between a layer and the consecutive one, moreover, each neuron performs a linear transformation followed by a non-linear activation function. Formally, let I be the number of layers and let m_i be the number of neurons in layer i, then the j-th neuron of the i-th layer perform the following computation

$$x_j^{(i)} = f\left(b_j^{(i)} + \sum_{k=1}^{m_{i-1}} w_{jk}^{(i)} x_k^{(i-1)}\right), \quad \forall j \in 1,2,3,...,m_i,$$

where $f(\cdot)$ is a non-linear activation function, $b_j^{(i)} \in \mathbb{R}$ is the bias for the neuron j , $w_{jk}^{(i)} \in \mathbb{R}$ is the link weight between neuron k of layer $i-1$ and neuron j of layer i, and $x_j^{(i)} \in \mathbb{R}$ is the output of neuron j of layer i.

More compactly, we can arrange the biases in a vector $b^{(i)} \in \mathbb{R}^{m_i}$, where the j-th component is $b_j^{(i)}$; the link weights in a matrix $W^{(i)} \in \mathbb{R}^{m_i \times m_{i-1}}$, where $W_{j,k}^{(i)} = w_{jk}^{(i)}$; the inputs in a vector $x^{(i-1)}$ where the k-th entry is $x_k^{(i-1)}$; then the computation of layer i can be written as,

$$x^{(i)} = f\left(b^{(i)} + W^{(i)} x^{(i-1)}\right),$$

where $f(\cdot)$ is computed element-wise. As an activation function we mention the sigmoid function (see Figure 5.1), that for any input $x \in \mathbb{R}$ it gives

$$f(x) = \frac{1}{1+e^{-x}}. \tag{5.1}$$

Since in this kind of layer, every neuron of the preceding layer is connected to each neuron of the following one, they are known as **dense layers**.

The biases and the weights are parameters and they are learned during training by minimizing a **loss function**, which is a metric that penalizes the parameters when the

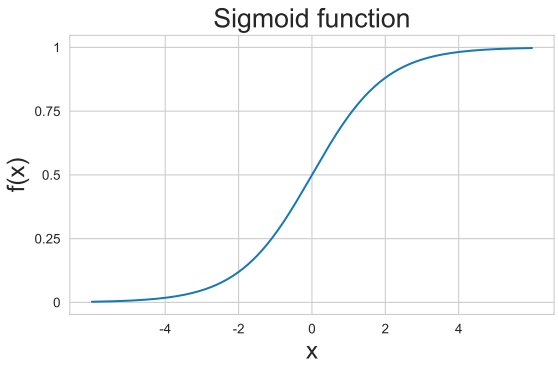

Figure 5.1. The sigmoid function.

model makes errors in a given task, e.g. classification or regression. Formally, we can indicate with Θ the set of parameters of the network and we can denote the model with $F(\cdot)$, that takes as input a vector x and gives as output $F(x)$. For clearness, during the discussion, we consider supervised learning tasks only. That is, for any data point x, we have a class label or response variable y and the learning task is to understand the functional relation between them. Formally, for every couple (x, y) we want to fit the set of weights Θ in order to satisfy the equation

$$F(x) = y; \tag{5.2}$$

the loss function will then take a form that during training it will encourage the adjustment of the weights Θ, so that the model satisfies equation 5.2, we denote it as

$$l(F(x\,;\Theta), y).$$

The notation $F(x\,;\Theta)$ strengthens the fact that the network outputs depend on the current values of the parameters Θ.

Next, instead of considering a single data point we consider a dataset of $n \in \mathbb{N}$ instances, that we denote with $\mathcal{D} = (x_i, y_i)_{i=1}^n$; then, the loss function computed over the dataset $\mathcal{D}$ is given by the average of the losses compute on every couple of points,

$$L(\mathcal{D}; \Theta) = \frac{1}{n} \sum_{i=1}^{n} l(F(x_i; \Theta), y_i).$$

Finally, the parameters Θ are obtained by minimizing the loss function, that is

$$\min_{\Theta \in \mathcal{X}} L(\mathcal{D}; \Theta), \tag{5.3}$$

where $\mathcal{X}$ is the feasible set where the parameters can range. Usually, the optimization is unconstrained, that is $\mathcal{X} = \mathbb{R}^{|\Theta|}$ where $|\cdot|$ is the cardinality of a set. On the other hand, the feasible set can also be constrained in order to limit the model capacity, but this is outside the scope of this thesis and we will always consider unconstrained optimization.

The problem 5.3 is tackled with *Stochastic Gradient Descent* (*SGD*), a numerical optimization algorithm that iteratively updates the parameters according to the gradient computed over random subsets of the dataset. Formally, at each iteration or epoch, we randomly split the dataset into a sequence of subsets or batches, that is $\mathcal{D} = \{\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_b\}$ where b is the number of batches. Then, iteratively for every $j = 1,2,\dots b$ we update the parameters as follows

$$\theta = \theta - \alpha \frac{\partial L(\mathcal{B}_j; \Theta)}{\partial \theta}, \quad \forall \theta \in \Theta,$$

or with a vector notation,

$$\Theta = \Theta - \alpha \nabla (L(\mathcal{B}_j; \Theta)), \tag{5.4}$$

where $\nabla(\cdot)$ is the gradient operator and α is an hyper-parameter called *learning rate* used for controlling the updating speed, and it is typically set to small values, e.g. 0.001.

The interpretation of the formula 5.4 is that for each step and for each batch the parameters are updated such that they give the greatest local decrease in the loss function over the corresponding data batch. For that reason, this is a quite natural and simple update rule but it has also some defects. First, the gradient indicates the direction where the function locally decreases most but it does not tell to what extent we can make a step in this direction in order to gain a (global) decrease: if the learning rate is too high we can even end up in an increase. Second, the gradients are computed independently at each step and batch and we do not take into account the history of the previous steps. For these reasons, researchers propose different optimization algorithms or optimizers to obtain better results: in this thesis, we will use **Adam** optimizer [19]. An exhaustive description of the optmizer is outside the scope of the thesis, but it worth noting that it basically consist in a variant of the formula 5.4, which can be written

$$\Theta = \Theta - \alpha \cdot d(\mathcal{B}_j; \Theta),$$

where $d(\mathcal{B}_j; \Theta)$ is called descent direction, and it take the place of the gradient in formula 5.4. The formulation of $d(\mathcal{B}_j; \Theta)$ contains the gradient, but also other quantities that improve the overall optimization process.

The update process is repeated for a given number of epochs. The data batches have all the same size; except for the last one: whenever the dataset size is not a multiple of the batch size, the last batch contains only the number of samples which is not yet included in the previous batches. The number of epochs as well as the batch size are hyper-parameters. The structure of the neural network is pretty general, and allows us to tackle different tasks by selecting a suited loss. For example, we can address a regression task by using the *Mean Squared Error* loss,

$$L(\mathcal{D}; \Theta) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2,$$

where $y_i \in \mathbb{R}$ is the response variable and $\hat{y}_i \in \mathbb{R}$ is the model prediction, of the sample i; or we can tackle a classification problem by selecting the *Cross Entropy* loss,

$$L(\mathcal{D}; \Theta) = -\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(\hat{y}_i = y_i) \log(\mathbb{P}(\hat{y}_i = y_i)), \tag{5.5}$$

where y_i is the class label, $\hat{y}_i$ is the predicted class, $\mathbb{1}(\hat{y}_i = y_i)$ has value 1 when the predicted class coincide with the actual one, and 0 otherwise; $\mathbb{P}(\hat{y}_i = y_i)$ is probability of predicting the correct class, which is estimated by the model.

In general, we can consider any kind of problem. That is, with a proper neural network architecture and a proper loss we can learn any kind of functional relation between and outputs. A relevant example is when $y = x$, where the network try to learn the identity map; this framework forms a class of neural network models called **autoencoder** 5.1.1. Differently, from what we have described so far the learning task is now unsupervised, because of the response variable is the input itself.

Independently from the learning task, what we want to avoid in neural networks, and more in general in Machine Learning models, is *overfitting*: the model is perfectly fit training data but it has no generalization capacity for new data. The literature propose a lot of techniques for address this issue. In this thesis, we consider **dropout** [43]: during training some neurons are randomly dropped, preventing the network to rely too much on specific neurons. The noise injected during training leads to a more robust model by significantly reduces the overfitting.

In the following, we describe how a dropout layer works. Let we consider a dropout probability $p \in [0,1]$, that indicates the probability of activating a given neuron, and suppose that the input has dimension n; then we define the dropout vector as

$$z = \frac{1}{p} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}, \quad r_i \sim Bernoulli(p),$$

where the quantities r_i assume value 1 with probability p and 0 with probability $(1 - p)$, moreover, they are randomly sampled during each prediction in the training phase. The vector z multiplies the input x by performing a product element-wise

$$x \cdot z.$$

Thereafter, when the training phase is over the dropout vector end its random behaviour and it keeps the input data untouched. Formally, the dropout vector is set to

$$z = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Autoencoder

An **autoencoder** is a neural network that is trained for learning the identity map, by a compression-decompression procedure. Indeed, it can be viewed as consisting of two parts: the **encoder** that learns how to encode an input with a smaller number of features, and

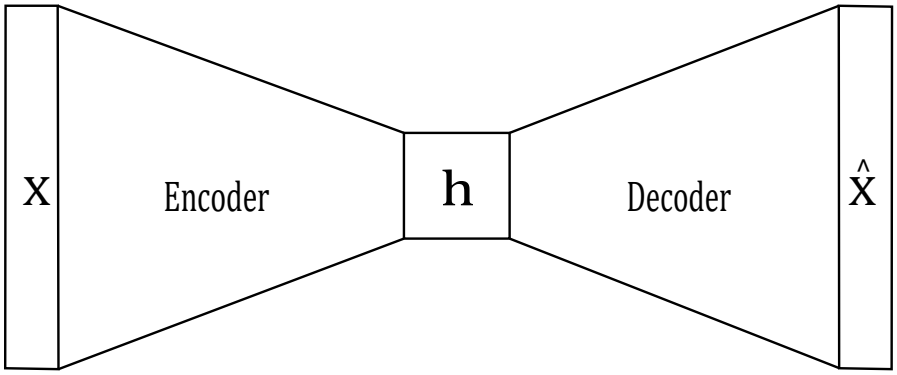

Figure 5.2. An autoencoder.

the **decoder** that learns how to get back to the original input. The autoencoder is the composition of the two blocks, see Figure 5.2. The model is learned by minimizing the error obtained during reconstruction.

Formally, let d and p be the input and the encoding dimensions, then the **encoder** is the function

$$f_{ENC} : \mathbf{R^d} \longrightarrow \mathbf{R^p},$$

while the **decoder** is the function,

$$f_{DEC} : \mathbf{R^p} \longrightarrow \mathbf{R^d},$$

finally, the **autoencoder** is the composition of the two blocks, that is

$$f = f_{DEC} \circ f_{ENC} : \mathbf{R^d} \longrightarrow \mathbf{R^d}.$$

The encoder output $h = f_{ENC}(x)$ is known as **code** or **latent representation**. It is the innermost part of the network, and it contains the smallest number of neurons: it then acts as the *bottleneck* of the architecture. By construction it extract the most relevant information that data contain as they are used by the decoder to restore the whole input. Therefore, it can be then used for *dimensionality reduction*.

Let $x \in \mathbf{R^d}$ be an input vector and $\hat{x} = f(x)$ be the autoencoder output, the difference

$$\mathcal{E}(x) = \hat{x} - x \in \mathbf{R^d}$$

is called **residual vector**, and measure how far the reconstruction deviates from the original input element-wise: we get a good reconstruction if the residuals are almost zero.

With this extent, we can define loss functions that penalize the residuals magnitude: for example we can consider the **Mean Squared Error**, computed over a set of data points $\mathcal{B} = \{x_1, x_2, \dots x_n\}$, as

$$L_2(\mathcal{B}) = \frac{1}{n}\sum_{i=1}^{n} ||\hat{x_i} - x_i||_2^2, \tag{5.6}$$

where the terms,

$$||\hat{x_i} - x_i||_2^2 = \sum_{j=1}^{d}([\hat{x_i}]_j - [x_i]_j)^2, \quad i = 1,2,\dots n,$$

are the *Euclidean norm* of the residual vectors and they are called **reconstruction errors**. The minimum of the function is reached when all inputs are perfectly reconstructed,

$$\hat{x_i} = x_i = f(x_i), \quad \forall x_i \in \mathcal{B}, \tag{5.7}$$

or else when the autoencoder is able to reproduce the identity map for every sample in $\mathcal{B}$.

However, being able to perfectly reproduce a restricted set of input data is far from being a useful task, and it corresponds to overfitting. It is more useful to learn the equations 5.7 only approximately but being able to generalize the reconstruction capability also to new data: this can be obtained for example by limiting the model parameters or adding regularization.

In this thesis, we use autoencoders for **anomaly detection** and **dimensionality reduction**. The encoder allows to compress data in a lower dimensional space, the encoded features brings all information that the decoder needs for restoring the original input. By choosing the network design we can decide:

- how many neurons use for encoding data,
- the complexity of the encoding (and decoding) transformation.

Indeed, we can obtain the lower dimensional representation by an arbitrarily high number of non-linear transformations in sequence. As a result, we can find hidden and complex relationships between input features and decrease the dimensionality without loosing too many information.

By learning how to summarize and reconstruct inputs, an Autoencoder learns the most intrinsic aspects of data, obtaining an high level abstraction that characterize the data it has learned. Furthermore, the minimization of the Mean Squared Error loss 5.6 leads the model not only to obtain minimal reconstruction errors for training instances but also to get a small error for new instances sampled from the same probability distribution.

On the other hand, if the new instances show relevant difference with training data, the model will find difficulties during reconstruction, ending up in a high reconstruction error. We can use this idea for anomaly detection:

- we estimate a level where reconstruction error normally ranges;
- we predict an anomaly whenever the reconstruction error is otside the range.

