DOCTORAL THESIS

INFORMATION DENSITY IN HIGHLY DIMENSIONAL TRAINING DATASETS

Luis Jesús Muñoz Molina

UCA
Universidad de Cádiz

Information density in highly dimensional training datasets

by

Luis Jesús Muñoz Molina

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of
Doctor of Philosophy of University of Cádiz

February 2024

Abstract

Esta Tésis Doctoral presenta tres importantes contribuciones en el contexto del análisis de la información contenida en conjuntos de datos que van a ser utilizados para entrenar modelos de Aprendizaje Automático. Estas contribuciones vienen dadas en la forma de nuevas herramientas y métricas para evaluar matemáticamente la mencionada información contenida. Además, en esta Tésis, diversos retos provinientes de diferentes escenarios industriales reales han sido resueltos a través de modelos de Aprendizaje Automático, en cuyo entrenamiento se han usado algunas de las mencionadas herramientas.

Abstract

This PhD Thesis presents three important contributions in the context of the analysis of the information contained in datasets devoted to train Machine Learning models. These contributions are given in the shape of new mathematical tools, techniques and measures devoted to evaluate the aforementioned contained information. Moreover, in the current PhD Thesis, different challenges coming from several real industrial scenarios have been solved by means of the use of Machine Learning algorithms, in whose training some of the aforementioned tools have been used.

Acknowledgements

Con estas palabras me gustaría expresar mi mas profundo agradecimiento a todas aquellas personas que han hecho posible esta Tésis Doctoral. A mis directores de Tésis, cuyo apoyo tanto moral como técnico, consejo y cercanía han sido cruciales. También mis agradecimientos a las personas integrantes del tribunal que ha evaluado este trabajo, quienes generosamente han entregado su tiempo, conocimiento y experiencia en la conclusión de este proyecto.

Me gustaría también agradecer a mi empresa, Capgemini Engineering y a mi centro de trabajo en particular, el AICAM, por brindarme la oportunidad y el espacio que necesitaba para poder no solo aprender, sino poner en práctica muchas de las hipótesis defendidas hoy aquí,

Finalmente, y en especial, mi más profundo agradecimiento a mi familia, en concreto a mis padres, hermana y pareja, cuyo apoyo, amor y confianza incondicional sin duda han hecho ésto posible. Esta Tésis es el fruto del trabajo de todos.

Contents

List of Figures

List of Tables

List of Abbreviations

The following list describes several symbols, abbreviations, and acronyms that will be later used within the body of this thesis.

$\mathbb{R}$	Real Numbers
V	Arbitrary Vector Space
ANN	Artificial Neural Network
CD	Control Dataset
CNN	Convolutional Neural Network
CS	Compressed Sensing
CSS	Categorical Standard Scalation
EOM	Empirical Overlaping Measure
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
LSTM	Long-Short Term Memory
MBM	Montecarlo Based Measure
ML	Machine Learning
MLP	Multi-Layer Perceptron
NAS	Neuromorphic Auditory Sensor
NN	Neural Network
PCA	Principal Component Analysis

PHE Pulsing Histogram Energy
RAL RAndom Assignation of Labels
RNN Recurrent Neural Network
SNN Spiking Neural Network
XAI Xplainable Artificial
XGB XGBoost

Chapter 1

Introduction and Motivation

Probably, one of the most remarkable capabilities of life is adaptability. The adaptability is the ability to change in order to suit or fit to new conditions. Since the beginning of its existence, the human kind took one step further with respect the other known forms of life by artificially adapting to different conditions and solving problems for which they were not biologically adapted. In most cases, life itself was a source of inspiration [44] for the resolution of these problems.

This adaptation capability is based in two main aspects: on the one hand, the ability to observe the surrounding world and identifying patterns in specific indicators that allows to predict how a phenomenon is going to behave in the future or under certain conditions. On the other hand, the ability to imagine, design and develop complex tools for increasing the efficiency in work or reducing the difficulty of specific tasks. With time, these two basic aspects of the human adaptation capability condensed into Science and Engineering, respectively.

Since then, humans have tried to simplify and automatize tasks and

procedures that take place in different contexts of their societies. This focus on the simplification and automation led to the creation of research fields that aim at developing artificial systems that try to mimic and improve the capabilities of the human brain for identifying and learning patterns from observation as well as solving problems. In recent history, the advances in different branches of knowledge such as computer science, mathematics or biology made it possible to develop the aforementioned *intelligent* artificial systems.

One of the branches that have achieved a major level of capability and maturity is the field of Machine Learing (ML), where the Artificial Neural Networks (ANN) are presented as one of its most flexible and powerful subjects of study [42]. From a theoretical perspective, an ANN is an interconnected system of mathematical functions called neurons that process the information contained in data (occasionally harvested by sensors [25], [4],) in order to yield a specific output. These systems have become popular in the recent years due to its ability for learning from data, identifying patterns or being able to understand language, understand visual information or emulate the biological nervous system in motor control. Some remarkable examples of ANN are, for instance, the Convolutional Neural Networks (CNN) [16], which are really good at processing visual information like in the case of YOLO [37], Transformers [41], which are able to understand the complex human language, or the Spiking neural Networks (SNN) [22], which are famous for their bio-inspired working mode and their energetic efficiency. Indeed, these NN are some of the SoA structures in contexts such as motor control [6].

These ANN belong to different families based on their architectures, which define how they process stimuli and obtain information from data sources of different nature. However, their performance does not depend

just on the architecture, these algorithms need to learn. In the context of NN, the learning or training process is a complex and expensive procedure (in time and energy terms) based on the exposition of the Neural Network to a specific set of data in order to identify patterns [16]. This process requires a method that allows the convergence of the relationships among neurons (synaptic relationships) and, also, a suitable dataset which must contain useful information for facing the task.

This training process allows the ML algorithms and, in particular, ANN, to learn and identify underlying patterns in the data, which will provide the algorithm with the required knowledge for properly tackling the target task. At high level, this training stage is therefore constituted by two main elements: on the one hand, the optimizer, which is the algorithm that makes the model learn and, in the particular case of ANN, this is done by strengthening or weakening the connections among neurons. On the other hand, the other relevant element or entity is the dataset, which should be a representative sample of the inputs that are going to be fed to the algorithm during the conduction of the target task, once the learning has been completed. For the model to learn, it is a necessary condition that the dataset contains information that can be used for facing the task to be accomplished, otherwise, the model will learn nothing regardless the power or structure of the model. Besides, some factors such as the dimension of the space of variables of the dataset give rise to pathological behaviours during training, like the curse of dimensionality [2], [43], [5], which increases the difficulty of the learning of the model due to the perturbation of the meaning of the distances in spaces of high dimension. In this context, it is fair to ask when a dataset is good enough for training a model in a specific task.

For giving an answer to the previous question, it can be asserted that

developing metrics and measures that could shade some light on the information contained in a dataset would be an useful contribution. However, the wide variety of complex tasks that the ML algorithms are able to face nowadays, together with the different nature and shape of the data to be processed, makes it difficult to provide just one single general metric. Thus, for instance, good properties on structured datasets for classification tasks rely on the geometrical structure of the data, while the properties of a dataset constituted by spiking signals mostly rely on variations on the spiking frequencies.

Hence, although there exists different techniques and methods for evaluating some characteristics of a dataset before the training is conducted [39], [15], [24], [26], [30], in most cases it is needed to run the training in order to find out how the model performs.

In this PhD Thesis, several relevant contributions are made by developing tools in the form of mathematical measures and techniques devoted to evaluate the information contained in datasets, so that a ML model can be properly trained for performing a specific task. It is presented as a journey along different scenarios that start with the use of classical information analysis techniques, visits the world of Biologically-Inspired sensors where a tool for evaluating spiking signals is created and last but not least, it finishes with the definition of different methods and metrics for evaluating the contained information in the data for classification tasks. Besides, after the analysis of the information in these different challenges, the different contributions are tested in experiments by means of the application of ML algorithms to industrial procedures in different manufacturing processes.

The first of the contributions of this PhD Thesis is presented in Chapter 2. It is described the use of an ANN for detecting uncalibrations of sensors in a series of cleanrooms of an important company of the Pharmaceuti-

cal sector. Previous to the application of this algorithm, several classical techniques of data analysis were used for measuring the suitability of the available data, which were collected by an already existing sensor system. The techniques used in this first chapter analyse several aspects of the data such as the existence of normal and abnormal values (outlier detection) or correlation analysis.

The second Contribution of this PhD Thesis is presented in Chapter 3. In this Chapter, it is provided a methodology and a ML algorithm for extracting indirect measures from mechanical features by using an artificial cochlea [21], [33], [13]. The algorithm has been trained, tested and deployed within a production line of an important company from Aerospace sector. Due to the spiking nature of the signals collected by this artificial cochlea, it has been developed a completely new technique called Pulsing Histogram Energy (PHE) for finding patterns in the data that are no visible at the bare eye. As it will be seen, this technique is based on the concept *signal energy* and its evolution along time. Besides, other techniques coming from some other mathematical contexts has been used for processing the data, reducing its vectorial dimension and making it compatible with a ML model.

Finally, the last contribution of this PhD Thesis is presented in Chapter 4. This Chapter is completely devoted to the creation of a series of metrics for measuring the information contained on datasets aimed for training ML algorithms on classification tasks. These metrics are based on theoretical principles coming from different mathematical areas such as Shannon's information theory, among others. After the definition of the metrics, its capability for measuring the suitability of the dataset for classification tasks is measured by training different standard models for classification tasks such as Multi-Layer Perceptron (MLP) or XGBoost (XGB). These models are trained on different public datasets like the Iris dataset or the Mobile

Price dataset. Then, the performance of the different models is compared to the evaluation of the dataset by the different developed metrics.

Chapter 2

Classical Methods

As presented in the introduction, this first chapter presents one of the main contributions of this PhD Thesis, this is, the application of a ML model for detecting the uncalibration of sensors. However, before the model could be trained, it was necessary to find out if the available data were good enough and contained the necessary information for the potential algorithms to properly identify the status of the sensors. In this case, different classical techniques were used. First, it is provided a description of the context of the challenge as well as an introduction to the associated data. After that, it is presented a formal introduction of the different techniques that were used for the preliminary data analysis as well as some examples of application to the real data. Finally, it is presented the whole development conducted for facing the whole challenge. Moreover, for deeper details the obtained results can be seen in the associated published work [31].

2.1 Introduction

The context of application of this first contribution consists in detecting or predicting uncalibrations in the sensor system of a set of cleanrooms located on the facilities of an Pharma company. Each room is endowed with three sensors of different type, namely, temperature, humidity and pressure. More precisely, the installed sensors are:

- nSens-HT-ENS: A combined sensor that includes a temperature sensor and a humidity sensor.
- Pascal ST/Z: A pressure sensor.

Each sensor is able to send, approximately, two samples per second. Regarding the environmental condition setting within the rooms, these conditions are generated by a HVAC system. This is a commercial Air Conditioning System. A relevant aspect of the whole setting for this monitoring system is that the information related to the HVAC system cannot be accessed directly from the cleanrooms. Besides, the set points for the different conditions are no controlled from the different rooms. This is important because the real generated condition cannot be directly accessed and therefore a statistical comparative method, like the T-Test, is not an option for detecting the uncalibrations. The humidity and temperature are supposed to be constant and the same for all cleanrooms. However, several aspects as the dimension of the rooms, or the temperature irradiated by several devices as washing machines or compounding machines can affect the conditions within the rooms. Because of these facts, temperature or humidity are not exactly the same in all rooms.

The dataset contained the following samples coming from a whole year of measurements.

Floor	Sensor Type	Number of sensors	Number of samples
2	Temperature	17	8935200
2	Humidity	17	8935200
2	Pressure	24	12614400
-1	Temperature	11	1100000
-1	Humidity	11	1100000
-1	Pressure	11	1100000

Table 2.1: Dataset summary.

An example of the behaviour of the data collected by the sensors can be seen in Figure 2.1

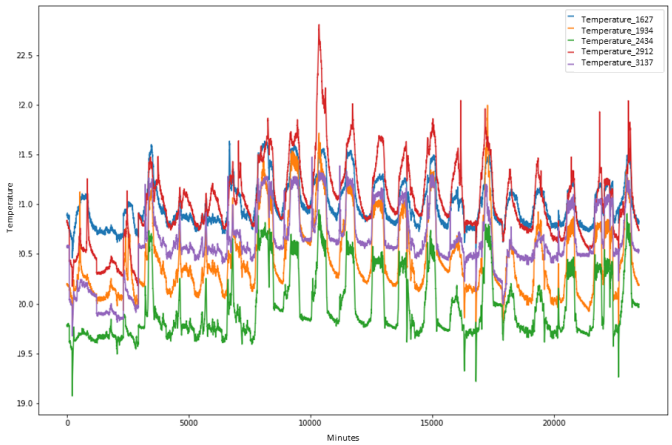

Figure 2.1: Time series associated to temperature sensors in cleanrooms 1627, 1934, 2434, 2912, 3137. This time series were collected during September 2019.

Now, the techniques applied to analyse and evaluate some properties of

the data are going to be presented.

2.2 Analysis Methods

Several techniques were used for analysing the data available in this challenge. All these techniques belong to what is considered as the basic set of analysis tools for identifying useful information within the data. These techniques are outlier detection, correlation analysis and, also, some techniques coming from classical time series analysis like stationarity analysis.

2.2.1 Outlier Detection

In the context of statistics and data analysis, an outlier is an observation that is distant or abnormal in statistical terms. This means that, provided an underlying random variable, some of the observed data are expected to appear with a very low probability, and therefore the occurrence of some of them should not be representative and lead to wrong conclusions. Therefore, prior to any further analysis or study the data must be cleaned up.

For the challenge faced in this first contribution, two different approaches to outlier detection are presented. Firstly, a preliminary sensor-wise outlier analysis, which is conducted based on some descriptive properties of the collected data.

Secondly, an analysis on the joint behaviour of the whole set of sensors is implemented. This second analysis is based on the Mahalanobis distance, which is devoted to the detection of abnormal values even though these could be not extreme values for any sensor.

Sensor-wise outlier analysis

In this case, a sensor-wise analysis for detecting potential outliers is conducted. These analysis are based on tests that rely on the dispersion of the data for detecting the outliers.

Test 1

Let $X = \{x_1, x_2, \cdots x_n\} \subset \mathbb{R}$ be a sample. Let $\overline{x}$ and σ denote the mean and the standard deviation of the sample X. If the data is assumed to be generated from a normal distribution, then, a value x is considered to be an outlier if

$$x \notin (\overline{x} - 2.5 \cdot \sigma, \overline{x} + 2.5 \cdot \sigma).$$

The reason why the factor 2.5 is used is because under the normality hypothesis, the 99% of the data belonging to a normal distribution fall inside this range.

Test 2

In case that the sample exhibits a skewed behaviour another approach is more suitable. Let $X = \{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}$ be a sample. Let Q_1, Q_2, Q_3 denote the quartiles associated to this sample. Then define the interquartile range as

$$IQR = Q_3 - Q_1.$$

Then, a value x is considered to be an outlier if

$$x \notin (Q_1 - 1.5 \cdot IQR, Q_3 + 1.5 \cdot IQR).$$

Multi-dimensional outlier analysis

Since there could exist an underlying dependence among the behaviour of the whole set of sensors, it is important to take into consideration that some

of the values of the whole set of measurements can be abnormal although none of the single measurements is abnormal by itself. For the detection of such kind of outliers it is necessary the usage of a tool that takes into account the joint behaviour of all the time series. In this PhD Thesis, the Mahalanobis distance has been used [27], [32], [28], [17]. Before the metric for detecting these outliers can be introduced, several auxiliary definitions are needed.

Definition Let X be a random variable. Let $\mu = \mathbb{E}[X]$ where $\mathbb{E}$ is the expectation operator. Then, the variance of X is defined by

$$Var(X) = \mathbb{E}[(X - \mu)^2].$$

Definition Let X and Y be two random variables. The covariance of X and Y is defined as

$$cov(X, Y) = \mathbb{E}[(X - \mu_X) \cdot (Y - \mu_Y)],$$

where $\mathbb{E}$ stands for the expectation operator and μ_X, μ_Y stand for the expectation of X and Y respectively.

Definition Let $X = (X_1, X_2, \cdots X_n)$ be a vector of random variables each with finite variance and expected values. Then the covariance matrix K_{XX} is the matrix whose entries are given by

$$K_{XX}[i, j] = cov(X_i, X_j).$$

Definition Let $X = (X_1, X_2, \cdots X_n)$ be a vector of random variables on $\mathbb{R}^n$ each with finite variance and expected values. Let $\mu = (\mu_1, \cdots \mu_n)$

be the vector of expected values and let K be the covariance matrix which is assumed to be positive-definite. Then, the Mahalanobis distance of a point $x = (x_1, \cdots x_n)$ to X is given by,

$$d_M(x, X) = \sqrt{(x - \mu)^T K^{-1} (x - \mu)}$$

The Mahalanobis distance normalizes the distance along the different directions of maximum amplitude. For this, it takes into account the dispersion of the data over these axes, whose direction and amplitude are inferred by the covariance, and which will not coincide in general with the main axes defined by the reference basis of the vector space. In Figure 2.2 it can be seen how the example data exhibit different dispersion values for two principal axis which not coincide with y and x axes.

As it can be seen, the example point in green can be detected as an outlier by the Mahalanobis distance but it would not be detected just by using the Euclidean distance. Mahalanobis distance provides a more accurate understanding of what an abnormal behaviour is.

Outlier analysis implementation

Once the different used techniques have been introduced the obtained results for some of the sensors are presented. Figure 2.3 shows the raw signal for different temperature sensors during Septemeber 2019.

In this case, most of outliers exhibit very extreme values with respect the normal behaviour of the time series. These values can be easily detected by Test 2 described in the section devoted to the sensor-wise analysis.

After filtering the time series by means of the application of test 2 and also the Mahalanobis distance, the graphic shown in Figure 2.1 is obtained. In this case, 1012 samples were removed. The same analysis is conducted for

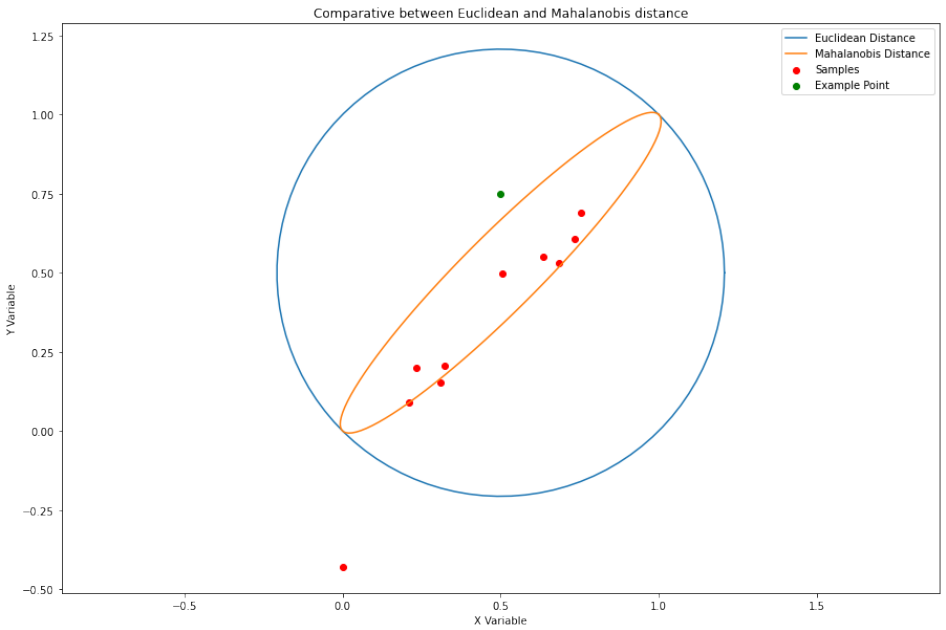

Figure 2.2: This figure shows the differences in the behaviour between Euclidean and Mahalanobis distance.

humidity and pressure sensors, where 3237 and 3222 samples were removed respectively for the month of September. This same procedure was repeated for the whole year so that the data can be analyzed in detail with the techniques that are going to be exposed in next sections.

2.2.2 Correlation Analysis

The correlation analysis is one of the most common types of measures for obtaining insights of the data. This kind of analysis measures if there exists some kind of statistical relation or dependence among a set of pro-

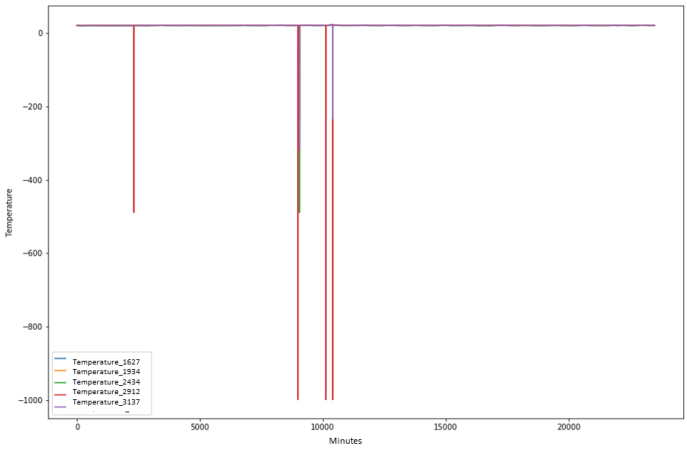

Figure 2.3: This figure shows several outliers read by the sensors in cleanrooms 1627, 1934, 2434, 2912, 3137 along September 2019. These outliers can be detected by Test 2 described in the sensor-wise outlier analysis.

vided variables. This relation may be of different nature depending on the behaviour of the variables, but mainly four main categories can be distinguished, namely, linear, negative linear, curvilinear or non correlation. This correlation is measured by means of different specific tests. It is important to note that a negative output of one of these specific tests does not imply the non-existence of correlation of some other type.

The behaviour observed in the sample shown in figure 2.1 suggest that could exist some kind of linear correlation among the measurements coming from the sensors located along different cleanrooms. Besides, this would be coherent with the fact that there exist just one single common set point, what would yield small discrepancies in the behaviour associated to the dif-

ferences in the configurations of the rooms, for instance, size or equipment. In case of high correlation values, this could be used for detecting potential uncalibrations if some of the coefficient values suddenly fall.

The first measure to be applied to the data for the search of information was the Pearson Correlation Coefficient, since this is a correlation coefficient specifically devoted to measure linear correlations.

Definition Given two random variables X and Y, let μ_X, μ_Y be the expectation of X and Y and σ_X, σ_Y be the standard deviation of X and Y respectively. Then, the Pearson Correlation Coefficient $\rho_{X,Y}$ is defined by

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \cdot \sigma_Y},$$

where and $cov(X,Y)$ stands for the covariance.

Correlation tests implementation

Once the correlation test that is going to be used has been introduced the obtained results for some of the sensors are presented. Tables 2.2.2 and 2.2.2 show the correlation values obtained for sensors in Figure2.1.

Sensor	T1627	T1934	T2434	T2912	T3137
T1627	1	0.653	0.997	0.997	0.998
T1934	0.653	1	0.653	0.652	0.653
T2434	0.997	0.653	1	0.995	0.996
T2912	0.997	0.652	0.995	1	0.996
T3137	0.998	0.653	0.996	0.996	1

Table 2.2: Sensor sample correlation. October 2019.

As it can be seen, the values obtained in October, which are shown in

Table 2.2.2, exhibit reasonably high correlation values for Pearson coefficient.

Sensor	T1627	T1934	T2434	T2912	T3137
T1627	1	0.840	0.556	0.001	0.602
T1934	0.840	1	0.818	0.0	0.841
T2434	0.556	0.818	1	0.0	0.996
T2912	0.001	0.0	0.0	1	0.002
T3137	0.602	0.841	0.996	0.002	1

Table 2.3: Sensor sample correlation. September 2019.

However, these values are much smaller in September for the same set of sensors. These results suggest that the potential linear correlation does not hold allover the year, what implies that there are undergoing procedures that affect the understanding of the behaviour of some sensors from the observation of some other sensors and therefore they cannot be used as an uncalibration indicator.