Autoencoders are a fairly general class of models because both encoder and decoder can have an arbitrary neural network architecture. For that reason, Autoencoders can work with any kind of data and tackle also different problems from anomaly detection and dimensionality reduction.

However, a general rule is making the autoencoder symmetric with respect to the bottleneck layer, that is, the encoder and the decoder implement the same transformations but in opposite directions. As a result, the model is encouraged to perform symmetric operations for encoding and then decoding input data. Indeed, it is quite natural to think that the compression process should be (approximately) a reversible transformation: from the encoded information we should get back to the original input by doing the same operation backward with a minimum error.

The choice of architecture for the two blocks usually depends on the kind of data we have. For example, if we deal with images we should consider *convolutional autoencoder*, where the encoder and the decoder are convolutional neural networks (see chapter 9 of [15]); or if we deal with raw time series we should consider *recurrent autoencoders*, that is where the two blocks are recurrent neural networks (see [15] at chapter 10). A more general architecture that works with any kind of data consist of simply stacking dense layers, in this case, we call just *autoencoders*.

5.1.2 eXtreme Gradient Boosting

In this section we present an efficient Gradient Tree Boosting method, known as **eXtreme Gradient Boosting** (**XGBoost**) [5].

A Gradient Tree Boosting method is an algorithm that learns an ensemble of decision trees [3] via local approximations. The idea of ensemble learning is that simple weak models can create a single stronger model. In particular, with boosting [13], the weak models are learned sequentially, so that at each step the new model is learned for correcting the mistakes of the previous ones.

In **XGBoost** the trees are learned by minimizing an objective function approximated via second-order Taylor expansion, where the objective function contains a task-specific loss and a regularization term. Indeed, *XGBoost* models can learn any kind of task, the only requirement is that the objective function must be differentiable.

In the following, we will explain the model by considering a binary classification problem, but statements can be generalized to an arbitrary one. Formally, we consider a dataset $\mathcal{D} = \{(x_i, y_i) \in \mathbb{R}^n \times \{0,1\}\}_{i=1}^m$, for any sample $x_i \in \mathbb{R}^n$ the model uses K trees to predict the output

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F},$$

where $\mathcal{F} = \{f(x) = w_{q(x)}, \quad q : \mathbb{R}^n \to T, \quad w \in \mathbb{R}^T\}$ is the functional space of the decision trees, T is the number of leaves of the tree, the function $q(\cdot)$ denotes the tree structure, that for any sample x gives as output the predicted leave, and $w_{q(x)}$ is the probability of predicting class 1 for the sample x.. Moreover, the trees are not independent, and they are related by the equations

$$\hat{y}_i^{(k)} = \hat{y}_i^{(k-1)} + f_k(x_i), \quad \forall k \in \{1,2,\dots,K\}, \tag{5.8}$$

where $\hat{y}^{(k)}$ is the prediction at step k for the sample x_i.

The training algorithm performs a sequence of K steps, one for each tree, where at each step, k, we consider the objective function

$$L^{(k)}(\mathcal{D}\ ; f_k) = \sum_{i=1}^{m} l(y_i, \hat{y}_i^{(k-1)} + f_k(x_i)) + \Omega(f_k), \tag{5.9}$$

where $l(y, \hat{y})$ is a classification loss and $\Omega(f)$ is a regularization term. The notation $L^{(k)}(\mathcal{D}\ ; f_k)$ enforces the fact that at each step, only the parameters of the k-th tree are optimized.

The equation is then approximated via a second order Taylor expansion, obtaining

$$L^{(k)}(\mathcal{D}\ ; f_k) \approx \sum_{i=1}^{m} \left[l(y_i, y_i^{(k-1)}) + g_i f_k(x_i) + \frac{1}{2} h_i f_k(x_i)^2 \right] + \Omega(f_k), \tag{5.10}$$

where

$$g_i = \left[\frac{\partial l(y, \hat{y})}{\partial \hat{y}} \right]_{(y,\hat{y})=(y_i, \hat{y}_i^{(k-1)})},$$

and

$$h_i = \left[\frac{\partial^2 l(y, \hat{y})}{\partial \hat{y}^2} \right]_{(y,\hat{y})=(y_i, \hat{y}_i^{(k-1)})}.$$

Finally, at each step, the approximated objective function 5.10 is minimized for training the k-th tree, by learning the tree structure and the leaves' weights.

If we select the function,

$$\Omega(f) = \lambda \sum_{j=1}^{T} w_j^2, \tag{5.11}$$

as regularization term, where λ is an hyper-parameter; we can show (see [5]) that for any tree structure $q : \mathbb{R}^n \to T$, the optimal leaves weights $w \in \mathbb{R}^T$ are given by

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \quad \forall j \in \{1, \dots, T\},$$

where $I_j = \{i \in \{1, \dots, m\} : q(x_i) = j\}$, and the corresponding optimal value is

$$L^*(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda}.$$

Therefore, the last quantity can be seen as a score for evaluating a given tree structure.

However, finding the optimal structure of a decision tree is known to be a hard combinatorial problem that cannot be solved exactly [20]. *XGBoost* exploits numerical procedures that efficiently find a sub-optimal solution; in particular, it uses the weighted quantile sketch as an approximation algorithm for split finding, and cache access patterns, data compression, and sharding for obtaining efficient management of the computational cost

both in terms of memory and time. The optimization details are outside the scope of the thesis and are reported in [5].

Moreover, the algorithm can be improved by adding additional features for managing the overfitting. We can introduce the shrinkage coefficient,

$$\eta \in [0,1],$$

to the equations 5.8, by obtaining,

$$\hat{y}_i^{(k)} = \hat{y}_i^{(k-1)} + \eta \cdot f_k(x_i), \quad \forall k \in \{1,2,\dots,K\}. \tag{5.12}$$

As a result, η reduces the influence of each single decision tree and leave room for future trees to improve the model. We can also limit the depth of the trees, by limiting, therefore, the feasible set of the decision trees. Otherwise, at each step, we can learn the tree by considering a random subset of dataset. Formally, we modify the equation 5.9 as follow

$$L^{(k)}(\tilde{\mathcal{D}}\,; f_k) = \sum_{i=1}^{m} l(y_i, \hat{y}_i^{(k-1)} + f_k(x_i)) + \Omega(f_k), \quad \tilde{\mathcal{D}} \subseteq \mathcal{D}.$$

In this thesis, we will consider *XGboost* for a multi-class problem, in which the formulation that we have presented is extended by considering a vector of probabilities rather than a single number in each leaf of the trees. Moreover, at each step, the tree structure is optimized by considering the Cross Entropy loss (see equation 5.5).

5.1.3 Open Set Recognition

Consider a trained classification model and imagine that we have discovered that it exists a new class that was not present during training; if we do not retrain the model and we ask to predict samples from this new class, it will give us predictions from the set of the old classes which are naturally wrong; while we would prefer that the model would be able to recognise that those samples belong to a class which is different from all of the known ones. This idea leads to the definition of **Open Set Classifiers**, some classification models able of

1. recognising the known from the unknown,
2. classifying among the known classes;

that is, in addition to the standard classification task it is able to detect whenever a concept is outside of its bag of knowledge.

More formally, let $\mathcal{C}$ be the set of classes available during training and let c' be the new discovered class. For any new sample x of class c' a standard classification model will always predict as class $c \in \mathcal{C}$, while an open set classifier will tell us that the class is unknown. We can formalize this concept by adding to the set of classes a new auxiliary class label, $\{unk\}$, that indicates the unknown,

$$\mathcal{C} \cup \{unk\}.$$

A closed set classifier can be easily reformulated for the Open Set scenario. Let's consider a probabilistic classifiers, that for any instance x gives as output the probability distribution across the classes

$$p = f(x) \in \mathbb{R}^{|\mathcal{C}|},$$

where $f(\cdot)$ is the model itself. Then the standard prediction strategy is to consider the class with maximal probability,

$$\hat{y} = \underset{i \in \mathcal{C}}{argmax}(p_i)$$

We can say that we accept this prediction only if the corresponding maximum probability is above some level of confidence, for example we predict as $\hat{y}$ only if we have a probability of 0.9. Formally,

$$\hat{y} = \begin{cases} \underset{i \in \mathcal{C}}{argmax}(p_i) & \text{if } \underset{i \in \mathcal{C}}{max}(p_i) \geq \tau \\ \{unk\} & \text{otherwise} \end{cases}$$

where τ is the threshold that selects the level of confidence, and can be set to any values in [0,1]. Usually, it make sense to require at least a probability of 0.5, but we can be more conservative by raising the threshold to higher values like 0.8 or 0.9. However, the right threshold depends on the specific problem and the corresponding data, and there can be more sophisticated strategy for selecting a good threshold.

Of course this is just the simplest strategy for obtaining an open set classifier, and it is far from being ideal. More complex approaches generally consist in the developing of classification models that by design take into account the presence of the unknown class. Here, instead we have just resorted the model outputs.

5.2 Proposed methods

In this section, we present 3 different Anomaly Detection models, and we assume to have a portion of data devoted to learn the models, which can be divided into two portions: the *training set* and the *validation set*. Moreover, both portions contain only normal data, that is data from condition monitoring when the bearings have no defects.

5.2.1 Autoencoder Anomaly Detection

In this thesis, we will use an autoencoder for **unsupervised anomaly detection** of signals from condition monitoring of rolling bearing: we fit an autoencoder with normal training data, and we then use the model to obtain reconstruction errors of new samples and we predict an anomaly whenever the error is above some well defined level. First, we go into details of the selected autoencoder architecture and we highlight the model's parameters. Next, we describe how the model is trained. Then, we explain how to define a reconstruction error threshold. Finally, we explicit the anomaly detection algorithm.

Since our data has tabular format (see Chapter 4), it is suitable to choose a simple architecture with 5 dense layers for our autoencoder, as reported in Figure 5.3. The encoder is made up of the first 3 layers: the third layer is the bottleneck layer; the last 3

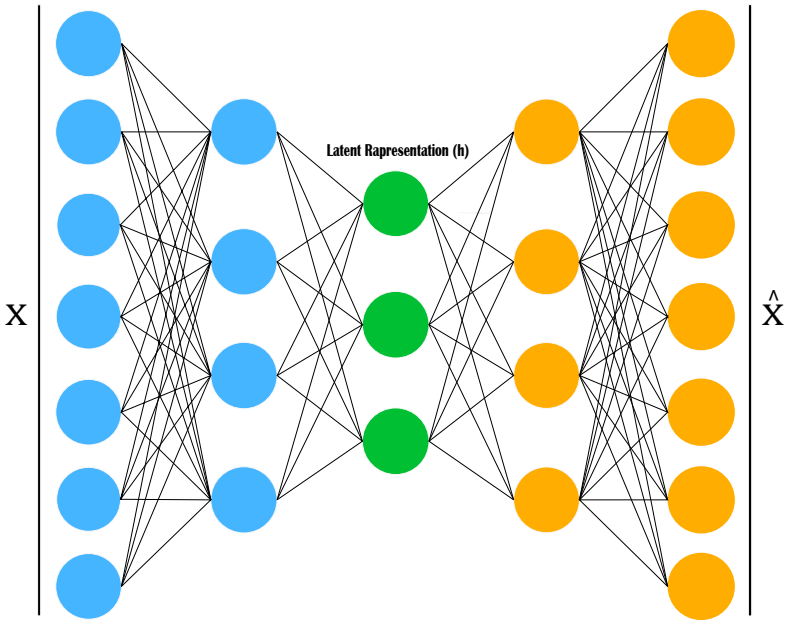

Figure 5.3. An Autoencoder with 5 dense layers.

layers forms the decoder. The inputs enter into the network from the first layer, while the output are obtained from the last network. Formally, it computes the following operations,

$$\begin{aligned} h_0 &= x \\ h_1 &= \sigma(W_1 h_0 + b_1) \cdot z_1 \\ h_2 &= W_2 h_1 + b_2 \\ h_3 &= \sigma(W_3 h_2 + b_3) \cdot z_2 \\ \hat{x} &= W_4 h_3 + b_4 \end{aligned}$$

where h_i is the output of layer i, $h_0 = x$ is the input while $\hat{x}$ is the output; W_i and b_i are the coefficient matrix and the bias vector that defines the linear transformation of layer i , $\sigma(\cdot)$ is the sigmoid activation function, while z_1 and z_2 are dropout vectors (see Section 5.1.1). The parameters of the model are W_i and b_i, for $i = 1,2,3,4$; their dimensions meet the input-output requirement of each layer, that is, if d_i is the output dimension of layer i, we have that

$$W_i \in \mathbb{R}^{d_i \times d_{i-1}}, \quad b_i \in \mathbb{R}^{d_i}, \quad i = 1,2,3,4.$$

The dimensions of the inner layers d_1, d_2 and d_3 are model's hyper-parameters, and are chosen by trials and errors. The chosen configuration is given by:

- $d_1 = d_3 = 40$,
- $d_2 = 5$.

The bottleneck dimension d_2 is set to a small value to make the learning task harder and force the model in learning interesting lower dimensional representations. The hidden dimensions d_1 and d_3 are set to the same value to attain network symmetry. Moreover, the value of 40 is selected as a good trade-off between the bottleneck dimension of 5 and the input dimension (which for our data is between 69 and 274, see 4).

In the dropout layers we select a probability of 0.01 for mitigating the overfitting.