2.2.3 Time Series Analysis

The last set of techniques to be applied to the data are related to its nature as time series. More precisely, the techniques depicted and applied within this section are devoted to analyse if the different time series associated to the data collected by the sensors are stationary. Intuitively, a process described by a time series is stationary when the underlying probability distribution associated to the process does not change along time. For a formal description, next definitions are needed,

Definition Let T be a discrete index and let $X = \{X_t\}_{t \in T}$ be a sequence of random variables defined on a probability space. In this context, X is

said to be a stochastic process.

Please note that in the context of stochastic process T usually has the meaning of time.

Definition Let $\{X_t\}_{t\in\mathbb{R}}$ be a stochastic process. Let $n \in \mathbb{N}$ and let $F(x_{t1+\tau}, \cdots, x_{tn+\tau})$ denote the distribution function of the joint distribution of $\{X_t\}_{t\in\mathbb{R}}$ and times $t1+\tau, \cdots, tn+\tau$. Then, $\{X_t\}_{t\in\mathbb{R}}$ is said to be strictly stationary if,

$$F(x_{t1+\tau}, \cdots, x_{tn+\tau}) = F(x_{t1}, \cdots, x_{tn}),$$

for all $\tau, t1, t2, \cdots tn \in \mathbb{R}$ and all $n \in \mathbb{N}$.

The motivation behind the application of a stationarity analysis is based on the fact that temperature and humidity conditions are supposed to be constant and the same for all the cleanrooms along the whole year. Therefore, a change in the stationarity of the time series, i.e. a change in the distribution function that models the observed data, could be associated to a potential uncalibration event. Before this fact could be considered as a suitable tool for detecting a potential uncalibration the stationarity of the time series must be guaranteed.

Definition Let $Y = \{y_t\}_t$ be an stochastic process and let us suppose that can be written as an autorregresive process of order p.

$$y_t = a_1 \cdot y_1 + a_2 \cdot y_2 + \cdots a_p \cdot yp + \epsilon_t,$$

where ϵ_t is a process with mean $\mu = 0$ and constant variance σ^2, i.e. white

noise. Consider the characteristic equation given by

$$m^p - a_1 \cdot m^{p-1} - a_2 \cdot m^{p-2} - \cdots - a_p = 0.$$

In that case, the process is said to be stationary if $|m| < 1$, and non stationary if $|m| = 1$, respectively. In that case, m is said to be an unit root.

It is very important to note that the existence of unit roots imply the non-stationarity of the stochastic process. This can be seen in the next example. Consider the stochastic process with associated autoregressive model given by

$$y_t = y_{t-1} + \epsilon_t.$$

The associated characteristic equation is given by,

$$m - 1 = 0.$$

Therefore, this stochastic process has an unit root. It can be observed that by substitution the process can be written in the following way,

$$y_t = y_0 + \sum_{i=1}^{t} \epsilon_i.$$

By computing the associated variance at an instant t it is obtained the following expression,

$$var(y_t) = \sum_{i=1}^{t} var(\epsilon_i) = t \cdot \sigma^2.$$

Since this expression depends on t, the associated distribution varies with

time, thus the process is not stationary.

Once the concept of unit root and its relation to stationarity has been presented, it is necessary to find a method for identifying the existence of an unit root for a given stochastic process. For the challenge presented in the current chapter, the Augmented Dickey's Fuller test was applied. This test stablishes a hypothesis test where the null hypothesis is that the time series can be represented by a unit root and therefore has some time-dependent structure which makes it non-stationary. The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.

$$\begin{cases} H_0 & \text{The stochastic process has an unit root.} \\ H_1 & \text{The stochastic process has not an unit root.} \end{cases}$$

For a given threshold the null hypothesis is rejected whenever the obtained p-value is under the provided threshold. The disquisition where the technical details of this test are presented can be found in section 17.7 in [19].

Stationarity Tests implementation

After the introduction of the basic theory and the tests that are going to be implemented for analysing some of the properties of the data, the results of the application of the tests for some of the sensors are presented.

In general, it has been observed that stationarity does not hold. This is probably due to the fact that there exists machinery in the rooms whose performance may affect the conditions in the room, and this performance is not constant along time.

This fact can be seen in table 2.2.3, where it is presented the p-values of the Dickey's Fuller test at a significance of the 5%. These P-values are

associated to the application of the test to the month of July of 2019.

Sensor	P-Value
T1627	0.6268
T1934	0.5472
T2434	0.4763
T2912	0.2854
T3137	0.4265

Table 2.4: P-Values associated to the application of the Dickey's - Fuller test to the sensors. July 2019.

It can be seen that the p-values are over the defined threshold and therefore the null hypothesis is accepted. These results suggest that the data is non stationary. Therefore, the change from stationarity to non-stationarity cannot be used as an indicator for detecting potential uncalibrations.

After applying the tests described in the previous subsections, it can be observed two relevant main points, namely, that linear correlation and stationarity cannot be ensured for a whole year. Therefore, simple correlation or stationarity cannot be used as indicators of a potential drift associated to an uncalibration event. However, as it has been seen, correlation or stationarity and non-stationarity can be seen for different temporal frames. This fact suggests that there could be some underlying patterns of higher complexity levels that rule the behaviour of the signals of interest. Section 2.3 introduce one of the main contributions of this PhD Thesis. It shows the architecture of a system based on ML that is capable of detecting drifts in data associated to uncalibrations based on patterns hidden in the data analysed in this section.

2.3 Methodology

In this section, the architecture of the implemented system, which is capable of detecting sensor uncalibration, is presented. The whole system has been developed using Python. Different libraries such as numpy, scipy and pandas have been used for the implementation of the filtering processes. For the implementation and training of the model TensorFlow2 [1] together with the Keras [23] functional API have been used.

The approach is based on ML techniques and it follows the block diagram shown in Figure 2.4. The architecture is based on an ANN which, during the training phase, learns how the sensors behave. Then, in the inference stage, the ANN is able to predict if sensors are becoming uncalibrated. The way data is processed is summarized in the following steps:

- The data is collected by the sensors. It is important to note that it comes from only one type of sensor, either temperature, humidity or pressure. This means that the network that processes this data is sensor-dependant.
- The data is filtered. Here, the outlier detection techniques described in subsection 2.2.1 and other pre-processing techniques are applied.
- The mean of all measured values in a specific time is computed and used as an estimation of the set point defined in the HVAC system.
- The computed mean value is fed to the ANN.
- The ANN infers the set of measurements that produced the input mean value.

- The inferred set of measures is compared with the real set of measurements. If the discrepancy exceeds the confidence interval, then it is considered a rejection.
- The rejection density is computed for a fixed time window. When rejection density achieves values that exceed a specific threshold, an uncalibration takes place.

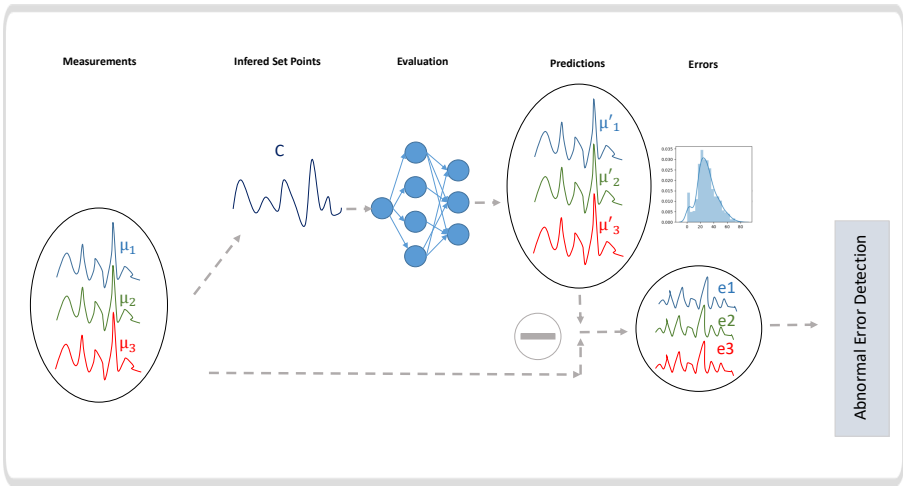

Figure 2.4: Block diagram of the approach presented. Five main stages can be observed: the measurement stage, the estimation of the set point, the evaluation of the estimated set point by the NN and the generation of the prediction and, finally, the computation of the error with respect to the real value.

2.3.1 Neural Network

The proposed ANN model is a MLP that consists of an input layer with one neuron, five hidden layers with three hundred neurons each and an output layer with as many output neurons as sensors. The ReLu function was used as activation function in the hidden layers. Since we are working in a regression model, the Linear function was chosen as the activation function for the output layer. The model was trained during 30 epochs using the Adamax optimizer. In order to determine the optimal number of layers and neurons per layer, Scikit-Learn's Grid Search algorithm (GridSearchCV) [35] was used. During the training stage, validation sets were used in order to control over-fitting. These same hyperparameters were also used for the rest of experiments performed within this work, as in the case of Transfer Learning. As it will be seen later, Mean Squared Error and Mean Absolute Error has been used as loss functions. It is important to note that the chosen loss function has a deep impact on the statistical behaviour of the prediction errors. After training the network with the mean value of the pre-processed data from the sensors, the network is able to decompose the input stimulus into the predicted value for each sensor. This is the expected behavior that each sensor should have, based on that input stimuli.

2.3.2 Error Computation

This is the final step of the proposed approach. In this stage, the difference between the real measured value obtained from the sensors and the ones predicted is computed, evaluated and classified as normal or anomalous. This evaluation is based on the work of [3] where Hierarchical Temporal Memory [20] was used to predict the next measurement. That prediction was then compared to the real measurement and residuals were computed.

In our approach, goodness of fit tests were applied on the errors obtained during the training stage. Thus, the underlying distribution of the error could be used to compute specific confidence intervals with the desired signification for each sensor. On the one hand, the absolute mean error of the sensors measurements properly fits a normal distribution. On the other hand, the mean squared error fits an exponential distribution.

Then, confidence intervals were generated by using the normal distribution associated to the error of each sensor. This choice was based on the fact that the absolute mean error was approximately equal to the uncalibration that was taking place. These confidence intervals are associated to each sensor and they represent the behavior of each sensor from the point of view of the system. They are known as the resolution of the system per sensor.

Finally, a rejection takes place whenever the value exceeds the bounds of the confidence interval. The density of rejections is computed as the rolling ratio between the number of rejections within a window of a fixed time length (with 1440 minutes as default value, which corresponds to the number of minutes in a day) and the length of the window.

The rejection density is the variable that triggers the uncalibration warning. An uncalibration is said to take place whenever a threshold is reached during a specified amount of time.

The proposed architecture is not able to determine if a sensor is calibrated or not. The expected behaviour is that the NN detects small differences between the current measurement and the one from which it learnt. Thus, in the eventuality of a maintenance task in any sensor, the architecture proposed might detect this new event as a potential uncalibration. This uncalibration event will remain active until this new condition of the sensor is re-learnt as 'calibrated' by the model.

The standard procedure to overcome this issue is to re-train the model under these new conditions. Thus, this process would require a very high temporal cost, since collecting data during almost a whole year would be necessary. For this reason, the use of transfer learning represents a feasible solution. A small amount of data is needed to teach the new condition to the NN by using transfer learning.

2.3.3 Transfer Learning

Transfer learning [7, 34] is a ML technique aimed at specializing ML models with a minimum amount of data. This technique consists of two main steps. The first step is training a model for a more general task conceptually related with the target or specialized task. For instance, the target task could be to identify a specific face in a picture and, in that case, the general task would be training a model for general face identification. The second step, would be to re-train a subset of layers of the model with a training dataset made out of elements corresponding to the specialized task (a specific face in the example presented).

In the case of the system presented in this section, multiple scenarios can benefit from transfer learning. These scenarios can be summarized as follows: changing the ground truth value for all sensors or for a subset of them, including new sensors and adapting to some other environments with, possibly, insufficient information available.