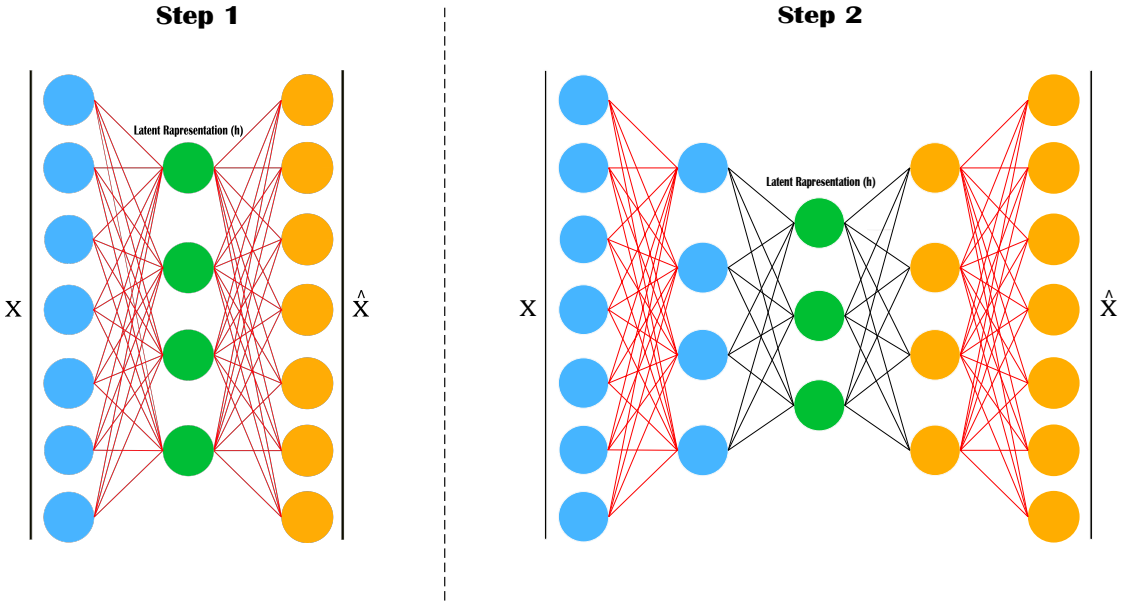

Figure 5.4. The figure reports a sketch of the architectures considered in the two steps training procedures. The red-colored arcs indicate that the weights learned in step 1 are used for initializing the same parameters in step 2.

We learn the weights by minimizing the Mean Squared Error loss (see formula 5.6); as an optimization algorithm we select the *Adam* optimizer with learning rate $\alpha = 0.001$, and $b = 64$ samples as batch size.

The training phase considers the training data which do not contain anomalies, and it consists of two sequential steps, see Figure 5.4. In step 1, we consider a subset of the architecture just described, by selecting the layers 0, 1, 3 and 4, where the two layers of size 40 are collapsed to a single one, that becomes the bottleneck of the new architecture. As a result, this sub-model is a smaller Autoencoder where the latent representation is made by a layer of 40 neurons. Formally, the sub-model compute the following operations,

$$\begin{aligned} h_0 &= x \\ h &= \sigma(W_1 h_0 + b_1) \\ \hat{x} &= W_4 h + b_4 \end{aligned}$$

We train this sub-model for 250 epochs, and we obtain the parameters,

$$W_1, W_4, b_1, b_4.$$

Then, in step 2, instead of randomly initializing all weights, we use the obtained parameters for initializing the shared weights, and we train the whole architecture for 1500 epochs.

After that, we consider validation portion of data for estimating a reconstruction error threshold. Since this portion contains only normal data we would define the threshold as the upper extreme where most of the reconstruction errors ranges. Specifically, for every instance x in the validation set we compute the reconstruction error,

$$e(x) = ||\hat{x} - x||_2^2 = \sum_{i=1}^{d} (\hat{x}_i - x_i)^2,$$

and we estimate a threshold as the 99% quantile of the empirical distribution of the reconstruction errors,

$$\mathcal{T} = \textit{quantile}(\mathcal{E}, 0.99),$$

where $\mathcal{E}$ is the vector that collects all reconstruction errors on validation data.

The use of the validation portion data rather than training one allows for a more reliable estimate of the reconstruction error. Indeed, validation instances are not seen by the model during training, so they reflect more faithfully the new samples that will come. Moreover, the use of the quantile rather than the maximum leads to a more robust estimate, with respect to the presence of possible outliers.

Finally, the **Anomaly Detection algorithm** work as follow: for any new sample x

1. we feed x into the Autoencoder and we obtain the reconstruction $\hat{x}$,
2. we compute the reconstruction error $e(x) = ||\hat{x} - x||_2^2$,
3. if $e(x) > \mathcal{T}$ we predict an **anomaly**,
4. otherwise we predict as **normal**.

5.2.2 Operating Condition Classification Anomaly Detection

The model that we have described in the previous section is an unsupervised approach for Anomaly Detection and it does not exploit any kind of additional knowledge about the data. Actually, from Section 3.2, we know that the bearings work in 3 distinct operating conditions: can we exploit this information to build a more anomaly detection algorithm?

Distinct operating conditions correspond to the different physical conditions of the underlying system, and then to different data-generating processes. From a theoretical point of view data from distinct operating conditions are generated from different underlying probability distributions. Consequently, a classification model can easily discriminate between operating conditions. On the other hand, from exploratory analysis (see Section

[3.2.1]) we know that frequency features are exposed to *data drift* due to natural bearing degradation; that is, the system degradation causes a change in the data generating process, which is evident in the frequency domain.

Moreover, we know that in each experiment the corresponding bearing work in the same operating condition from the beginning to the end, then a classification model should be able to predict always the same class when it is monitoring a given experiment. However, we expect that data drift will cause a strong reduction in model performances: when the bearing wears out the model will find data with different patterns from the ones discovered during training, and consequently, the probability of being in the correct operating condition class decreases. The more the defect are evident the more we expect that the classification probability decreases.

We can therefore exploit this hypothetical behavior for Anomaly Detection: for every experiment we know in which operating condition it runs, so we can monitor the classification probability of being in the correct class, and predict an anomaly when it falls below a safe level. Indeed, a decrease in such probability can only be due to the bearing degradation because the operating condition does not change during the experiment. Moreover, is natural to think that in most situations bearing's failure is preceded by a slow degradation process; for example, the bearing's surface becomes scratched, the scratch expands, and then it becomes more and more deep until it ends up in a crack in the ring. For that reason, we expect a progressive and monotonic reduction in the classifier performance as a progressive decrease in the classification probability of the correct class.

Interestingly, the classifier is exploited in an unusual way: we do not use it for predicting new labels but instead we monitor the classification probabilities to detect anomalies in correspondence of performance decrease. Indeed, the class label is an information that we we always know in advance, because in each experiment the bearings run in the same operating condition from the beginning to the end. More in general, the operating condition when a machine works can change during but it is set by a technician or by an automated software; in any case it is a condition that is defined by the production scheduling rather than something we should infer. Similarly, rather than trying to mitigate data drift with domain adaptation techniques, we exploit it to individuate a degradation process.

In the following, we describe the steps we make for building an anomaly detection model that exploits the classification probabilities and the insights that we have explained so far. First, we define the classifier that we have chosen. Then, we describe how we can compute classification probabilities thresholds. Finally, we explicit the anomaly detection algorithm.

As classification model choose the **XGBoost classifier** (see Section5.1.2), with $K = 500$ trees and shrinkage parameter $\eta = 0.3$. Moreover, we consider 3 hyper-parameters that are chosen with 10 fold cross-validation, namely:

- the regularization coefficient, λ, of formula 5.11;
- the depth of each tree;
- the subsample ratio that is used for randomly sampling a portion of the dataset at each training step, by considering only a data subset in equation 5.10.

The Table 5.1 report the hyper-parameters grid. After cross-validation, the *XGBoost classifier*, with the selected hyper-parameters configuration, is trained by considering the *training set*.

Parameter	**Values**
Maximum Depth of a tree	$\{3, 6, 9\}$
Subsample ratio	$\{0.5, 0.75, 1.0\}\}$
λ	$\{0, 2.5, 5\}$

Table 5.1. Hyper-parameters grid.

Let we denote with $f(\cdot)$ be the *XGboost* model, that for every sample x it gives as output a probability vector,

$$\hat{p} = f(x) \in \mathbb{R}^{|\mathcal{C}|},$$

where $\mathcal{C}$ is the set of classes, which in our case is $\mathcal{C} = \{1,2,3\}$.

Then for any data point $(x, c) \in \mathbb{R}^d \times \mathcal{C}$ in the validation set,

- we compute the predicted probability vector $\hat{p} = f(x)$,
- we consider the correct class probability $\hat{p}_c$,
- we accumulate such probability in the set $\mathcal{P}_c$.

As a result, $\mathcal{P}_c$ contains all of the predicted probability related to the correct class for any class c.

For any class c, we compute the threshold as the 1% quantile of the probability values in the set $\mathcal{P}_c$,

$$\tau_c = \textit{quantile}(\mathcal{P}^c, 0.01), \quad \forall c \in \mathcal{C}. \tag{5.13}$$

As a result, for any class the thresholds are the lower extreme where of most the probabilities for normal instances range. Finally, the **Anomaly Detection algorithm** works as follow: for any new sample (x_i, c_i)

1. we compute predicted probability $\hat{p} = f(x_i)$,
2. we select the probability of being in the correct class $\hat{p}_{c_i}$,
3. if $\hat{p}_{c_i} < \tau_c$ we predict an **anomaly**,
4. otherwise we predict as **normal**.

Once more, we remark that we can consider the correct class probability because we always know in advance the operating condition where the system runs.

The approach is quite simple and it is similar to the Autoencoder, but it leverages classification probability rather than reconstruction errors. Moreover, here we have a vector of threshold while with the Autoencoder we had just a single number.

Interestingly, we can note a strong analogy between this algorithm and the *Open Set Recognition* framework, described in Section 5.1.3: the thresholds we use for detecting anomalies are, actually, rejection thresholds. Indeed, they are the levels of probability required by the model to make a prediction among the known classes. As a result, we have built an open set classifier where the unknown class should be intended as the anomaly class. The only difference is that here we use the correct class probability while in Section 5.1.3 we have used the maximum one.

5.2.3 Classification-Reconstruction Anomaly Detection

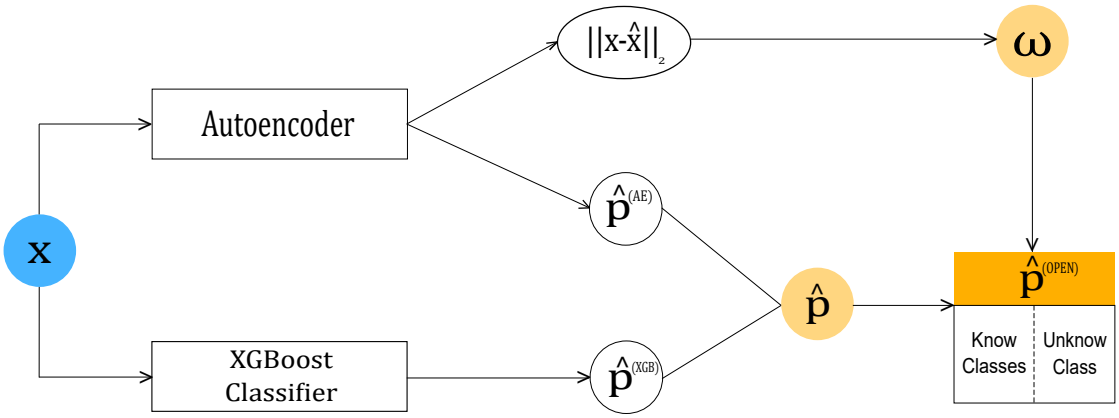

Figure 5.5. The Classification-Reconstruction Anomaly Detection model: $||x - \hat{x}||_2$ is the reconstruction error (see formula 5.17); $\hat{p}$ is the probability distribution of the operating conditions (see formula 5.15), from the ensemble classifier; ω are the class confidence weights (see formulas 5.19); finally, $\hat{p}^{(OPEN)}$ is the model output (see formula 5.16), used for detecting anomalies.

The methods proposed in the previous sections are effective for detecting anomalies, but they show important conceptual limitations.

Autoencoders detect anomalies by recognizing what is not normal, but they do not use any additional knowledge to enrich the concept of normality. Imagine that we are monitoring a machine that works in different conditions or where the process has different

phases: an Autoencoder will just understand what is the normal behavior as a whole, but it does not learn such peculiarities.

On the other hand, the model described in the previous section leverages this kind of information but it does not have a technique for modeling degradation, it just detects anomalies by looking when the classifier decreases its confidence in predicting the right class. However, as we will see in Chapter 6, the closed set of classes may lead to unstable predictions when a bearing wears out. Indeed, when a bearing degrades we observe a decrease in the probability of the correct class, because of data drift; correspondingly, we see an increase in the probability of some other class, because probabilities always sum up to one. However, is hard to believe that the other probabilities can increase arbitrarily because the bearing does not change its operating condition. As a result, the model becomes somehow undecided in which class should predict. This behavior is born from the fact that we are considering a closed-set classifier that learns to discriminate across operating conditions from normal data but it cannot understand whenever an instance is unknown.

In this section, we propose a novel approach for Anomaly Detection that overcomes the weakness of the previous models by considering jointly the coding-decoding reconstruction and a classification tasks for building a richer concept of normality and complementary detect anomalies. Figure 5.5 reports an outline of the model.

In the development of the model, we have taken inspiration from [49], where the authors propose a Classification-Reconstruction method for Open Set Recognition, in the context of Computer Vision. Specifically, we have brought the idea of exploiting the reconstruction and the classification tasks synergically as well as the use of class-confidence weights; but we have changed both the neural network architecture and the expression of the confidence weights.

For building the model, we start by extending an Autoencoder by adding a few layers for classification as bifurcation of the architecture from the bottleneck layer, making the model able of performing both reconstruction and classification (see Figure 5.6). The classification power is then enhanced by considering also an independent classifier, namely a *XGBoost* classifier, and averaging their predictions. Then, we use the two tasks simultaneously for building a strong open set classifier that is able to adjust the predicted probability distribution to both recognizing whenever an instance is unknown and discriminating across operating conditions. Finally, we exploit the open set classifier for detecting anomalies.