Transfer learning is applied by dividing the original dataset into three smaller subsets, namely, a training subset for the original model, which will be called model A, a re-training subset for the application of transfer learning (model A will be renamed to model B after this process) and, finally, a testing subset for both models. Regarding the training subset, it

contains 70% of the datapoints. This stage corresponds to the training for the general task. Regarding the subset devoted to be used for re-training the model and, thus, for the application of transfer learning, it contains the 20% of the datapoints. The amount of required datapoints can be reduced to up to 2.5% of the total (10000 datapoints per sensor) in the case of temperature and humidity. Finally, the testing subset includes 10% of the datapoints. This subset was used to test the performance of the neural network, both before and after the application of the transfer learning. A constant offset was added in the last part of the test set in order to simulate the recalibration due to a maintenance task. It is important to note that this offset is not applied to the whole test set in order to properly see the change of conditions and the associated evaluation by both, model A and model B.

As mentioned, one of the main purposes of the application of transfer learning is the reduced amount of necessary samples for training a model. This amount is directly associated to the number of parameters to retrain, which, at the same time, is closely related to the number of layers to retrain. The nature of the target task, i.e., how abstract it is, and where (in which layers) the concepts learnt are located within the NN, are crucial factors which will determine the amount of required data. During the current research, transfer learning has been applied on the very last hidden layer of the NN, i.e., only the weights connecting the last hidden layer with the output layer has been retrained. The obtained results suggest that training this subset of weights was enough for the existing requirements.

2.4 Results

This section shows the results obtained using the methods presented in section 2.3. Two different experiments were conducted: (1) to evaluate the capability for the detection of uncalibrations of the current approach, through performance evaluation experiments; (2) to test the flexibility and scalability of the system proposed, through transfer learning experiments.

2.4.1 Performance Evaluation Experiments

These experiments were conducted to test the detection of uncalibrated sensors using the architecture presented. The dataset used is the one shown in section 2.1. Uncalibrations were introduced in the testing set as drifts of different nature (e.g linear, exponential or logarithmic) across time and in different temporal frames (for instance, at the beginning, in the middle and at the end of the year).

An uncalibration is considered to be detected once the rejection density value has reached a predefined threshold, which depends on the sensor, and whose mean value is 0.8. Futhermore, this threshold has to be reached during a predefined period of time, which also depends on the sensor, and can be defined as two weeks, since it is enough time for all sensors and it is a reasonably low period of time for uncalibration detection. An example of the application of this technique can be seen in Figure 2.5 where rejection density for both the upper and lower bound of the confidence interval are shown.

The second and third plots show the rejection density values. It can be seen that these values stay low whenever a uncalibration is not taking place. Conversely, when a uncalibration takes place, the rejection density

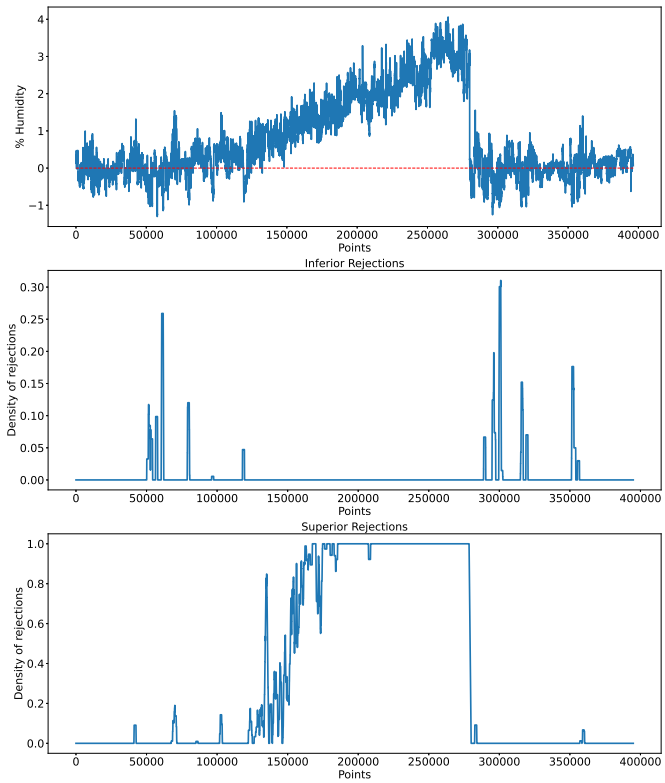

Figure 2.5: Uncalibration detection through rejection density. In the upper subplot, the blue trace shows the error corresponding to a linear uncalibration and the red dashed line stands for the error zero value. Middle and lower sublots show the upper and lower rejection density respectively.

reaches values close to one.

Regarding the performance of the approach, uncalibrations were detected for all different types of sensors. However, two different scenarios can be distinguished, namely, a first optimistic scenario, where the uncalibration is detected within the tolerance ranges defined by the quality

requirements of the pharma industry (i.e. $\pm 0.5°$,$\pm 3\%$ and ± 0.5P in temperature, humidity and pressure respectively), and a non-optimistic scenario, where the uncalibration is detected over the tolerance range. For all those sensors that are in the optimistic scenario, an estimation of the remaining time until the tolerance range gets exceeded can be provided.

Thus, for the case of the temperature sensors, the proposed approach was tested on the available data coming from temperature sensors over 17 rooms. All uncalibrations were detected for all sensors. In the best scenario, the architecture detected uncalibrations associated to deviations of 0.25°C (tolerance 0.5°C). This accuracy was reached for 16 rooms out of 17 of the second floor. In the worst scenario, the uncalibration was detected for deviations of, approximately, 0.5°C.

For the humidity sensors case, the proposed method was tested on the available data coming from humidity sensors over 17 clean rooms. All uncalibrations were detected for all sensors. In the best scenario, the algorithm detects uncalibrations associated to deviations of 2% (tolerance 3%). This accuracy was reached for 14 rooms out of 17 of the second floor. In the worst scenario, the uncalibration was detected for deviations close to 3%.

Finally, for the pressure sensors case, the proposed method was tested on the available data coming from over 24 pressure sensors. All uncalibrations were detected for all sensors. In the best scenario, the algorithm detected uncalibrations associated to deviations under 0.5 Pascals (tolerance 0.5 Pascals). This accuracy was reached for 6 out of 20 rooms of the second floor. In the worst scenario, the uncalibration was detected for deviations close to 1.5 Pascals.

Some other ANN architectures were tested during the development of this research. The same experiments were conducted using a Wide & Deep Neural Network for temperature and humidity sensors and Recurrent Neu-

ral Network (RNN) for the pressure sensors. The architecture of the Wide & Deep Neural Network was based on the MLP used on the implemented approach, but also includes an input layer per sensor, so that the precise information coming separately from each sensor can be included. This can be seen in Figure 2.6.

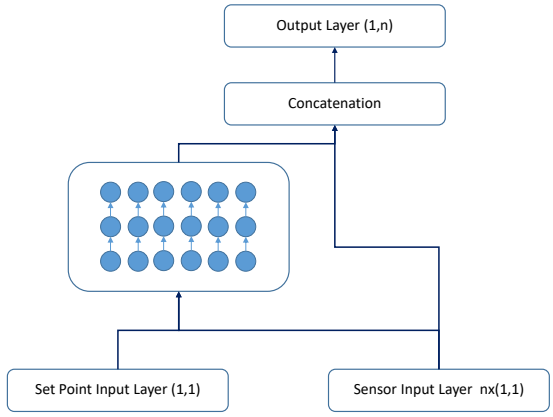

Figure 2.6: Wide and Deep model diagram. In this figure, the tested architecture of the wide and deep model is shown. It can be seen $n+1$ different input layers; an input layer with one single neuron, which is connected to a hidden block of five hidden layers with three hundred neurons each. This input layer takes the estimated global set point as input. Next, n input layers with one single neuron each. Each input layer takes the measurements coming from the different associated sensors as input. Finally, these input layers are concatenated to the output of the hidden block and connected to the output layer, which has a neuron per sensor.

Wide and Deep architecture yielded better results for most of the rooms.

However, for two of them, the uncalibrations were not detected. It has been observed that, for these two rooms, how the sensor is supposed to behave under a specific condition is not learnt by the neural network, and the identity function is approximated instead. Thus, when an anomaly of any kind was introduced on the input signal, this same value was retrieved by the NN and, therefore, the uncalibration was not detected. It has been observed that the worst results were obtained from these two rooms independently of the architecture. This fact suggests that the local conditions inside these rooms had a deep impact on the learning capabilities of the different models.

Regarding the RNN, one single Long-Short Term Memory (LSTM) layer was used, containing 164 units and the ReLu activation function. As output layer, a Dense layer with as many units as sensors was used. In this case, the Linear function was used as activation function. The model was trained by using the Adamax optimizer. In this case, uncalibrations were not detected either. This was probably caused due to the long-term nature of uncalibrations. Uncalibrations are long term phenomenons, thus, the deviation associated to an uncalibration will be observed during long periods of time. Hence, the deviation will be included as relevant by LSTM neurons during the information inclusion stage, that is, when the input gate evaluates what information is relevant and what information is not. In this case, the learning of this long-term phenomenons by the NN provoke the worst results.

2.4.2 Transfer Learning Experiments

These experiments were conducted to test the capability of the architecture for learning a new condition with the less temporal cost. Thus, these

experiments were split in three categories: change in global or specific conditions, adding new sensors and adapting to some other environments with insufficient information available. To evaluate the transfer learning technique, the error obtained for the main model (model A) and the retrained model (model B) under the specific new condition were compared.

Change of conditions

These experiments were conducted to test the capability of the architecture proposed for learning a new condition as the current calibration status for one or more sensors. In the case of the individual offset, in Figure 2.7 two different phases of the error obtained from model A are shown. The first one corresponds to the model evaluation under normal conditions. The second one corresponds to the model evaluation under the generation of a specific offset. It can be seen how the error is displaced, approximately, three units which is the same amount of the generated offset.

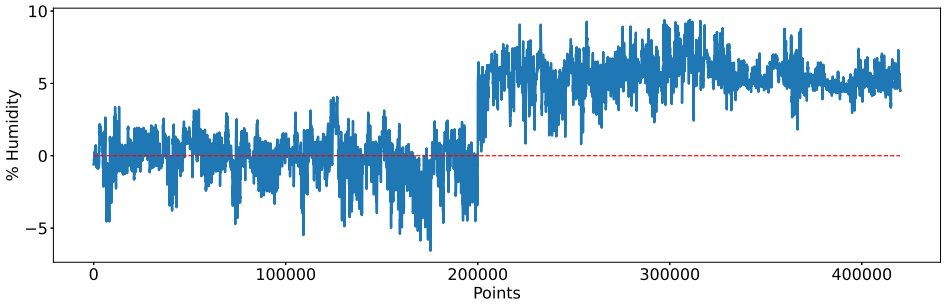

Figure 2.7: In this figure, the error associated to a scheduled recalibration can be seen. The blue trace shows the error and the red dashed line stands for the error zero value. The recalibration takes place at the mid point of the figure. Since transfer learning is not applied, once the recalibration takes place, the error immediately increases.

However, in model B, it can be seen how the error returns to the same range of values as previously. This behaviour suggests that the expected results are achieved. This experiment was reproduced for different rooms and the different types of sensors, obtaining similar results. It can be observed in Figure 2.8.

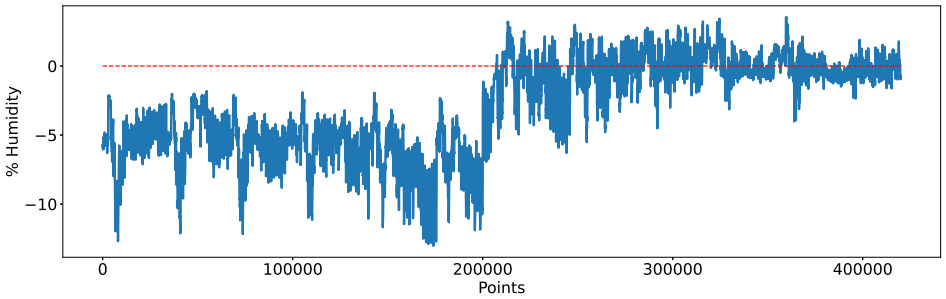

Figure 2.8: In this figure, the blue trace shows the error associated to a scheduled recalibration and the red dashed line stands for the error zero value. In this case, transfer learning is applied, thus, once the recalibration takes place, the error immediately decreases. It is important to note that this model is not applied in the previous moments to the recalibration task.