The development of such a model requires two sequential phases:

- The training of the *Autoencoder-Classifier* and *XGBoost* models;
- The adjustment of the probabilistic model to the Open Set framework;

As a result, we expect that this new model is able of obtaining more stable predictions even when the bearings wear out: specifically we expect that when the model encounter the data drift the decrease in the correct class probability causes an increase in the *unknown class probability*.

In the following, we go into details in each of the two phases for model development.

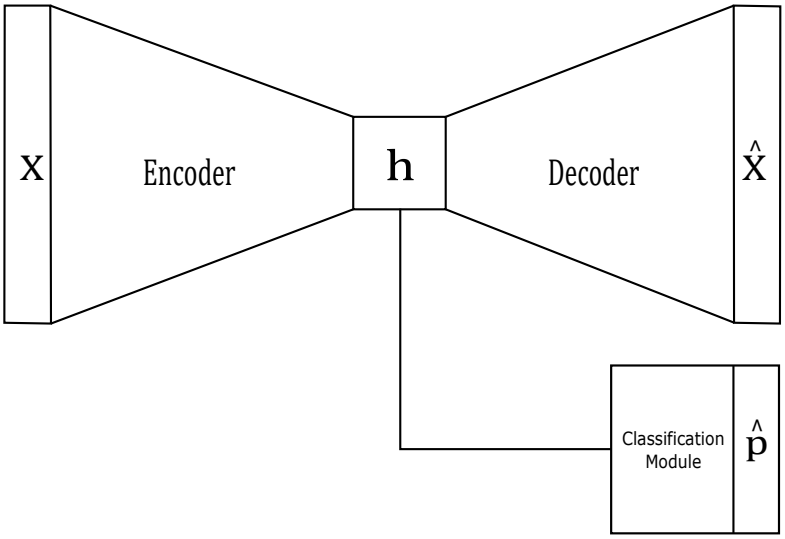

Figure 5.6. An Autoencoder with a classification module.

Phase I - Training

In the first phase we use the training set for learning two sub-models: an Autoencoder with a classification module and an XGBoost classifier. We start by explaining the modifications we introduce in the Autoencoder, described in Section 5.2.1, for tackling the two tasks, and then we explain how it is combined with the other model.

The bottleneck of an Autoencoder contains the encoded features that brings the most of the information that data contains, therefore, they seems good inputs for a classifier. So, we insert a simple classification model, made up by two layers, after the bottleneck of the Autoencoder: the first is a dense layer with sigmoid activation, while the second is a dense layer with softmax activation, from which we obtain probabilities.

We recall that the softmax function is defined as

$$[\text{softmax}(z)]_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}},$$

where $z \in \mathbb{R}^n$ is an input vector.

As a result, the model has two outputs: the reconstruction error and the classification probabilities, as shown in Figure 5.6.

Formally, for any input x the Autoencoder-classifier perform the following operation

Autoencoder

$$
\begin{aligned}
h_0 &= x \\
h_1 &= \sigma(W_1 h_0 + b_1) \cdot z_1 \\
h_2 &= W_2 h_1 + b_2 \qquad (\textit{Bottleneck layer}) \\
h_3 &= \sigma(W_3 h_2 + b_3) \cdot z_2 \\
\hat{x} &= W_4 h_3 + b_4
\end{aligned}
$$

Classifier

$$
\begin{aligned}
h_4 &= \sigma(W_5 h_2 + b_5) \cdot z_3 \\
\hat{p}^{(AE)} &= \text{softmax}(W_6 h_4 + b_6)
\end{aligned}
$$

where $\hat{x}$ is the reconstruction and $\hat{p}^{(AE)}$ is the predicted probability vector, and z_1, z_2, and z_3 are dropout layers with probability 0.01; while W_i, b_i with $i = 5{,}6$ are the new parameters that account for the classification module. We have introduced a superscript to the output probabilities because we will consider also the XGBoost classifier. The parameters of the model are W_i and b_i, for $i \in \{1,2,\ldots,6\}$; their dimensions meet the input-output requirement of each layer, that is, if d_i is the output dimension of layer i, we have that

$$W_i \in \mathbb{R}^{d_i \times d_{i-1}}, \quad b_i \in \mathbb{R}^{d_i}, \quad i \in \{1,2,3,4,6\},$$

and

$$W_5 \in \mathbb{R}^{d_5 \times d_2}, \quad b_i \in \mathbb{R}^{d_5}.$$

As in Section 5.2.1, the layers's dimensions are chosen by trial and errors. Specifically, we set:

- $d_1 = d_3 = 40$,
- $d_2 = 10$,
- $d_5 = 5$,
- $d_6 = |\mathcal{C}|$;

where $|\mathcal{C}|$ is the number of classes, which is equal to 3 in our experimental setting.

We learn the Autoencoder-Classifier parameters with a two-step procedure. First, the Autoencoder alone is trained for 750 epochs by minimizing the Mean Squared Error loss. The obtained Autoencoder weights will initialize the corresponding parameters for the next phase. Then, we train the overall architecture by minimizing, jointly, the Mean Squared Error (MSE) from the decoder's output, and the Cross Entropy (CE) from the classifier's output. Specifically, we consider the overall objective function

$$L(\mathcal{D}; \Theta) = \mathcal{L}^{(MSE)}(\mathcal{D}; \Theta) + \mathcal{L}^{(CE)}(\mathcal{D}; \Theta), \tag{5.14}$$

where $\mathcal{L}^{(MSE)}(\mathcal{D};\Theta)$ is the MSE loss (see formula 5.6), and $\mathcal{L}^{(MSE)}(\mathcal{D};\Theta)$ is the CE loss (see formula 5.5), and we minimize it for 750 epochs. As in Section 5.2.1, for the optimization, we consider the Adam optimizer with learning rate $\alpha = 0.001$, and $b = 64$ samples as batch size.

Let $f(\cdot)$ be an XGBoost classifier with the same hyper-parameters described in the previous section, that for any input x gives as output the probability vector

$$\hat{p}^{(XGB)} = f(x) \in \mathbb{R}^{|\mathcal{C}|}.$$

Then, we train the XGBoost on the training set.

As a result, we have two classifiers able to discriminate the operating conditions in which a bearing run, in terms of predicted probabilities. Therefore, it is natural to consider an ensemble classifier that merges predictions from the Autoencoder-Classifier and the XGBoost. Specifically, we obtain a single probability vector by averaging their predictions,

$$\hat{p} = \frac{\hat{p}^{(NN)} + \hat{p}^{(XGB)}}{2}. \tag{5.15}$$

Phase II - Probability calibration

The second phase merges the two outputs (and tasks) into a single object, by developing an open set classifier.

The probabilities from the ensemble classifier are balanced for taking into account the possibility of encounter unknown concepts. Specifically, each probability $\hat{p}_j$ is weighted by a coefficient w_j that quantify the degree of confidence the model have in making such prediction. We refer to the quantities $\{w_j\}_{j=1}^{|\mathcal{C}|}$ as **class-confidence weights**. The idea is the following. When a bearing is healthy the model should accurately predict the correct operating condition. On the other hand, when the bearing degrades, the classifier not only decreases in performance but it become also uncertain with its prediction because it encounters instances that are more and more different from the ones seen during training. By construction, the Autoencoder will recognize this kind of drift, so it will play a crucial role in defining suitable coefficients w_j.

Specifically, each weight w_j is a number between 0 and 1

$$w_j \in [0,1], \quad \forall j \in \mathcal{C},$$

and it is smaller the higher the reconstruction error is. Moreover, if some coefficient is strictly smaller than 1, the adjusted probabilities do not sum up to one anymore: the residual account for the **unknown class**.

Formally, the new probabilistic model is given by

$$\hat{p}^{(OPEN)} = \begin{cases} \hat{p}_j w_j & \text{if } j \in \mathcal{C}; \\ \sum_{j \in \mathcal{C}} \hat{p}_j (1 - w_j) & \text{otherwise;} \end{cases} \tag{5.16}$$

where the last row is the **unknown class probability**,

$$\mathbb{P}(x \in \{unk\}) = \sum_{j \in \mathcal{C}} \hat{p}_j (1 - w_j) = 1 - \sum_{j \in \mathcal{C}} \hat{p}_j w_j = 1 - \sum_{j \in \mathcal{C}} \hat{p}_j{}^{(OPEN)},$$

and can be seen as the probability of being in none of the known classes.

We note that the weights as well as the predict probabilities depend on data instances, so implicitly suppose that $w_i = w_j(\cdot)$ and $\hat{p_j} = \hat{p_j}(\cdot)$ are functions rather than simple numbers.

Next, we describe how we compute the weights, w_j. In Section 5.2.1 we consider the reconstruction errors to determinate thresholds and then we predict an anomaly whenever the threshold is exceeded. Here instead we want quantify by how far the threshold is surpassed. Specifically, for any class $c \in \mathcal{C}$ we consider all of the instances of that class in the validation set and for each we compute the reconstruction error

$$E(x) = ||\hat{x} - x||_2^2 = \sum_{i=1}^{d} (\hat{x}_i - x_i)^2 \tag{5.17}$$

and we collect these quantities in the vector $\mathcal{E}_c$; then we compute the threshold for class c as the 99% quantile of reconstruction error empirical distribution,

$$\mathcal{T}_c = \text{quantile}(\mathcal{E}_c, 0.99), \quad \forall c \in \mathcal{C}. \tag{5.18}$$

Finally, for every class $c \in \mathcal{C}$ we compute the weight as

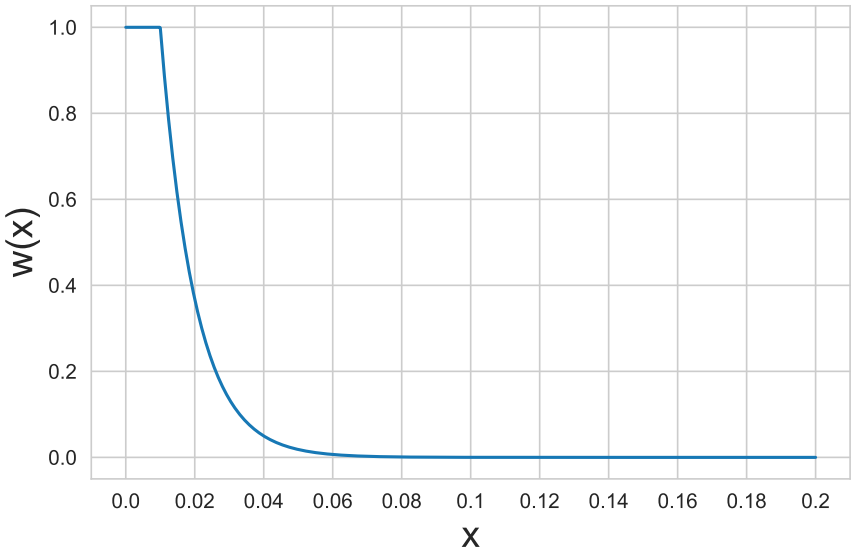

Figure 5.7. An example of the confidence function (see formula 5.19) with a threshold of 0.01.

$$w_c = \begin{cases} e^{-\frac{E(x)-\mathcal{T}_c}{\mathcal{T}_c}} & \text{if } E(x) > \mathcal{T}_c \\ 1 & \text{if } E(x) \leq \mathcal{T}_c \end{cases} \tag{5.19}$$

Note that if $E(x) \leq \mathcal{T}_c$ the probability is not altered,

$$w_c = 1 \implies \hat{p}_c^{(OPEN)} = \hat{p}_c w_c = \hat{p}_c, \quad c \in \mathcal{C}.$$

Instead, if $E(x) > \mathcal{T}_c$ the probability is shrunk,

$$w_c = e^{-\frac{e(x)-\mathcal{T}_c}{\mathcal{T}_c}} < 1 \implies \hat{p}_c^{(OPEN)} = \hat{p}_c w_c < p_c,$$

and the unknown class probability is positive. Indeed,

$$\hat{p}_c^{(OPEN)} < p_c \implies \sum_{c \in \mathcal{C}} \hat{p}_c w_c < 1,$$

then,

$$\mathbb{P}(x \in \{unk\}) = \sum_{c \in \mathcal{C}} \hat{p}_c (1 - w_c) = 1 - \sum_{c \in \mathcal{C}} \hat{p}_c w_c > 0.$$

Interestingly, we can interpret formula 5.19 as follows. For sake of simplicity we drop the class-specific subscript. The quantity

$$\frac{E(x) - \mathcal{T}}{\mathcal{T}}$$

is the percentage of excess, in terms of reconstruction error, from the threshold, T. Thus, the formulas 5.19 defines weights that:

- if the threshold is not exceed, the keep the probability untouched,
- else, the shrink according to the percentage of excess.

For example, if $E(x) = T$, then,

$$w = 1;$$

if $E(x) = 2 \cdot T$, then,

$$w = \frac{1}{e};$$

or, more in general, if $E(x) = k \cdot T$,

$$w = \left(\frac{1}{e}\right)^{k-1}.$$

This formulation is a variant of the one proposed by [49], for developing an Open Set classifier. In [49], the authors consider a Weibull distribution for modeling the class confidence weights. However, such a formulation is inappropriate for anomaly detection because it could also penalize normal instances. Instead, the use of a threshold and a piece-wise function allow to not penalize samples that, according to the Autoencoder, should not be anomalous, and penalize, proportionally, the instances with a reconstruction error that exceeds the threshold.

We denote the model that we have obtained as $\mathcal{CR}(\cdot)$, that for any sample $x \in \mathbb{R}^d$ it gives as output the $|\mathcal{C}| + 1$ classification probabilities

$$\hat{p}^{(OPEN)} = \mathcal{CR}(x) \in \mathbb{R}^{|\mathcal{C}|} \times \mathbb{R}, \tag{5.20}$$

that account both form known and unknown classes.

Anomaly Detection algorithm

Differently from Section 5.2.2, the design of the model explicitly takes into account the possibility of discovering unknown classes, and, as a result, we do not need to define threshold anymore for understanding when we meet anomalies. Indeed, for any new sample (x_i, c_i), we can use the open set classifier to predict the class with maximum probability, if this coincides with the actual one c_i, then x_i is normal, otherwise it is an anomaly.

Formally, for any data point $(x_i, c_i) \in \mathbb{R}^d \times \mathcal{C}$,

1. we compute predicted probability probability $\hat{p}^{(OPEN)} = \mathcal{CR}(x)$,
2. we get the predicted class with maximum probability $\hat{c}_i = \underset{j \in \mathcal{C} \cup \{unk\}}{argmax} (\hat{p}_j^{(OPEN)})$,
3. if $\hat{c}_i = c_i$, we predict as **normal**,
4. otherwise we predict an **anomaly**.

Interestingly, we can observe that two factors drive the prediction. First, the reconstruction error enters the probabilistic model, with formula 5.16, by scaling the probabilities from the classification model. When the class confidence weights assume values that approach zero, the probabilistic model shifts the probability distribution towards the unknown class, resulting in an anomaly prediction. In this situation, the reconstruction task completely drives the prediction. Instead, if class confidence weights assume intermediate values, the decision depends on the mutual relations between classification probabilities.

Chapter 6

Results

In this chapter, we investigate the Anomaly Detection models, proposed in Section 5.2, on the two benchmark datasets described in Chapter 2. Specifically, we conduct an analysis in two directions:

1. a **qualitative evaluation**, described in Section 6.1, where the models are tested on the *PRONOSTIA* dataset with the aim of understanding how they can be used for monitoring an industrial process.

2. a **quantitative evaluation**, described in Section 6.2, where, by considering the *CWRU* dataset, we compare the proposed models in terms of an anomaly detection score.

6.1 Qualitative analysis

The PRONOSTIA dataset (see Section 2.1) is a collection of 17 run-to-failure experiments, where rolling bearings are monitored under different operating conditions. At the beginning of each experiment the installed bearing is intact. Then, as the time passes the bearings suffer from the natural degradation, and the experiments proceed until they end up in failures. Therefore, since the process is continuously monitored by the sensors, we have the complete picture that describe the degradation process that lead the bearing to the failure. However, we do not have labels that state when the bearings start to show signs of defects. For that reason, we can only conduct a qualitative analysis on this dataset.

The data is processed according to the operation described in Section 4.2, resulting in a dataset with 24018 rows and 277 columns (the format described by equation 4.2). Then, in order to test the Anomaly Detection models, we split the dataset into distinct portions for training and testing.

In Section 5.2, we have pointed out that the proposed *Anomaly Detection* requires only normal instances for training. Generally speaking, we can think of normal data when we are monitoring rolling bearings, or more in general machines, under ordinary conditions,

while anomalous data refers to situations like unexpected events, or unexplainable behaviors. However, it is not trivial to discern which data are normal or anomalous, indeed, it is precisely the problem considered in developing anomaly detection models. In a real industrial scenario, we should require the contribution of a domain expert. However, in this case, our data derive from run-to-failure experiments and we can make an assumption. Specifically, since most of the experiments' duration ranges from 1 hour to 8, and at the beginning the rolling bearings are brand new, we can assume that for the first 30 minutes we are monitoring bearings with no defects, or at least with minimal ones.

Therefore, we split the dataset into two portions:

- the *learning set*, made up of the first 30 minutes of monitoring from all experiments, that will be used to train Anomaly Detection models;
- the *test set*, that contains the remaining part from all experiments, and will be used for testing Anomaly Detection models.

In addition, we further divide the *learning set* into two portions, namely the *training set* and the *validation set*, with a 80/20 stratified random splitting, by considering the *operating condition* as class label. That is, we divide the learning set into two parts that account respectively for 80% and 20% of the original set, moreover, the records in the two parts are randomly sampled by conserving the same operating condition class proportion of the original set for both splits.

Finally, we normalize the numerical data with a [0,1] **min-max scaling**. Specifically, let we consider a matrix $X \in \mathbb{R}^{m \times d}$ that contains numerical data, for each column $j \in \{1,2,\dots,d\}$ we compute the minimum,

$$m_j = \min_{i \in \{1,2,\dots,m\}} X_{i,j}, \tag{6.1}$$

and the maximum,

$$M_j = \max_{i \in \{1,2,\dots,m\}} X_{i,j}; \tag{6.2}$$

after that, we normalize the j-th columns as

$$X_{i,j} \leftarrow \frac{X_{i,j} - m_j}{M_j - m_j}, \quad \forall i \in \{1,2,\dots,m\}, \tag{6.3}$$

and we repeat the procedure for all column $j \in \{1,2,\dots,d\}$. We compute the quantities 6.1 and 6.2 by considering on the *training set* while we apply the normalization 6.3 to all sets.

The normalized datasets will be considered in the Sections 6.1.1, 6.1.2, and 6.1.3 for training and evaluating the proposed Anomaly Detection models.

6.1.1 Autoencoder Anomaly Detection

In this section, by considering the *PRONOSTIA* dataset 2.1, we investigate the *Autoencoder* anomaly detection model described in Section 5.2.1.

By considering the training set we learn the model's parameters by minimizing the Mean Squared Error (MSE) loss (see equation 5.6), with a procedure of two steps, as described in Section 5.2.1. The first step is used just to initialize the parameters for the second phase, by optimizing the weights only for a restricted portion of the model. The second phase considers the whole architecture, and the optimizer performs 1500 epochs. Figure 6.1 reports the training history of the second phase in terms of loss computed on training and validation set. The training portion only is used by the optimizer for learning the model's parameters, while the validation set is just considered for obtaining a benchmark loss score at each epoch. From the Figure 6.1 we can see that the training and validation losses are close to each other, and, therefore, we can conclude that the model does not overfit.

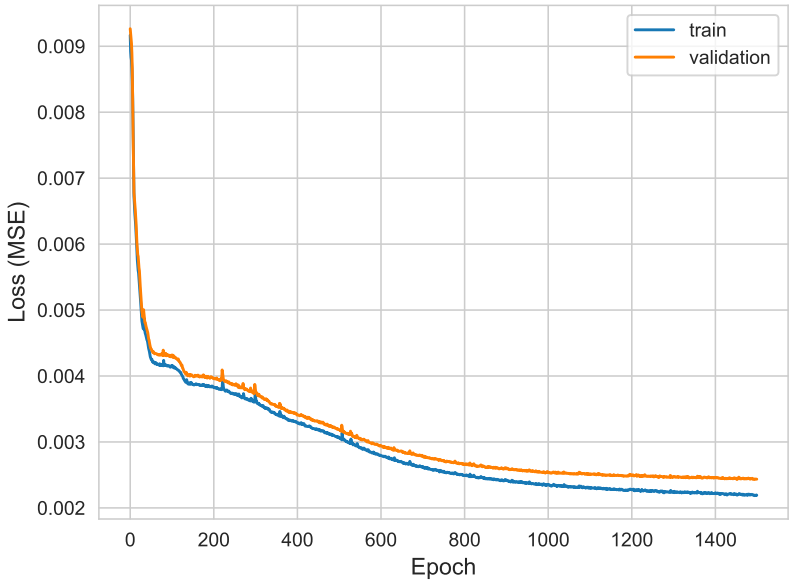

Figure 6.1. Autoencoder's training loss history.

Next, we consider the validation set and we compute the reconstruction error threshold as the 99% quantile of the reconstruction errors. We obtain a value of

$$\mathcal{T} = 0.0089.$$

Figure 6.2 shows that considering validation rather then training data leads to determinate a more robust threshold. Indeed, since during training we minimize the MSE loss, compute the threshold from training data may easily lead to underestimate the new samples's error. Specifically, with training data we get a value of 0.0072, that is around 20% less than the value obtained with the validation portion. This underestimate could lead the model to be too sensible in evaluating the reconstruction errors by predicting

anomalies even when there are none. Moreover, since validation data are unseen by the model during training, they are representative of the new instances that will come, and the obtained threshold leads to discriminate anomalies more carefully.

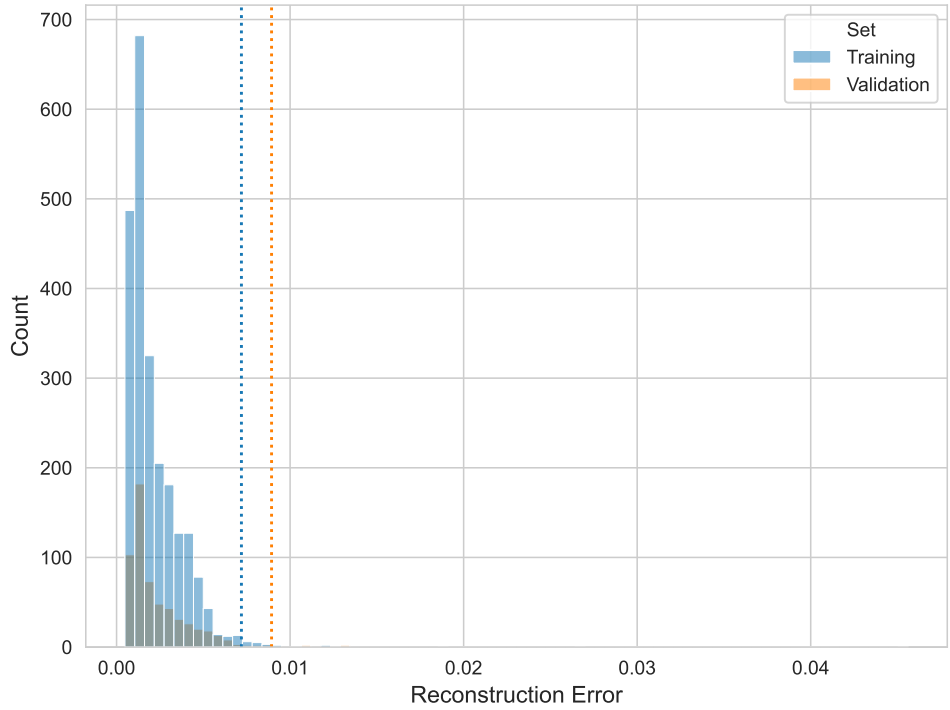

Figure 6.2. The histogram of the reconstruction error distribution over the training and validation set. The vertical dotted lines are the corresponding thresholds.

As a result, the obtained model predicts an anomaly whenever the reconstruction error is above the value of 0.0089. Then, we can consider the test portion and investigate how the model behave in monitoring new signals. For example, Figure 6.3 reports the anomaly predictions over time for the signals from experiments *Bearing*1_1, *Bearing*2_2, *Bearing*3_2 (see Table 2.1), the same are reported in Figure 3.3 of Chapter 3. In the figure we have also included the first 30 minutes of monitoring of the experiments, that forms the learning set. From Figure 6.3 we can see that the experiments depicts very different degradation patterns both in terms of hours of execution and reconstruction errors:

(a) Starting from hour 1 the algorithm detects anomalies; the reconstruction error remains below the value of 0.2 (which is around 22 times more the threshold) until hour 4; then it increases more and more until the end of the experiment.

(b) Almost at the beginning of the test period the model starts to detect anomalies; the reconstruction error sharply increases until it reach a maximum between hour 1 and 1.5; then it decreases until hour 2, by always showing values above the threshold; finally it suddenly increase to extremely high values until the end of the experiment.

(c) The algorithm detect an isolated anomaly before one hour; then it identify a normal behaviour until hour 4, where it detect a sequence of anomalies; surprisingly, the anomalies disappears until around four hour and an half, after that, we can see a sudden increase to the reconstruction error.

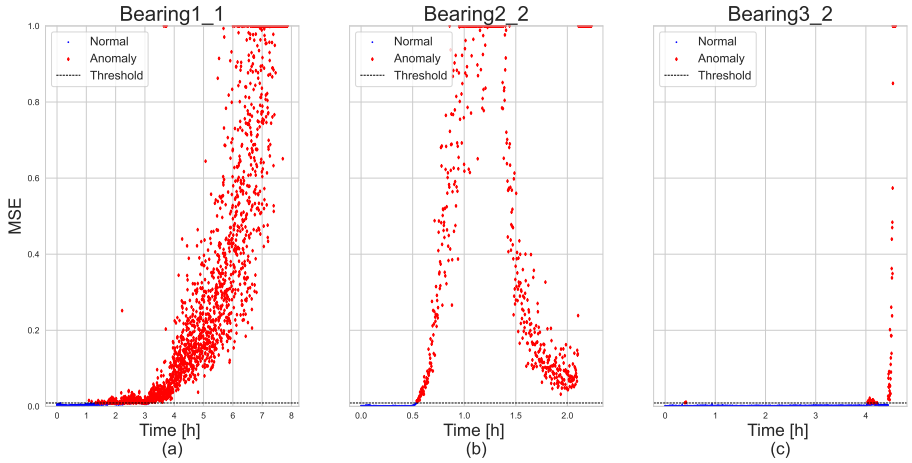

Figure 6.3. Monitoring 3 different experiments with the *Autoencoder*. The reconstruction error are cut-off above 1 for visualization purposes.

The analysis confirms the insights that we bring from the analysis of the raw signals of Figure 3.3: the experiments 1_1 and 2_2 show early sign of defect with large advance, while in experiment 3_2 the bearing is healthy for most of the time and it shows a fast degradation pattern. Therefore, from a qualitative point of view the algorithm is able of correctly detecting anomalies on new data.