It should be noted that the minimum number of datapoints to properly use transfer learning is around ten thousand samples. This number of samples corresponds, approximately, to one week of data collection.

Regarding the generation of an offset for the whole set of sensors, the model did not detect any change in its behavior. It suggests that, if the behaviour of the whole set of sensors is equally changed, then, no effect on the joint behaviour can be detected. Thus, no uncalibration is taking place from the point of view of the model. This fact proposes an interesting question regarding how the joint behaviour could be learnt.

Adding new sensors to the model

These experiments were conducted to determine the time that the architecture takes to learn and include the behaviour of new sensors in the loop. These tests were made using the humidity and temperature sensors.

A first dataset with data coming from 13 sensors out of 17 was taken as main set. For this set of sensors, $o_1 = 322677$ datapoints were available. In this case, model A was trained with this dataset. Then, model B was re-trained with data coming from both the initial set of 13 sensors and a set of 4 new sensors. The obtained results show that uncalibrations on the new set of sensors can be properly detected by model B. Figure 2.9 shows how uncalibrations are detected in the new sensors that were included for the temperature case.

Adapting to some other environment with insufficient information available

These experiments were conducted to test if the architecture could be trained with data coming from one specific location and then be implemented on a different location after a transfer learning process.

In this case, model A was trained with data coming from the second floor with 17 sensors of both types, temperature and humidity. Then, model B was re-trained with a small amount of data coming from the basement. Approximately, 80000 datapoints were used. The obtained results show that uncalibrations are detected by model B in this new environment. Furthermore, the problem of the lack of information was solved by applying this method. Figure 2.10 shows an example of how uncalibrations were detected by model B in this specific context.

The obtained results shows that transfer learning is a valid technique

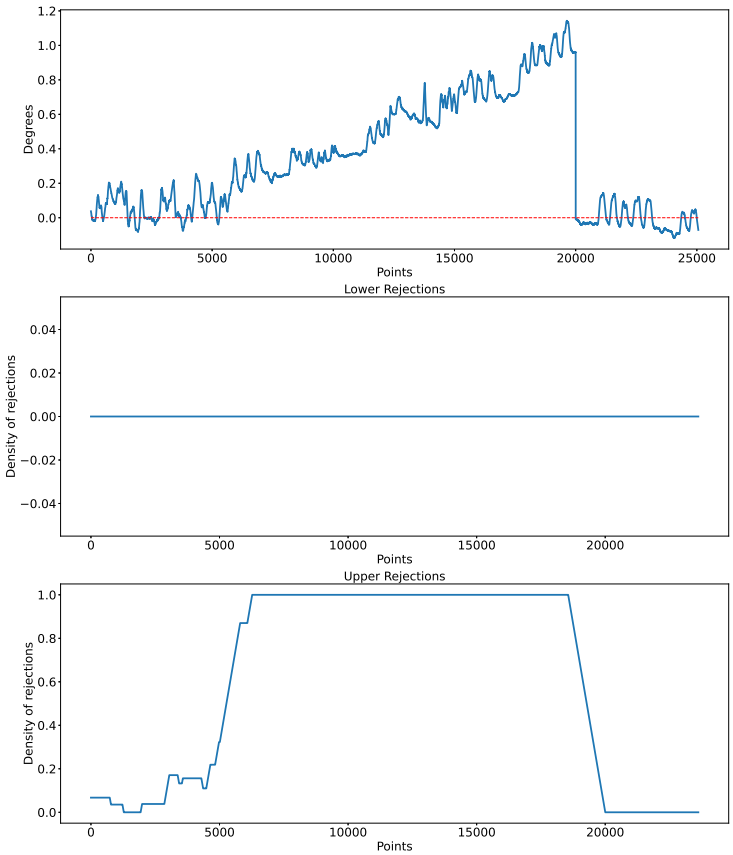

Figure 2.9: Uncalibration detection for new included sensors after transfer learning. The detection of an uncalibration for one of those sensors is shown. This density reaches the maximum possible value from a critical point and during the whole uncalibration phenomenon.

for the adaptation of the system to some other locations.

Regarding the performance of the architecture proposed, uncalibrations can be detected even for values smaller than the defined tolerance ranges

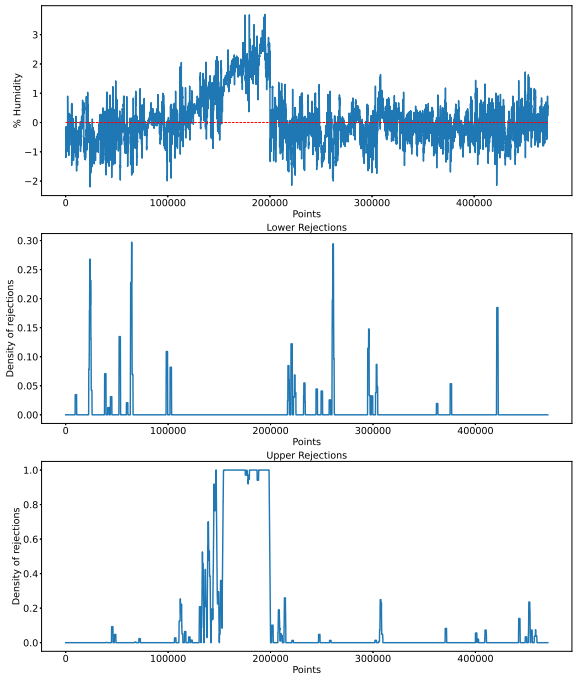

Figure 2.10: Uncalibration detection in the basement. The detection of an uncalibration for one of those sensors is shown. This density reaches the maximum possible value from a critical point and during the whole uncalibration phenomenon.

for most cases in temperature and humidity sensors. A summary of the obtained resolutions and the corresponding tolerances per sensor type are shown in Table 2.5.

Table 2.5: Resolution of the different systems.

Sensor Type	Min	Mean	Max	Tolerance
Temperature	0.10 º	0.26 º	0.64 º	0.5 º
Humidity	0.49 %	1.17 %	3.11 %	3 %
Pressure	0.23 Pa	1.77 Pa	6.68 Pa	0.5 Pa

The application of transfer learning shows the flexibility and scalability of the architecture, enabling the use of the model in a wide variety of contexts such as sensor addition, integration within new environments and partially solving problems associated with the lack of information available.

2.5 Discussion

It is proven that transfer learning retrieved good results for very low amounts of data. The minimum amount of required data is about 10000 samples, which is the data obtained from one week of observations at a sampling rate of one sample per minute. The fact that only the last weight matrix was needed to re-train means that the information associated to all these modifications were learned on a very abstract level in the NN [16]. This suggest, on the one hand, that offsets have a low impact on the joint dynamics of sensors. Thus, the joint behavior of the whole set of sensors is ruled by much deeper relationships than the addition of specific values. On the other hand, the results obtained for pressure sensors, despite they are not under a common constant condition, may be explained by this fact. The sudden changes in pressure values could be understood as an offset, what implies that the model has a deep knowledge of the sensor behavior, although these offsets reduce the accuracy of the detection of uncalibrations.

The architecture presented has been trained and tested in a real context,

with data coming directly from a production pharma factory. Besides, due to the flexible capabilities associated to sensor additions and re-training, the system can be seen as a resilient system as defined in [45]. Furthermore, the architecture presented in this work is currently deployed in Azure [12], where the system performs the uncalibration detection directly from the information of the sensors from a digital twin of the building. This digital twin contains the current status of the physical sensors.

2.6 Conclusions and Future Works

In this work, a system with the capability for detecting uncalibrations in real time has been presented. This system was able to detect all the presented uncalibration events (100% accuracy). In most cases, these uncalibration events could be detected before the specified tolerances were exceeded. Furthermore, this solution can be easily retrained to be adapted to a variety of different scenarios, such as new environmental conditions, the integration of new devices in the sensor network and the deployment in new places never seen before by the system. This adaptability is achieved by means of Transfer Learning.

At the moment of the publication of this contribution, this is the first time that potential uncalibrations of a set of sensors are online detected whenever the set point is unknown. Furthermore, the proposed architecture can be extended by means of transfer learning to a wide range of different fields.

Regarding the future work, different objectives are considered: (1) extending the results obtained after the application of transfer learning. Training the architecture with a generic set of sensors of one type (such as temperature sensors) and then, testing the performance on sensor sys-

tems of a different kind, for instance, engine, smart-city and power plants sensors systems, among others; (2) obtaining information about the learning of the solution through the application of Explainable Artificial Intelligence (XAI) [18]. This could provide useful information about the potential scalability and flexibility of the solution and the different models to which it could be applied.

Chapter 3

Methods for Biologically Inspired Sensors

In Chapter 2 of this PhD Thesis, some of the techniques belonging to the set of classical tools for data analysis were introduced in a real application context. In the present Chapter, the second contribution of this PhD Thesis will be presented, namely, the innovative usage in the context of Machine Learning of mathematical tools coming from some other domains and the development of a completely new technique for analysing the information contained in the spiking data generated by biologically-inspired sensors. All these techniques were, as in the case of Chapter 2, successfully tested in a real production environment, in the facilities of a company of the Aerospace sector.

3.1 Introduction

The context of application of this contribution is the extraction of mechanical information by means of biologically-inspired procedures. More precisely, the idea is to improve the procedure of measuring the thickness of a material stack by using an artificial cochlea [21], [13]. The current procedure is conducted by a strain gauge which is able to provide an accuracy of the order of micrometres but it requires, approximately, 3.5 seconds per measuring point. Reducing this time could significantly improve the efficiency of the overall process. In the case of this contribution, it has been developed a completely new approach based on the experimental artificial cochlea, which is a biologically-inspired sensor. The data generated by this sensor is evaluated by a brand new ML model. Due to the innovative nature of the approach, no previous dataset was available, so it had to be generated and collected during the development of the whole system.

The data generated by the artificial cochlea is a series of pulses distributed in different channels. Each channel is associated to a range of frequencies [33]. Therefore, a pulse in a specific channel means that the perceived sound contains frequency components that lie within the associated frequency interval corresponding to that channel. Besides, the pulsing rate is associated to the intensity of the emitted sound. The sequence of pulses generated by the sensor in a specific temporal frame is called a cochleogram.

Figure 3.1 shows an example of cochleogram associated to a drilling event.

The generated dataset consisted in the spiking pulses associated to, approximately, seven hundred drills collected in production, together with ninety drills collected at the calibration area in the final implementation

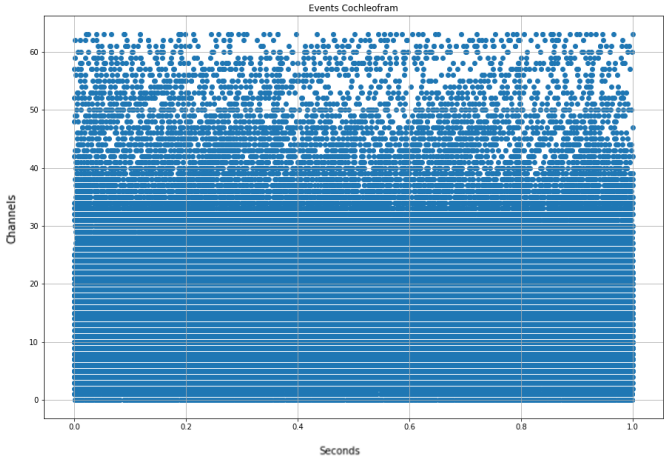

Figure 3.1: Raw cochleogram representation when drilling action is performed.Y represents the channels associated to frequencies and X axis represents the time. Each blue spot stands for a spiking event (a pulse) in that instant.

facilities. Also, ten extra drills were collected at the laboratories in the University of Cádiz for making a preliminary analysis of the capabilities of the sensor. Each of these drills has associated five seconds of measurements. It is important to note that this sensor is asynchronous, and moreover, the sampling rate is not constant. Indeed, the number of pulses per channel depends on the intensity of the sound. Thus, the sampling rate vary between few thousands of samples per second to two hundred thousand samples per second. Therefore, the dataset containing the raw data (i.e. the non-processed data) was conformed by, approximately by 456000000 pulses. It is important to note that not only the nature of the data makes it difficult

to find out the existence of patterns, but also, the massive amount of generated data increases the difficulty of the understanding of the existence of contained information. As it was aforementioned, the development of a metric for the identification of hidden patterns in this kind of data is one of the main contributions of this PhD Thesis.