On the other hand, is difficult to interpret the reconstruction error values when they are above the threshold. Specifically, the model detects as anomalies the instances with a reconstruction error above the threshold, but we can ask: the anomalies are all equal to each other? An anomaly could be more severe than another? Does a greater error correspond to a more severe anomaly?

If the threshold has a value of 0.0089 we can say that if the threshold is slightly exceed, for example when the reconstruction error is around 0.01, the anomaly is mild: it moderately deviates from normality. But outside of this range, it is difficult to draw conclusions: discern between the severity of a value of 0.02 (around 2 times the threshold) or 0.05 (around 5 times the threshold) or even 0.2 (around 22 times the threshold) is impossible. This problem arise from the fact that the reconstruction error is a quantity with no

prior upper bounds. Indeed, the values reported in the pictures of Figure 6.3 are cut off at 1 because the reconstruction error extremely increase at the end of the experiments: without limiting the plot we could not observe the behaviour of the reconstruction error in the proximity to the threshold.

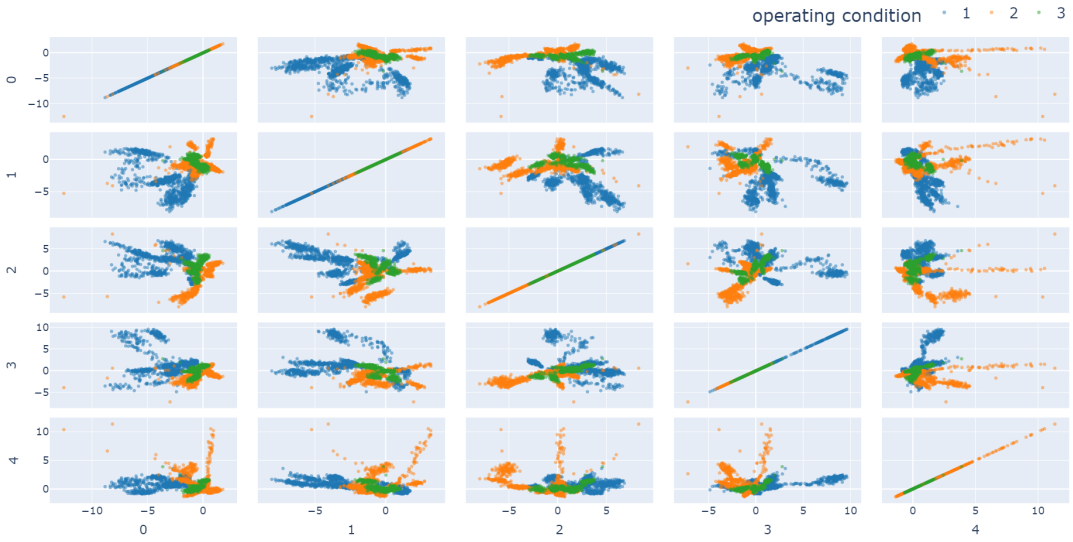

Figure 6.4. Latent representation of data from *learning set*.

Finally, is interesting to investigate if the model somehow understand the presence of the distinct operating conditions. To this extent, we explore the latent representation of the data from learning set. We recall that the latent representation is the output of the encoder, and it summarizes the data with a 5 dimensional representation that conserve the most relevant information.

Figure 6.4 reports the latent representations of the data from learning set: we can see that the data from the distinct operating conditions are organized into clusters with stretched profiles that often overlap each other. Moreover, the figure is quite similar for the one presented with PCA, in Section 3.2.1.

As we can expect, since signals from different operating condition shows distinct characteristics (as we have observed in Section 3.2.1) the points of distinct classes are arranged into different clusters. However, it is not clear the nature of the stretched profiles and of the overlapping of the clusters.

6.1.2 Operating Condition Classification Anomaly Detection

In this section, we consider the *PRONOSTIA* dataset, and we investigate the Anomaly Detection model proposed in Section 5.2.1.

The model exploits an auxiliary classification task for detecting anomalies. Specifically, it exploits the fact that the bearings are monitored under different operating conditions, that implicitly generates signals with different characteristics, as we have observed in

Section 3.2.1. Moreover, it leverages the presence of data drift caused by the natural degradation for detecting the anomalies, by spotting the decrease in the classifier performances.

As a first step, we consider the problem of discriminating among operating conditions, and we select the *XGBoost* classifier as classification model. Then, we determinate its hyper-parameters with a 10 fold cross validation: we split the learning set in 10 folds, and at each iteration we select nine of these folds to train the model and the remaining one for the evaluation; we repeat the procedure until all folds are tested. Moreover, the folds are made by preserving the percentage of samples for each class, and we select the accuracy as evaluation score.

We recall that the accuracy is defined as the ratio between the number of correct prediction and the total number of samples,

$$Accuracy = \frac{Number\ of\ correct\ prediction}{Number\ of\ samples}.$$

As a result, for each hyper-parameter configuration we obtain 10 accuracy scores, and we select the one that maximize the mean accuracy. Table 5.1 reports all hyper-parameters configurations together with the mean and the standard deviation of corresponding scores.

From the Table 5.1 we can see that best configuration in terms is given by:

- $Maximum\ Depth\ of\ a\ tree = 3$,
- $\lambda = 5$,
- $Subsample\ ratio = 0.5$.

With these hyper-parameters, every tree can have at most 3 levels and is trained on a randomly sampled data portion that accounts for the half of the dataset. Moreover, we observe that the selected parameters are the ones that most limit the overfitting. We can expect, therefore, that the model will show good generalization skills.

Next, we train the *XGboost* classifier with the selected set of hyper-parameters on the training set only. Figure 6.5 reports the classification performances on training, validation, and test sets. We can see that the model gains perfect results in the training set, and similarly, in the validation, it makes only two misclassifications, by gaining 99.66% of accuracy. Contrarily, in the test set, we observe a relevant decrease in performance. This behavior confirms the assumptions we have considered for developing the model in Section 5.2.2. As time runs, the model suffers from data drift, because the natural degradation of the bearings causes a change in the probability distribution that generates data, and, consequently, the classifier finds difficulties in predicting the correct class.

We exploit this behavior for detecting anomalies: we set thresholds that declare the required probability for predicting an operating condition; if the probability falls below the level, we raise an anomaly warning.

Specifically, as described in Section 5.2.2, we consider the validation set and we compute the class-specific thresholds according to formula 5.13, and we obtain:

1. Threshold for class 1 : 0.991,

Ranking	***Maximum Depth of a tree***	λ	***Subsample ratio***	***Accuracy Mean***	***Accuracy Standard Deviation***
1	**3**	**5.0**	**0.5**	**0.9641**	**0.0986**
2	9	5.0	0.5	0.9607	0.1067
3	6	5.0	0.5	0.9603	0.1066
4	3	2.5	0.5	0.9572	0.1181
5	9	2.5	0.5	0.9572	0.1137
6	6	2.5	0.5	0.9569	0.1135
7	3	0.0	0.5	0.9479	0.1437
8	3	5.0	1.0	0.9469	0.1491
9	9	5.0	0.75	0.9459	0.1499
10	3	2.5	0.75	0.9455	0.1533
10	3	5.0	0.75	0.9455	0.1533
12	6	5.0	0.75	0.9452	0.1509
13	3	2.5	1.0	0.9452	0.1509
14	9	5.0	1.0	0.9445	0.1518
15	6	5.0	1.0	0.9438	0.1515
16	9	2.5	1.0	0.9434	0.1537
17	9	2.5	0.75	0.9428	0.1604
18	6	0.0	0.5	0.9427	0.1581
19	6	2.5	1.0	0.9427	0.1536
19	3	0.0	1.0	0.9427	0.1558
21	6	2.5	0.75	0.9421	0.1602
22	9	0.0	0.5	0.9421	0.1602
23	3	0.0	0.75	0.941	0.1644
24	6	0.0	0.75	0.939	0.165
25	9	0.0	0.75	0.9386	0.1682
26	6	0.0	1.0	0.9386	0.1625
27	9	0.0	1.0	0.9352	0.1659

Table 6.1. All the considered hyper-parameter configurations of XGboost.

2. Threshold for class 2 : 0.938,

3. Threshold for class 3 : 0.976.

We observe that the thresholds are rather demanding in terms of classification performance. Indeed, for example, to predict class 1 we ask for a classification probability of at least 0.991, thus, the model should be almost sure in the prediction.

Moreover, by looking at Figure 6.6 we can see that the probabilities show different

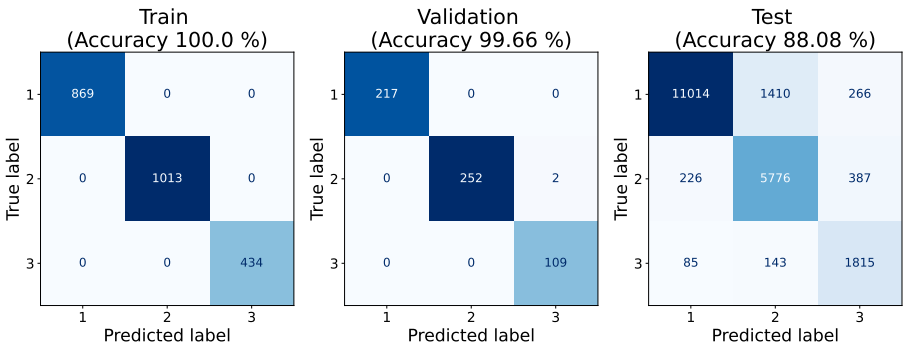

Figure 6.5. Confusion matrices by considering, training, validation, and test sets.

patterns for the distinct classes. The distribution of classification probability for class 1 is narrow around 1.0, while the ones of classes 2 and 3 are more skewed toward 0.9. Consequently, the thresholds assume different values.

Interestingly, we can observe that *XGBoost* model seems quite a good classifier for discriminating operating conditions. Indeed, in the validation set, where there are unseen samples that should not be still exposed to data drift, the classifier makes only a minimal number of mistakes, and predictions are made with corresponding very high probabilities.

Then we can investigate the Anomaly Detection model, made up of the *XGBoost* together with the probability thresholds, in monitoring the vibration signals. Since the model is structured analogously to the *Autoencoder* explored in the previous section, we expect similar results. Indeed, in both models, we calculate thresholds starting from indicators computed on validation data which define from which values we should consider instances as anomalies. The only difference is that this model uses probability as indicators, while the *Autoencoder* uses reconstruction errors.

In Figure 6.7 we can see the classification probability for the correct operating condition, monitored over time in 3 different experiments. In each plot the threshold is reported as a black dotted line, and the anomaly predictions are shown by changing color and shape.

We can see common points with the results of the previous section:

- in experiment *Bearing*1_1 (see 6.7*(a)*), the model detects anomalies after one hour;
- in experiment *Bearing*3_2 (see 6.7*(c)*), before 1 hour the model detects isolated anomalies, while after 4 hour it detect many anomalous instances.

On the other hand, in experiment *Bearing*3_2 (see 6.7*(c)*), the model detect also anomalies between one hour and two, which are not recognized by the Autoencoder. Moreover, in *Bearing*2_1 (see 6.7*(b)*) the model detects anomalies only it the last minutes of the experiment, while the Autoencoder detect the degradation process at early stages.

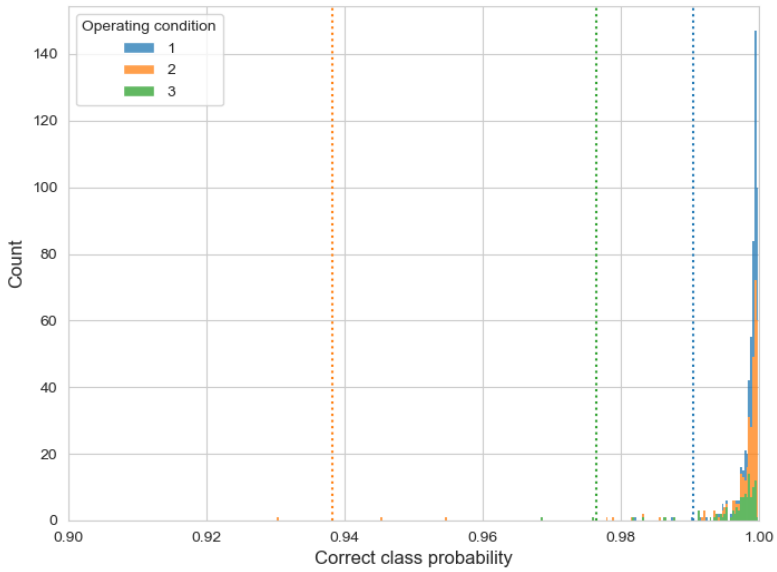

Figure 6.6. Histograms of correct class probabilities on the validation set and the corresponding rejection thresholds as vertical dotted lines.

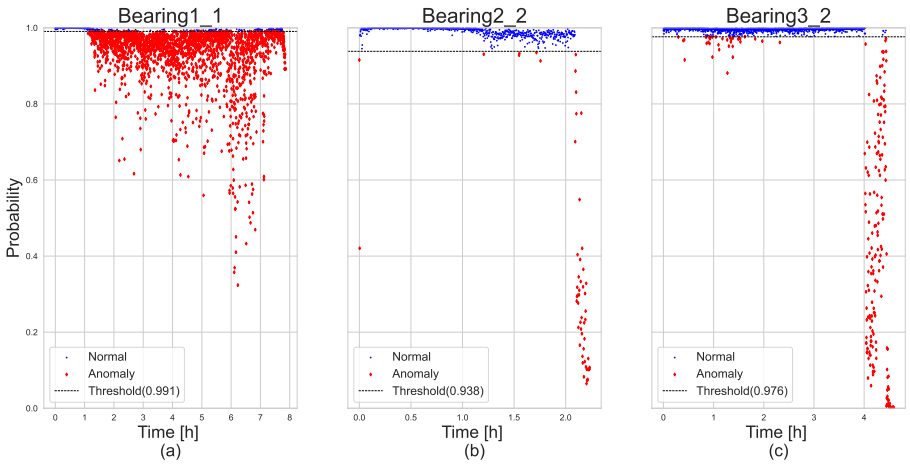

Figure 6.7. Monitoring 3 different experiments with the model describe in Section 5.2.2. The same experiments are described in Figure 3.3 of Chapter 3.