3.2 Analysis Methods

In this subsection, it is presented the different metrics and techniques used for both, processing the data as well as identifying hidden patterns on them.

3.2.1 Methods for Biologically Inspired Sensors

As commented in [44], sensor development is evolving towards sensor designs strongly based on nature, which beyond higher sensitive capabilities also provide some other advantages like lower consumption rates. However, as it has been mentioned in the introduction to this chapter, the shape and nature of the data generated by this kind of sensors is hard to understand. Next subsection introduces one of the most relevant contributions in this PhD Thesis, the development of a completely new metric that is able to provide information about the existence of underlying patterns in data of spiking nature. This metric is called Pulsing Histogram Energy.

Pulsing Histogram Energy

Pulsing Histogram Energy takes as starting point the cochleograms showed in Figure 3.1. Then, for a fixed time window, the histogram associated to the ocurrences of pulses spiked along the different channels are computed.

Formally:

Definition

Let $n \in \mathbb{N}$ and let $t0, T \in \mathbb{R}^+$. Next, consider the sequence of pulses $P_{t0} = \{p_i^{c_j}\}_{i \in \mathbb{N}}$, where the upper index $c_j \in \{1, 2, \cdots n\}$ denotes the index of the channel where the pulse is generated and the lower index i is the index of the pulse, with every pulse $p_i^{c_j} \in P_{t0}$ taking place in the time interval $(t_0, t_0 + T)$. The histogram $H(P_{t0}) \in \mathbb{R}^n$ associated to the sequence P_{t0} is a vector $(u_1, u_2, \cdots, u_n)$ given by,

$$u_k = \sum_{p_i^{c_k} \in P_{t0}} 1, \ k \in \{1, 2, \cdots n\}.$$

Intuitively, this histogram represents the contributions of the different frequency components to a sound for a fixed-length time window and, therefore, this histogram can be understood as the frequency spectrum provided by the artificial cochlea and associated to the incoming audio signal.

An example of this kind of histograms for 64 channels can be seen in Figures 3.2 and 3.3, whose raw signal was collected at the laboratories in the University of Cádiz.

Figure 3.2 is associated to a train of pulses where the drill-bit was rotating but was not drilling any surface. The histogram exhibits how the spikes are smoothly spread and a well defined bell shape with its peak at, approximately, channel 20 (2.3KHz) can be seen. Besides, it can be observed that spikes are less frequent in higher channels.

Figure 3.3 is associated to a train of pulses where the drill-bit was drilling a surface. It can be noted how the peak is displaced towards channels 12-13 (4.14KHz). Again, as observed in the corresponding cochleogram,

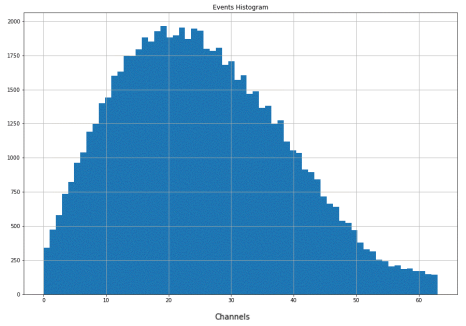

Figure 3.2: Histogram representation when no drilling action is performed. Y axis represents the number of events and X axis represents the channels. The histogram exhibits how the spikes are smoothly spread and a well defined bell shape with its peak at, approximately, channel 20 (2.3KHz) can be seen. As it was observed in the corresponding cochleogram, spikes are less frequent in higher channels.

events at higher channels are less frequent.

Hence, differences in the behaviour of histograms seem to capture changes in the incoming source signal associated to changes in the phenomena that is generating the signal, but still, underlying patterns seem hard to grasp. Next theorem establishes a relation between the frequency domain and the time domain that will allow for the identification of interesting patterns.

Parseval's Theorem

Let $M \in \mathbb{N}$ and consider the sequence $x = \{x_n\}_{n=0}^{M} \subset \mathbb{R}$. Let $X =$

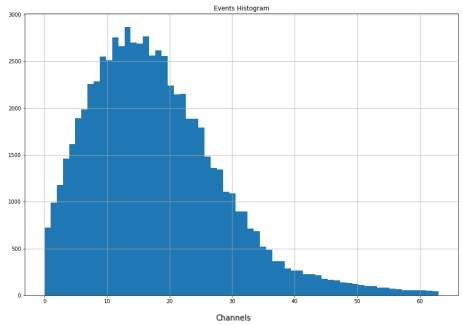

Figure 3.3: Histogram representation when drilling action is performed.Y axis represents the number of events and X axis represents the channels. In this case it is observed how the peak is displaced towards channels 12-13 (4.14KHz). Again, as observed in the corresponding cochleogram, events at higher channels are less frequent.

$\mathcal{F}(x) = \{X_n\}_{n=0}^{M}$ be the Discrete Fourier Transform of x. Then,

$$\sum_{n=0}^{M} x_n^2 = \frac{1}{M} \sum_{n=0}^{M} X_n^2.$$

Mathematically, this result means that the Fourier Transform is an unitary operator. However, in the context of Engineering, this result means that the square of the Frobenious norm of a finite discrete signal is the same that the square of the Frobenious norm of its Fourier transform (except by a scaling factor). Equivalently, the energy of a signal in time domain is the same that the energy of that same signal in frequency domain.

The Frobenious norm is usually interpreted in this context as the energy of a signal, hence, the previous observation can be used for defining a metric that computes the energy associated to the cochleogram in time

domain from the data collected in frequency domain.

Definition

Let $n \in \mathbb{N}$ and let $t0, T \in \mathbb{R}^+$. Next, consider the sequence of pulses $P_{t0} = \{p_i^{c_j}\}_{i \in \mathbb{N}}$, where the upper index $c_j \in \{1, 2, \cdots n\}$ denotes the index of the channel where the pulse is generated and the lower index i is the index of the pulse, with every pulse $p_i^{c_j}$ taking place within the time interval $(t_0, t_0 + T)$. Let $H(P_{t0}) = (u_1, u_2, \cdots, u_N)$ be the associated histogram. The Pulsing Histogram Energy is defined as the square of the Frobenious norm of the histogram,

$$E\big(H(P_{t0})\big) = \sum_{i=1}^{N} u_i^2.$$

Once this metric has been defined, it is time for its application to the data.

Application of the Pulsing Histogram Energy

In this contribution, the PHE was used in two different occasions, namely, as a preliminary capability test of the method at the laboratories in University of Cádiz and in the calibration area of the production line. In both cases, interesting patterns were uncovered by the method. In the case of the preliminary tests in the University of Cádiz, PHE was applied on the whole set of consecutive cochleograms collected during these tests.

As it can be seen, ten equally spread patterns can be seen in Figure 3.4. This fact shows that there exists significant differences among cochleograms associated to different working regimes of the target device. It is important

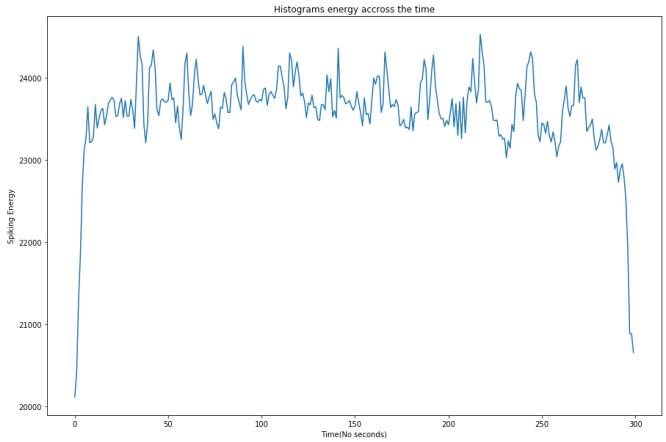

Figure 3.4: Histogram Energy along time. Ten equally distributed patterns can be seen. These patterns, defined by two consecutive peaks, are the patterns associated to the ten performed drills.

to note that, although these differences cannot be detected by the bare eye, it does can be detected by PHE

Moreover, it is important to remind that channels are associated to different frequency bands. Thus, it is reasobale to think that not all the sets of channels will react in the same way to the same stimulus. Therefore, instead of computing the energy for the whole histogram, PHE can be applied to blocks of channels. This allows to find out if energy behaves uniformly along the whole set of channels or it has specific component behaviour.

In Figure 3.5 energy has been computed for blocks of 8 channels. It can be seen that there exist different behaviours for the different blocks of channels. The observed behaviour provide some hints about the reactivity of the different blocks of channels to the different stimulus, with increasing

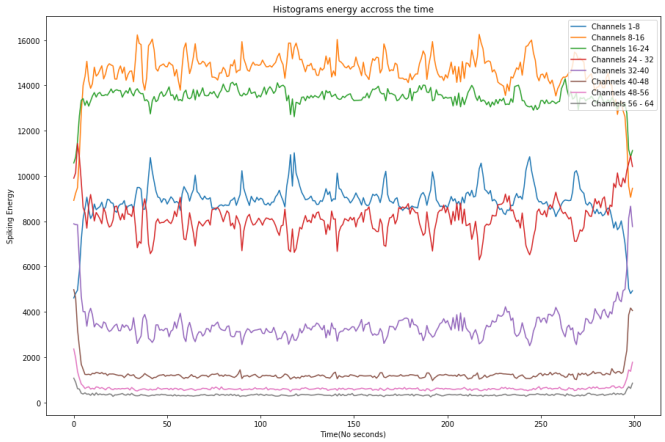

Figure 3.5: Histograms Partial Energy along time. The energy of the histograms has been computed on clusters of channels as specified in the legend.

or decreasing peaks in energy. It can be seen how some blocks of channels are more reactive (positively or negatively reactive) to the drilling events than others. Indeed, channels (8-16), (1-8) and (24-32) are more reactive. Some others are not reactive at all, for instance, (40-48), (48-56) and (56-64).

After the test at University of Cádiz were conducted, some tests in the production line were conducted. These tests were done to check how the artificial cochlea measurements and its capability were affected by the environmental industrial noise. Thus, a set of drills was conducted in the industrial facility with the target robotic arm. More precisely, ninety drills were performed in the calibration area on fiber and aluminium stacks with three different tools. During these tests, the acquisition system was syn-

chronized with the robot control system so that the drilling events could be properly matched with the associated patterns in the incoming signal returned by the artificial cochlea.

In figure 3.6, the PHE associated to a series of drilling events can be seen. Indeed, periodic patterns associated to different subprocesses of the operation are visible. Some of these patterns are associated to processes that are not of the interest of this study, for instance, auxiliary processes devoted to obtaining some other variables of the process. The whole set of patterns is repeated each thirteen seconds and, thanks to the synchronization with the control system, it can be identify which pattern (the red square) is associated with the pure drilling process, i.e, the event where the drill bit is working on the surface. A more detailed view of the pattern of interest can be seen in Figure 3.7.

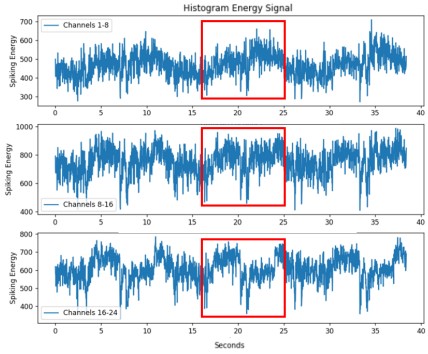

Figure 3.6: Histogram Energy Diagrams across time for a series of drills. Periodic patterns (red squared) can be observed in the different subplots. The periodicity of the patterns is coherent with the periodicity of drilling events.

The application of the PHE showed that the artificial cochlea is capable

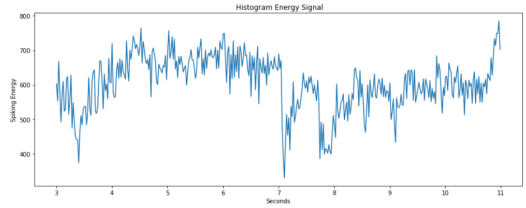

Figure 3.7: Histogram Energy Diagrams across time for a pure drilling event. The pattern associated to the drill bit working on the surface can be seen.

of properly detecting the drilling events although the observed patterns are not similar to those observed during the tests at the laboratory in the University. This mismatch is due to the existence of environmental audio sources that introduce noise into the collected signal and also the mechanical differences coming from the structure of the robotic arm.

After the introduction of PHE, whose purpose is finding out patterns in the data generated by the artificial cochlea, next section describes some other methods that, in this case, are devoted to reduce the data dimensionality.

3.2.2 Processing Methods

In this section, some other methods related to the processing of the data are introduced. In this case, the processing of the data is devoted to face one of the main issues when working with data and, more specifically, with the dimensionality of the space where the data is embedded. This phe-

nomena is called *the curse of dimensionality* [5], [2], [43], and it is specially important when Machine Learning models are going to be used. The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. In the specific case of ML, these phenomena affect to the efficiency of the learning of the model itself, which may not properly learn the patterns of interest or even learn nothing at all. Appart from this fact, also the performance in terms of time response and computational resources consumption is affected.

Due to the massive sampling rate exhibited by the artificial cochlea, it is necessary to reduce the dimensionality of the datapoints that constitute the dataset in order to ease the learning task of the ML model as well as reducing the temporal cost of the inference process, which would enable the use in production.

As it will be detailed later on this subsection, the nature of the cochleograms is very sparse, i.e., cochleograms are mostly made out of zeros with some not null values. This is due to the fact that although the maximum sampling rate of the sensor is very high (of the order of hundreds of KiloHertzs), it is relatively small when compared to the frequency of the clock of the Field-Programmable Gate Array (FPGA) (of the order of tens of Mega-Hertzs) where the Cochlea is deployed. The clock of this FPGA is the one that manages the generation of pulses and defines the maximum temporal resolution of the cochleogram.

For this reason, it has been decided to work on a technique specially devoted to compress data when it is of sparse nature. Moreover, this technique allows for complete recovering of the original signal under certain conditions (what is called lossless compression) so, whenever these condi-

tions hold, it can be ensured that all the information is preserved and, therefore, compressed data can be perfectly used for training the model, since it will keep being recognisable. This technique is called Compressed Sensing [14], [9].

Compressed Sensing

Next, some key ideas, definitions and main results of how CS works are going to be introduced. In case that a deeper level of detail is desired, the paper where the technique is exposed by the very first time can be found in [9].

Now, let $x_0 \in \mathbb{R}^m$ be an unknown *sparse* signal. That is, a signal whose support $T = \{t : \ x_0(t) \neq 0\}$ has a small cardinality with respect the value m.

Next, assume that this signal is wanted to be recovered from a series of observations made through a set of linear measurements of the form,

$$y_k = \langle x_0, a_k \rangle \text{ with } k \in \{1, 2, \cdots, n\}, \tag{3.1}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product and $a_k \in \mathbb{R}^m$ are known tests signals and $n << m$, that is, there is much more unknowns that observations. At first glance, this may seem impossible. However, this is indeed possible under certain conditions.

First, it is important to observe that equation 3.1 can be rewritten in the following way,

$$y = Ax_0, \tag{3.2}$$

where $y \in \mathbb{R}^n$ and A is a $n \times m$ matrix where the $k-th$ row is given by the corresponding a_k with $k \in \{1, 2, \cdots, n\}$. Then, x_0 can be exactly recovered by solving the following minimization problem using the l_1 norm,

$$\min ||x||_{l_1} \text{ subject to } Ax = y, \tag{3.3}$$

provided that the matrix A obeys the *Uniform Uncertainty Principle* [11], [10]. This principle states that the operator represented by the matrix A obeys a *Restricted Isometry Hypothesis*.

Definition

Let A be the $n \times m$ matrix associated to the observations $a_k \in \mathbb{R}^m$ with $k \in \{1, 2, \cdots, n\}$. For every integer $1 < S < m$, the S-Restricted Isometry constant δ_S is defined as the smallest quantity such that,

$$(1 - \delta_S)||c||_{l_2}^2 \leq ||A_I c||_{l_2}^2 \leq (1 + \delta_S)||c||_{l_2}^2, \tag{3.4}$$

where $I \subset \{1, 2, \cdots, n\}$ is an index with $|I| < S$ and A_I denotes the submatrix which is obtained by extracting the columns corresponding to indices in I from the matrix A. In this case, A is said to follow the S-Restricted Isometry Hypothesis.

Intuitively, this definition means that the projection of the operator along a subspace behaves *almost* as a isometry, that is, it preserves the distances (and angles in case that the norm in both, origin and target spaces, is generated by a scalar product).

Once the minimum necessary concepts of CS has been introduced, a way for finding such matrices (operators) is necessary. As exposed in [9], finding these matrices can be done through different approaches (proofs and

additional discussion can be found in [10]). Due to its facility in generation, in this PhD Thesis it has been decided to use randomly generated matrices with Independent and Identically Distributed (i.i.d) entries. Indeed, in [10] is shown that a matrix whose entries are generated by a Gaussian distribution with zero mean and variance $\frac{1}{n}$ satisfies the required conditions with an overwhelming probability whenever S is *small enough*.

Next, the application of the compressed sensing is presented.

Compressed Sensing application

As previously exposed, the application of CS is motivated by the need of reducing the dimensionality of the matrix associated to cochleograms, since they are too big.

In order to ease and increase the efficiency of the computation, a relatively small (50×100) matrix $\mathcal{M}$ has been defined for this case.

The compression is applied channel-wise, this is, the same matrix applies the compresion on every single channel, hence, the yielded compressed cochleogram will also have the same number of channels than the non-projected one.

Besides, the projections are applied by dividing the original cochleogram $\mathcal{C}$ in chunks $C_{j \in J}$ of size (100×64), where J is an index that depends on the cochleogram. Thus, for each $j \in J$, the projected cochleogram $\overline{C}_j$ is defined by

$$\overline{C}_j = \mathcal{M} \cdot C_j.$$

In Figures 3.8 and 3.9 it can be seen the projected cochleograms associated to a void signal (i.e. a fragment of signal where no drilling process was taking place) and a spinning signal, respectively. It is important to note that, in both cases, there was ambient noise. Besides, another rele-

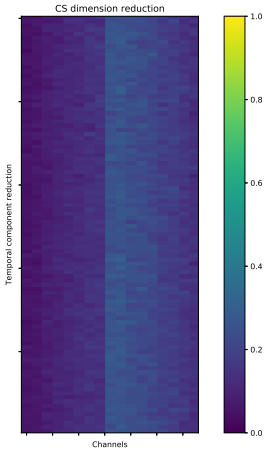

Figure 3.8: Result of the application of CS to a cochleogram coming from a void signal. X axis represents the specific channels. Y axis has no meaning since it is the compressed dimension.

vant aspect to be taken into account is that, due to the clock resolution of the FPGA as well as the communication protocols between the FPGA and the work station where the signal was processed, the size of the raw cochleograms was not constant. This imply that the division of the raw cochleogram in fix-size chunks is not exact and, therefore, there is a reminder. In this cases, the projected cochleograms were completed with zero values until the target size was reached.

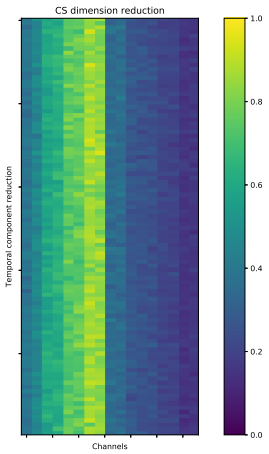

Figure 3.9: Result of the application of CS to a cochleogram coming from a spiking signal. X axis represents the specific channels. Y axis has no meaning since it comes from the compression of the time dimension.

3.3 Methodology

In this section, the architecture of the implemented system, which is capable of properly measuring the thickness of a material stack, is presented. The whole system has been developed using Python. Different libraries such as numpy, scipy and pandas have been used for the implementation of the PHE and the CS procedures. For the implementation and training of the model, TensorFlow2 [1] together with the Keras [23] functional API have been used.

A block diagram of the whole setup can be seen in Figure 3.10.

The main contribution of this chapter is not only based on the using

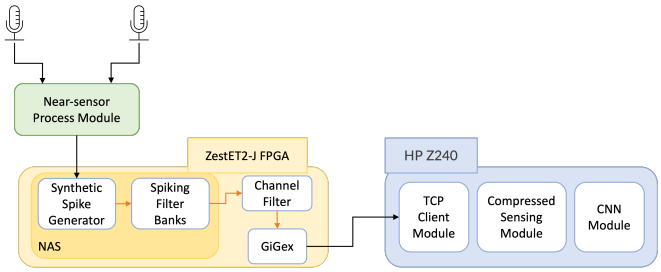

Figure 3.10: Block diagram of the proposed architecture. The digital microphones collect the audio signal which is sent to the NAS sensor in order to be processed and converted to frequency domain. Then, the raw Cochleogram is sent to HP Z240 and projected into a lower dimensional space by means of Compressed Sensing. Finally, the projected Cochleogram is fed into the NN and a value for thickness is provided.

of CS for reducing the size of cochloegrams while preserving the information, but also in the creation of a new ML technique for measuring the thickness of the material stack. The architecture of this ML model is based on an Convolutional Autoencoder together with an attention mechanism [41] which, during the training phase, learns how the thickness of a material stack on a specific drilling point relates to specific patterns hidden in the cochleograms whose dimenssion has been reduced through CS. Then, in the inference stage, the Network is able to predict the thickness of the material stack with a high level of accuracy. The way data is processed is summarized in the following steps:

1. Audio signal is collected by NAS, which is deployed in the FPGA.

2. The signal is streamed in fix-time intervals to HP Z240 workstation, which integrates a Nvidia Quadro K2200 as Graphics Processing Unit (GPU) device, where the data is processed and the inference run.

3. The raw cochleogram collected for this fix-length time interval is then split into fixe size chunks and projected by using CS.

4. The projected cochleogram is then fed into the Neural Network and a thickness measure is obtained.

3.3.1 Neural Network

The measurement of the thickness is made using the architecture shown in Figure 3.11. This measure is computed by a convolutional autoencoder endowed with an attention mechanism applied to the refined input signal.

The idea behind this architecture is that the autoencoder block learn how to encode and decode the input projected cochleograms. Besides, since this architecture is build upon 1D CNN, the learnt patterns will be independent on the time variable (since each channel of the NAS is associated to a channel of the CNN). Finally, the attention mechanism, which is connected to the output of the encoder, automatically learns where to focus in order to efficiently compute the target thickness. The architecture of the Convolutional and Transpose Convolutional Blocks is summarized in Table 3.1.

All the 64 channels provided by the NAS have been included in this approach although it is already known that not all of the channels are reactive to the phenomenon of interests, as it is explained in subsection 3.2, where the PHE is presented and the analysis conducted.

3.4 Results

This section shows the results obtained by the architecture proposed on this Chapter.

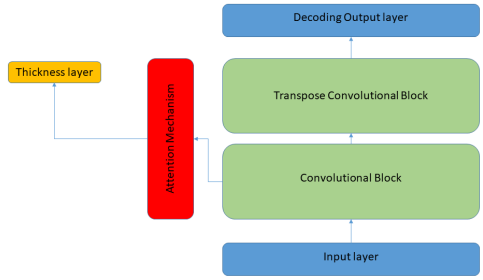

Figure 3.11: Architecture of the Convolutional Autoencoder with Attention Mechanism [41]. The main block is a Convolutional Autoencoder which integrates an encoding convolutional block and a decoding transpose convolutional block. The Convolutional Autoencoder block is unsupervised and learns an efficient representation of the cochleograms in a lower dimension space (the hidden space after the encoding stage). The attention mechanism is connected at the end of the encoding convolutional block. This mechanism learns what features of the compressed representation of the cochleogram are relevant for the computation of the thickness. From the attention mechanism, a Dense layer computes the thickness.