In addition, by looking with more attention at Figure 6.7*(a)*, we can make further considerations:

- it is not true that the more the degradation process proceeds, or equivalently, the more the defect become evident, the more the probability decreases;
- the predicted probabilities are unstable: after hour 1 we can find also *Normal* prediction (blue points), even toward the end of the experiment.

Therefore, the model has many limitations and the probabilities are hard to be interpreted. As we have anticipated in Section 5.2.3, the behavior is caused by the so called *closed set assumption*: the classifier believe that they exist only the class it already known, and it does not contemplate the possibility of discover unknown classes.

6.1.3 Classification-Reconstruction Anomaly Detection

In this Section, we consider the **Classification-Reconstruction** model, described in Section 5.2.3, and we investigate its effectiveness in monitoring signals and detecting anomalies on *PRONOSTIA* dataset. The model is made up of two main blocks, as shown in Figure 5.5:

- an Autoencoder with a classification module,
- an XGBoost classifier;

the two are trained separately over the training set, and then they are merged for obtaining an Open-Set classifier, which outputs can be used for anomaly detection.

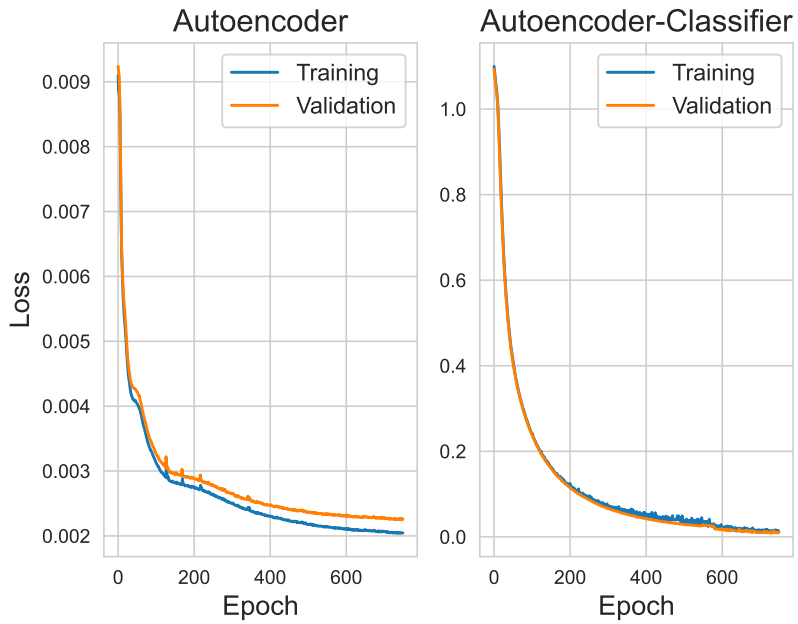

Figure 6.8. Training history for the Autoencoder-Classifier.

First, we train the Autoencoder-Classifier with a procedure of two steps: we train the Autoencoder for 750 epochs by minimizing the Mean Squared Error loss (see formula 5.6), then we add a classification module by connecting it to the encoder's output, and we train the overall architecture by minimizing jointly, the MSE and the Cross-Entropy (see formula 5.14). Figure 6.8 reports the loss history across the epochs for the two steps. On the left, we can see the first step where we train the Autoencoder without the classification module: the loss history is similar to the one of Figure 6.1, presented in Section 6.1.1. On the right is shown the loss curve for the training of the Autoencoder-Classifier. We can see that at the beginning the loss is above 1 and the optimization leads its value towards 0, showing that the model learns to tackle both tasks.

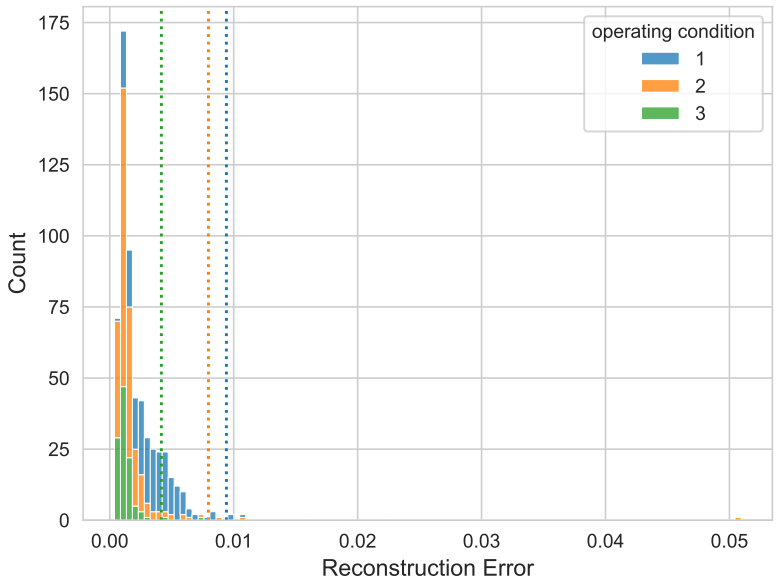

Figure 6.9. The histogram of the reconstruction error distribution in the 3 different operating conditions. The vertical dotted lines are the corresponding thresholds.

Next, we consider the reconstruction output of the Autoencoder-Classifier for computing the thresholds required for the confidence functions (see formulas 5.19). By applying the formulas 5.18, we obtain

1. Threshold for class 1 : 0.0094,
2. Threshold for class 2 : 0.0079,
3. Threshold for class 3 : 0.0041.

Since the thresholds are different the class-confidence functions will shrink differently from the corresponding probabilities.

Then, the XGBoost model is learned on the training set, by selecting the same hyper-parameters configuration determined in Section 6.1.2.

After this, we build the **Classification-Reconstruction Anomaly Detection** model by merging the outputs of the two blocks, together with the class-confidence functions, as described in Section 5.2.3 (see Figure 5.5 for an outline of the model). The model overcomes the issue of defining thresholds, by using the whole output probability vector for predicting anomalies, as described in Section 5.2.3. Moreover, it leads to the definition of a sensible health indicator: since in each experiment we know the operating condition it runs, for assessing the health of the bearing, we can monitor the probability that refers to this specific class.

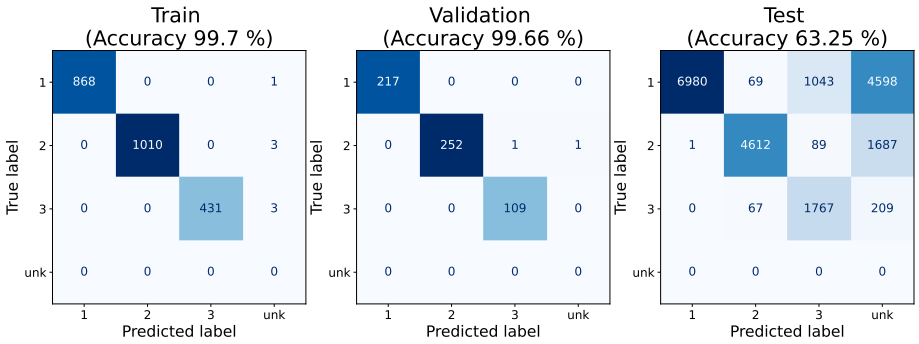

Figure 6.10. Confusion matrices by considering, training, validation, and test sets.

First, we inspect the model's skills in classifying the operating conditions, and investigate the effect of the data drift. Figure 6.10 reports the classification performances on training, validation, and test sets. As in the previous section, we can see that the model gains perfect results in the training set, and similarly, in the validation, it makes only a few misclassifications. Contrarily, in the test set, we observe a relevant decrease in performance, together with an relevant number of predictions related to the *unknown class*. This confirms our hypotheses: the data drift causes a shift in the predicted probability distribution towards the *unknown class*.

Next, we test the model performances in monitoring the run-to-failure experiments. Figure 6.11 reports the health indicator and the anomaly prediction in the monitoring of 3 distinct experiments (the corresponding raw signals are reported in Figure 3.3):

(a) The value of the health indicator is stable at around 1.0 until one hour of monitoring; then its value starts to decrease and the algorithm predicts anomalies; until 3 hours we can see a progressive decrease in the health indicator, and accordingly, an increase in the number of anomaly predictions; after 3 hour most of the predictions reaches the value of 0.0; after 4 the health indicator is stable at 0.0, showing that the bearing is completely worn.

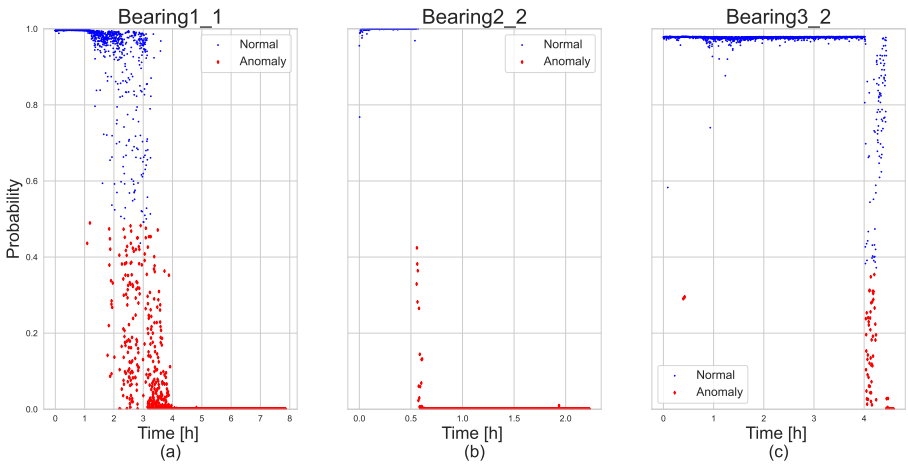

Figure 6.11. Monitoring 3 different experiments with the *Classification Reconstruction* model.

(b) After some minutes from the beginning of the test period the model detects anomalies by showing a drastic change in the condition indicator.

(c) Except for some isolated points, the condition indicator show value of around 1.0 for the first four hours of monitoring; then the value decrease, and the model predicts anomalies; at the end of the experiment, the indicator reaches the minimal value.

From the analysis, we can argue that the health indicator allows interpretations of the anomaly severity. Indeed, since it is a probability, its range is restricted to [0,1], we can make, therefore, the following considerations:

- when it is around 1 we can consider the bearing as completely healthy,
- As the signals deviates from the ordinary the health indicator decreases,
- when it is around 0 the bearings are totally damaged,
- for intermediate values we let the model decide if it is an anomaly by considering the whole probabilistic model.

We end the section by investigating the latent representation of the model. From Figure 6.12, we can see that the distinct operating conditions are organized into well-defined clusters. The stretched profiles and the overlap between clusters that we have seen in Section 6.1.1 are not present anymore: it seems that the model has gained more complete knowledge of the underlying data.

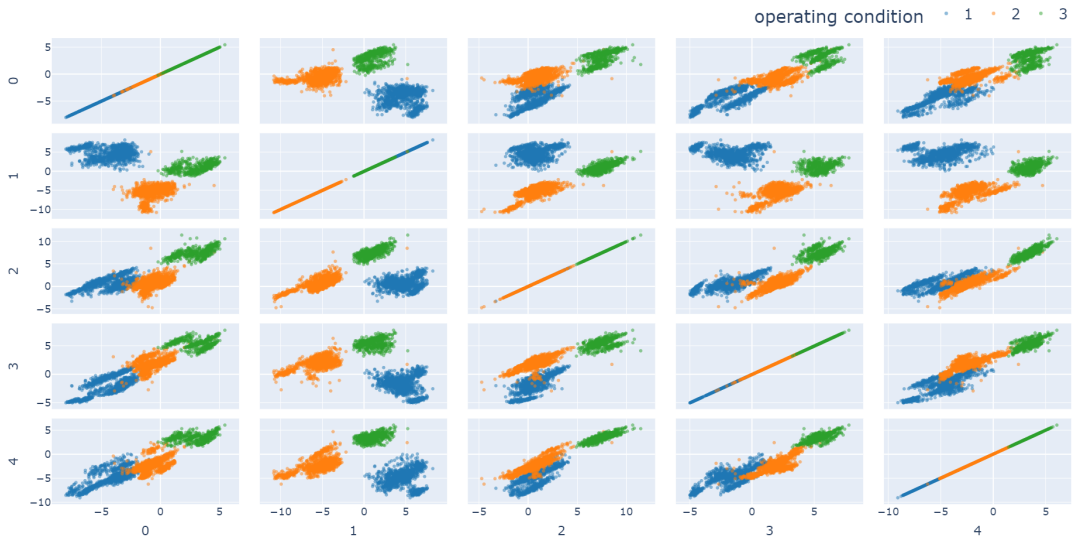

Figure 6.12. The first 5 encoded features of data from *learning set*.

6.2 Quantitative analysis

The *CWRU* dataset (see Section 2.1) contains signals from condition monitoring of brand new as well as defective rolling bearings, monitored by a single vibration sensor. The defects are artificially originated by damaging the bearings in specific locations, resulting in different fault types. We process the data according to the operation described in Section 4.3, by obtaining a dataset with 2589 rows and 72 columns (the format is described by equation 4.3).

Then, in order to evaluate the Anomaly Detection models, we split the dataset into distinct portions for training and testing.

Let us denote the dataset with $\mathcal{D}$. Since in this dataset we have labels that declare which records are normal or anomalous, we can divide the dataset into separate portions:

- $\mathcal{D}^{(normal)}$, the set of normal instances;
- $\mathcal{D}^{(anomaly)}$, the set of anomaly records.

Then, we divide $\mathcal{D}^{(normal)}$, in two parts with an 80/20 stratified random splitting, in which the stratification is made by considering the operating condition class. The largest part will be the learning set. The other part, together with $\mathcal{D}^{(anomaly)}$ forms the *test set*. We use the learning set to train the model and the *test set* for evaluation. Then, the learning set is further divided into two sets with another stratified split, by obtaining the *training set* and the *validation set*.

Finally, we normalize the numerical data with a [0,1] **min-max scaling**, in which we compute the quantities, defined by equations 6.1 and 6.2, by considering the *training set* while we apply the normalization, according to formula 6.3, to all sets.

Therefore, we train the models by considering only normal data, and we test their capacity to recognize anomalies as well as normal instances. Specifically, we select the $F1$ score as a quality measure and we consider the following evaluation protocol:

1. we select a random seed s that controls the first random splitting;
2. we obtain *training*, *validation*, and *test* sets;
3. we train the models on the learning set and we evaluate them, by computing the $F1$ score, on the test set;
4. we iterate the procedure for 30 different random seeds, and we collect the obtained scores.

We recall that, in the context of Anomaly Detection, the $F1$ score is defined as:

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP}, \\ \text{Recall} &= \frac{TP}{TP + FN}, \\ \text{F1-score} &= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \end{aligned}$$

where TP, FP, and FN are defined in Table 6.2.

		Predicted	
		Normal	Anomaly
True	Normal	TN	FP
	Anomaly	FN	TP

Table 6.2. The confusion matrix.

The *Recall* measures the model's capability of recognizing the anomalies. While the *Precision* quantifies the proportion of correctly detected anomalies over the total number of anomaly predictions. These two metrics are complementary, and they are combined into the $F1$-score, which is the harmonic mean of the two.

As a result, we can obtain an overall evaluation of the models that does not depends on the specific data-partition that we have chosen, which can be used for making a reliable comparison.

Table 6.3 reports the results in terms of the mean and standard deviation of the obtained scores. We can see that the mean scores are above 0.99 and the standard deviations are of the order of the third decimal digit or less; therefore all models show excellent and stable performances.

The **Classification-Reconstruction model** outperforms the other methods by showing the highest mean score and the lowest standard deviation. Consequently, we can argue that the proposed model improves anomaly detection performances of an Autoencoder.

Model Name	*Mean*	*Standard Deviation*
Autoencoder	0.99926	0.00044
Operating Condition Classification	0.99046	0.00773
Classification-Reconstruction	**0.99984**	**0.00017**

Table 6.3. Performance of the Anomaly Detection models evaluated on *CWRU* dataset, in terms of $F1$ score.

Chapter 7

Conclusions

In this thesis, we have investigated the connection between **Open Set Recognition** and **Predictive Maintenance** by proposing a novel Anomaly Detection model for rolling bearings fault detection.

First, we have presented two benchmark datasets for condition monitoring of rolling bearings through vibration sensors, and we have proposed a preprocessing pipeline for extracting frequency-domain and time-domain features from raw signals. Moreover, we have explored the data by using Fourier Analysis, observing that the natural degradation process can be seen in terms of data drift: as the defect becomes evident the data distribution changes due to different physical conditions of the system.

Then, we have described Anomaly Detection models that can recognize the data drift by spotting the failure's incipient. First, we have analyzed the use of Autoencoder in Anomaly Detection by monitoring the reconstruction error, in which the method requires setting a threshold that defines the critical value above which we detect anomalies. Next, we observed that Autoencoders do not exploit the fact that the bearings work under three different operating conditions. Then, we have developed an Anomaly Detection model that leverages this information by using a classifier together with thresholds defined in terms of probability for detecting anomalies. However, we have found that such a model leads to unstable predictions because of the closed set of classes. Finally, in Section 5.2.3, we have developed a novel Anomaly Detection model that exploits an Open Set classifier to build a rich concept of normality and detect anomalies. Moreover, the model can recognize anomalies without requiring the definition of sensible thresholds, which is a critical point of the previous models.

We have evaluated the models in terms of qualitative and quantitative analysis, showing that the novel approach outperforms the others in terms of anomaly detection performances, and it leads to the definition of an interpretable condition indicator that can be used for driving maintenance decisions.

Further Works

The **Classification-Reconstruction Anomaly Detection** model that we have proposed requires a set of well-defined classes that characterizes the signals and can be investigated in different industrial scenarios that meet this requirement. For example, we

consider an industrial process in which the sensors monitor the development of a product across different assembly stages; or where a machine tool can work in different operating modes for performing distinct actions.

Moreover, our model can be extended and, possibly, improved by considering different kinds of modifications to its building blocks. In the following, we list some ideas.

First, instead of considering only the XGboost as the additional classifier in equation 5.15, we can extend the model by considering a stack of classifiers and average out all of their predictions.

Second, we can modify the neural network architecture by considering convolutional layers, allowing us to work directly with raw signals.

Then, we can investigate different methods for computing the quantiles in equation 5.18. As an example, we can consider the method proposed in the article [11], which is particularly suitable for skewed distribution.

Moreover, instead of considering the reconstruction error to define the class-confidence weights in equation 5.19, we can consider the distances from the class-specific centroids in the latent space, as reported in [49].

Finally, we can consider different losses for training the Autoencoder. For example, we can learn the reconstruction task by minimizing the Mean Absolute Percentage Error loss [8]. As a result, the reconstruction error is expressed in relative terms rather than absolute ones, leading, thus, to a more clear interpretation. Differently, we can consider the clustering-based losses [12] [28] to learn the classification task, obtaining clusters in the latent space that are well separated and narrow around the centroid. These losses are particularly suitable for defining the class-confidence weights in terms of distances.

Bibliography

[1] Abhijit Bendale and Terrance Boult. Towards open world recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1893–1902, 2015.

[2] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.

[3] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[4] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[6] Francesca Cipollini, Luca Oneto, Andrea Coraddu, Stefano Savio, and Davide Anguita. Unintrusive monitoring of induction motors bearings via deep learning on stator currents. *Procedia Computer Science*, 144:42–51, 2018. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2018.10.503. URL https://www.sciencedirect.com/science/article/pii/S1877050918322129. INNS Conference on Big Data and Deep Learning.

[7] Mingliang Cui, Youqing Wang, Xinshuang Lin, and Maiying Zhong. Fault diagnosis of rolling bearings based on an improved stack autoencoder and support vector machine. *IEEE Sensors Journal*, 21(4):4927–4937, 2021. doi: 10.1109/JSEN.2020.3030910.

[8] Arnaud De Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48, 2016.

[9] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[10] Anton Dries and Ulrich Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):311–327, 2009.

[11] Ted Dunning and Otmar Ertl. Computing extremely accurate quantiles using t-digests. *arXiv preprint arXiv:1902.04023*, 2019.

[12] Dario Fontanel, Fabio Cermelli, Massimiliano Mancini, Samuel Rota Bulo, Elisa Ricci, and Barbara Caputo. Boosting deep open world recognition by clustering. *IEEE Robotics and Automation Letters*, 5(4):5985–5992, 2020.

[13] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[14] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3614–3631, 2020.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[17] Hanqing Hu, Mehmed Kantardzic, and Tegjyot S Sethi. No free lunch theorem for concept drift detection in streaming data classification: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(2):e1327, 2020.

[18] Fabio Immovilli, Marco Cocconcelli, Alberto Bellini, and Riccardo Rubini. Detection of generalized-roughness bearing fault by spectral-kurtosis energy of vibration or current signals. *IEEE Transactions on Industrial Electronics*, 56(11):4710–4717, 2009. doi: 10.1109/TIE.2009.2025288.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[21] Pengfei Liang, Chao Deng, Jun Wu, Zhixin Yang, Jinxuan Zhu, and Zihan Zhang. Single and simultaneous fault diagnosis of gearbox via a semi-supervised and high-accuracy adversarial learning framework. *Knowledge-Based Systems*, 198:105895, 2020.

[22] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[23] Han Liu, Jianzhong Zhou, Yang Zheng, Wei Jiang, and Yuncheng Zhang. Fault diagnosis of rolling bearings with recurrent neural network-based autoencoders. *ISA Transactions*, 77:167–178, 2018. ISSN 0019-0578. doi: https://doi.org/10.1016/j.isatra.2018.04.005. URL https://www.sciencedirect.com/science/article/pii/S0019057818301514.

[24] Douglas A Lyon. The discrete fourier transform, part 4: spectral leakage. *Journal of object technology*, 8(7), 2009.

[25] Atefeh Mahdavi and Marco Carvalho. A survey on open set recognition. In *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 37–44. IEEE, 2021.

[26] Larry Manevitz and Malik Yousef. One-class svms for document classification. *Journal of Machine Learning Research*, 2:139–154, 01 2001. doi: 10.1162/15324430260185574.

[27] Wentao Mao, Wushi Feng, Yamin Liu, Di Zhang, and Xihui Liang. A new deep auto-encoder method with fusing discriminant information for bearing fault diagnosis. *Mechanical Systems and Signal Processing*, 150:107233, 2021.

[28] Dimity Miller, Niko Sunderhauf, Michael Milford, and Feras Dayoub. Class anchor clustering: A loss for distance-based open set recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3570–3578, 2021.

[29] R Keith Mobley. *An introduction to predictive maintenance*. Elsevier, 2002.

[30] Patrick Nectoux, Rafael Gouriveau, Kamal Medjaher, Emmanuel Ramasso, Brigitte Chebel-Morello, Noureddine Zerhouni, and Christophe Varnier. Pronostia: An experimental platform for bearings accelerated degradation tests. In *IEEE International Conference on Prognostics and Health Management, PHM'12.*, pages 1–8. IEEE Catalog Number: CPF12PHM-CDR, 2012.

[31] Kim-Anh Nguyen, Phuc Do, and Antoine Grall. Multi-level predictive maintenance for multi-component systems. *Reliability engineering & system safety*, 144:83–94, 2015.

[32] Numpy. Numpy.org. URL https://numpy.org/.

[33] Hai Qiu, Jay Lee, Jing Lin, and Gang Yu. Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics. *Journal of sound and vibration*, 289(4-5):1066–1090, 2006.

[34] Yongyi Ran, Xin Zhou, Pengfeng Lin, Yonggang Wen, and Ruilong Deng. A survey of predictive maintenance: Systems, purposes and approaches. *arXiv preprint arXiv:1912.07383*, 2019.

[35] Roozbeh Razavi-Far, Ehsan Hallaji, Maryam Farajzadeh-Zanjani, and Mehrdad Saif. A semi-supervised diagnostic framework based on the surface estimation of faulty distributions. *IEEE Transactions on Industrial Informatics*, 15(3):1277–1286, 2018.

[36] Roozbeh Razavi-Far, Ehsan Hallaji, Maryam Farajzadeh-Zanjani, Mehrdad Saif, Shahin Hedayati Kia, Humberto Henao, and Gerard-Andre Capolino. Information

fusion and semi-supervised deep learning scheme for diagnosing gear faults in induction machine systems. *IEEE Transactions on Industrial Electronics*, 66(8):6331–6342, 2018.

[37] B. Samanta, K.R. Al-Balushi, and S.A. Al-Araimi. Artificial neural networks and support vector machines with genetic algorithm for bearing fault detection. *Engineering Applications of Artificial Intelligence*, 16(7):657–665, 2003. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2003.09.006. URL https://www.sciencedirect.com/science/article/pii/S0952197603001143.

[38] Walter J. Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E. Boult. Toward open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7):1757–1772, 2013. doi: 10.1109/TPAMI.2012.256.

[39] Walter J. Scheirer, Lalit P. Jain, and Terrance E. Boult. Probability models for open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2317–2324, 2014. doi: 10.1109/TPAMI.2014.2321392.

[40] SKF. Skf bearing rating life. URL https://www.skf.com/uk/products/rolling-bearings/principles-of-rolling-bearing-selection/bearing-selection-process/bearing-size/size-selection-based-on-rating-life/bearing-rating-life.

[41] Julius Orion Smith. *Mathematics of the discrete Fourier transform (DFT): with audio applications*. Julius Smith, 2008.

[42] Apache Spark. Apache spark. URL https://spark.apache.org/.

[43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[44] V. Sugumaran, V. Muralidharan, and K.I. Ramachandran. Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing. *Mechanical Systems and Signal Processing*, 21(2):930–942, 2007. ISSN 0888-3270. doi: https://doi.org/10.1016/j.ymssp.2006.05.004. URL https://www.sciencedirect.com/science/article/pii/S0888327006001142.

[45] Case Western Reserve University. Case western reserve university bearings dataset. URL https://engineering.case.edu/bearingdatacenter.

[46] Muhammad Habib ur Rehman, Ejaz Ahmed, Ibrar Yaqoob, Ibrahim Abaker Targio Hashem, Muhammad Imran, and Shafiq Ahmad. Big data analytics in industrial iot using a concentric computing model. *IEEE Communications Magazine*, 56(2):37–43, 2018.

[47] Martin Vetterli, Jelena Kovačević, and Vivek K Goyal. *Foundations of signal processing*. Cambridge University Press, 2014.

[48] Heng Wang and Zubin Abraham. Concept drift detection for streaming data. In *2015 international joint conference on neural networks (IJCNN)*, pages 1–9. IEEE, 2015.

[49] Ryota Yoshihashi, Wen Shao, Rei Kawakami, Shaodi You, Makoto Iida, and Takeshi Naemura. Classification-reconstruction learning for open-set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4016–4025, 2019.

[50] Shen Zhang, Fei Ye, Bingnan Wang, and Thomas G Habetler. Semi-supervised bearing fault diagnosis and classification using variational autoencoder-based deep generative models. *IEEE Sensors Journal*, 21(5):6476–6486, 2020.