3.4.1 Performance Evaluation Experiments

As it has been previously mentioned, several tests were conducted in order to analyze and check the behaviour of the system at different integration levels: a basic capability test, an environmental integration test and a performance test. Basic capability test was done to check that the NAS was able to detect the acoustic patterns associated to the process under study. This test was conducted in the laboratories of the University of Cádiz, where a much more controlled environment was available. The results of

Layer Type	Filters	Kernel Size
Convolutional	8	7
Convolutional	32	5
Convolutional	32	3
Convolutional	32	3
Transpose Convolutional	32	3
Transpose Convolutional	32	3
Transpose Convolutional	32	5
Transpose Convolutional	64	7

Table 3.1: Convolutional and Transpose Convolutional specifications. This table contains the specifications regarding the number of filters and kernel sizes of the convolutional autoencoder block.

these experiments were introduced during the application of the methods described in 3.2.1. Then, the environmental integration tests were conducted to analyse how the sounds coming from other sources inside the final deployment location could affect the pattern detection. These tests were conducted on test pieces located in the calibration area of the production line within the industrial facility, ant their results were also presented in the applications commented in 3.2.1.

Finally, the performance tests were done to check the accuracy level that the system was able to achieve. For this experiment, the system required some prior training using a dataset coming from the real process. It is important to note that these tests took place on a real production process, i.e., a real industry facility.

Performance Test

These tests were done to check the behaviour of the system in the final production line: a real industrial environment of an important company of

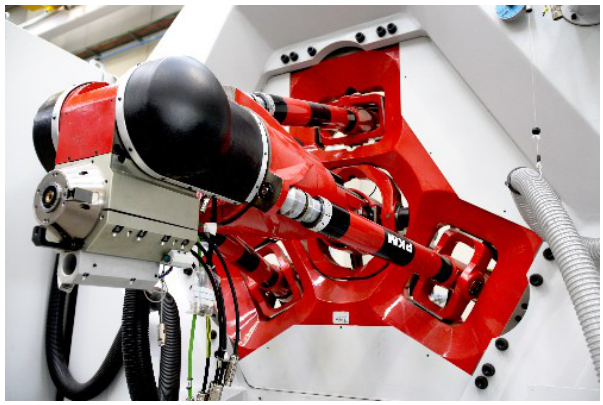

Figure 3.12: Tricepts is a five axis hybrid parallel kinematic machine. It can be used for dynamic applications or as CNC machining center. In this project, the Tricepts performs different tasks related to drilling processes such as drilling, riveting countersinking as well as measuring procedures.

the Aerospace sector. From these tests, the final results presented in this Chapter were obtained. The main differences with respect to the environmental integration tests are that these tests were conducted with signals coming from drilling events on the manufactured pieces and also that these tests were conducted considering a bigger dataset. Besides, this dataset was the one used for training the model described in section 3.3.1. The collected dataset was made out of seven hundred drilling events, all of them coming from the same drilling tool. The dataset was divided in: six hundred drilling events for training the model, fifty for the validation and the remaining fifty drills for testing. As it can be seen in the results section 3.4, the trained model exhibit good accuracy levels on the testing set, which suggests that good information levels can be found on the data collected for this testing stage.

3.4.2 Obtained Results

The results shown are obtained during a drilling operation in a real industrial production facility. The model described in 3.3.1 has been trained with the signals associated to six hundred drills performed by an unique tool during several previous production windows. Furthermore, the real thickness associated to these drills was used in order to train the thickness output layer, which can be seen in the model architecture presented in Figure 3.11. For this training, twenty five epochs were enough in order to achieve the convergence of the model as well as avoiding over-fitting. The obtained results were computed on a sample of one hundred drills coming from a posterior production window. Both, the training and testing datasets come from a population of drills whose mean thickness is of 7.26mm. The evaluation of the model on the testing dataset yields a mean error of 0.75 mm, with a standard deviation of 0.54mm, and maximum and minimum error values of 1.80mm and 0.031mm respectively.

The average time required by the system for performing the thickness measuring task is 0.025 seconds with a standard deviation of 0.005 seconds. The mechanical system that is currently conducting this task requires between 3 and 4 seconds in order to accomplish the same task. These performance results have been obtained by using tensorflow 2.7 with CUDA 11.2 on a Nvidia Quadro K2200.

It is important to note that, even though the obtained accuracy might seem relatively high with respect the mean thickness, the used dataset is relatively small and moreover, not all the possible thikness values were equally sampled (since not all parts in the processed elements are equally frequent). The obtained results suggest that a more balanced and representative dataset will endow the model with the need versatility for properly

infer the thickness of these less frequent points. Another important aspect to be taken into account is the overwhelming reduction in processing time that has been achieved.

3.5 Discussion

One of the main questions that could arise is that the proposed architecture could provide better results if standard vibro-acoustic sensors would be used instead of using the biologically-inspired ones. It can be said that this is still an open issue. It is important to note that we are comparing the standard technology, which is more mature, with biologically inspired sensors, whose definition and working principles are currently under development and whose nature is harder to understand. This is why the obtained results also depends on how this information is treated and processed. Thus, in the short term, it is probable that the obtained results can be equaled or even overcome by using standard sensors. However, the results obtained in this contribution along with the ones shown in [13] prove that the way that biological architectures store information as well as the potential amount of information that can be gathered [22] is bigger. Therefore, it is reasonable to think that, in the medium term, the performance will be better and hardly equaled by standard sensors.

3.6 Conclusions and Future Works

As it has been detailed, the proposed system achieves high level of accuracy with a minimum time requirement. The obtained results show that it is possible to develop biologicall-inspired systems with the capability for measuring and estimating process parameters just by ”hearing” or, in a more

general context, gathering the necessary information in a way similar to that of biological organisms. In this case, the accuracy obtained compared to the required time for running the inference, totally overcomes the existing method. Furthermore, these results suggest that the current accuracy of the system can be increased by using datasets with a higher number of drill instances and with a more heterogeneous representation of drills with different associated thickness. Regarding the current limitations and drawbacks, it is important to remark that these kind of methods usually requires big amounts of data in order to properly learn the target features. Besides, it is also required a preliminary analysis of the information contained in the gathered signals in order to determine if this biologically-inspired approach is suitable or not for a specific target problem. Despite these circumstances, it is believed that in a near future, the integration of biological approaches into the industrial world will enable the increment of the efficiency in both, auxiliary and main processes. Thus, in the future, it is planned to extend the current methodology both, to some other biologically-inspired sensors such as artificial retinas, and also keep on developing all the auxiliary techniques shown in subsections 3.2.1 and 3.2.2, that have been used for determining the existence of relevant information contained in the in these kind of sensors.

Chapter 4

Methods for Classification Tasks on Structured Datasets

In the two previous Chapters, the reader made a journey through the use of different data processing techniques in the context of ML in order to obtain insights of the data that were going to be used for training the models. In Chapter 2, the presented techniques belonged to the ordinary set of tools of a Data Scientist. Chapter 3 took one step further by introducing mathematical tools coming from some other scientific contexts, non related *a priori* with ML. Also, a completely new technique for extracting information of spiking data generated by Biologically-Inspired sensors was presented. The present Chapter contains the last contribution of this PhD Thesis. In this case, it is presented a completely brand new set of tools for measuring the information contained in a dataset for classification tasks.

4.1 Introduction

The context of application of this contribution is the analysis of the suitability of a dataset for training a ML model in a classification task on structured data.

In recent years, ML has experienced a huge development, which has allowed that different tasks, previously reserved to humans, can now be performed by algorithms [8], [36] with a high level of success. During this period, scientist have focused on the architecture of the algorithms, developing different modules that process the data in different ways in order to conceptualize and structure the information contained in the dataset [41], [29]. However, the invested efforts in order to determine the quality of the dataset and how good it is for the target task is much smaller. As it has been seen in Chapter 2 and 3, there exists some techniques and methods, such as PCA [30], that allow to infer some aspects of how the data behaves in the dataset but, in practice, experience plays a major role when it comes to acquiring and dealing with a dataset. A good dataset contains information, thus, it is fair to ask if this information can be measured somehow by means of the use of the existing theory.

In this Chapter, it is presented two new metrics (Empirical Overlpaing Measure and Montecarlo based Measure) for measuring the complexity in a classification task based on the overlaping of the classes. Also, a set of Python functions have been developed for easing the computation of the metrics. Finally, the capabilities of these metrics are tested with experiments on two open-access datasets (Iris Dataset and Mobile Price Classification Dataset) by analysing the performance of two general purpose classifiers like a Multi-Layer Perceptron [16] and XGBoost [38]. The obtained results show that both metrics are able to properly predict the performance

of the classifiers, although pathological cases exist for the case of the Empirical Overlaping Measure that are successfully handled by the Montecarlo based Measure.

4.2 Analysis Methods

In the current section, it is presented the theoretical foundations of the main contributions presented within this chapter, this is, different tools for measuring the quality of a dataset for being used in classification tasks on structured data.

Therefore, it is important to clarify some preliminary concepts.

Definition
For a given set V, let $v \in V$. The element v is said to represent a structured data if, for a given context, each component has a defined and fixed interpretation. Otherwise the vector is said to represent an unstructured data.

An example of structured data is element $v \in \{0, 1\} \times \mathbb{R}$ whose second component stand for the sex and the age of a person. Conversely, an example of unstructured data is a digital photo, where each single pixel represents different semantic concepts even within the same context.

Definition
A classification task is a task that consists in the definition of a function that assigns items in a collection to target categories or classes.

Intuitively, in order to properly map each item to its corresponding class, these items must exhibit attributes and features that allows to differentiate each one from the others. In this context, let I be a set of items, let $\mathbb{R}$ be the real numbers and let V be a real n-dimensional vector space.

Definition

A *representation* is defined for the set of items I as a function

$$\rho : I \longrightarrow V.$$

Thus, a representation for a set of items I is a function that associates a vector to each item $i \in I$. This vector may represent quantitative values associated to different features of this item (when dealing with structured data) or, maybe, each component has not a specific meaning (when dealing with unstructured data). It will be said that they are structured or unstructured representations in each case, respectively.

It is important to note that the definition of a suitable representation is a tough task. Moreover, the success of a classification task will depend on the definition of the representation. Let us propose an example. Let us consider the set of items I integrated by all the individuals that belong to the specie Iris Oratoria. On the one hand, we will define a first representation given by

$$\rho_1 : I \longrightarrow \mathbb{R}^2$$

This representation associates to each individual its total length (thorax plus abdomen) and its abdomen width. On the other hand, we will define a second representation given by

$$\rho_2 : I \longrightarrow \mathbb{R}^3$$

This representation associates to each individual the mean vector of all its rgb values (red, green and blue). For instance, we take a picture of the individual and we compute the mean red, green and blue. If it is defined a classification task in order to classify each individual as *male* or *female*, it will be seen that both of them (males and females) are mostly green, therefore, a color-based representation is not good for this classification task. However, females are larger with a mean length of 8 cm in contrast with the mean length of the males of 6.5 cm. For this task, the length of the individuals and the width of its abdomen is a more suitable representation.

Although this is clear for this particular example, it is not always easy to define a representation. Thus, a natural question arises, how good is the chosen representation?.

According to [39], the complexity of the image (image of a function) of a representation is low if the image of the reference class by the representation is *disjoint* from the images of the other classes, if the image of the reference class by the representation is not ambiguous and if the defined decision boundaries are not complex. More specifically, if the images of two classes by the representation are not disjoint, then there could exist items belonging to different classes whose images are very similar. This will make these items difficult or impossible to properly classify. Therefore, in order to define the aforementioned measure, measuring the intersection among different categories inside a dataset will be the target point. For this purpose, it will be needed some auxiliary metrics, which are going to be described in the following sections.

4.2.1 Spread Measure

In this subsection, it is provided the definition of a measure for the spread of the items belonging to a specific class for a specific representation. This measure can be used for understanding how data points are distributed around the center of mass of one class belonging to a dataset, which makes it suitable for being used as a dispersion measure of general purpose.

Given a set of items $I_C \subset I$ belonging to an unique class C, consider a representation

$$\rho : I \longrightarrow V$$

where V is an arbitrary n-dimensional vector space over the reals endowed with a norm $||\cdot||$. Let κ_C denote the center of mass of $\rho(I_C)$. Next, consider the set of distances $\Delta^{\rho}_{\kappa_C}$ defined by

$$\Delta^{\rho}_{\kappa_C} = \big\{||\rho(i) - \kappa_C|| : \forall i \in I_C\big\}.$$

For a natural number $m > 1$, let $Q_m(\Delta^{\rho}_{\kappa_C})$ be the set of m-quantiles of the set of distances $\Delta^{\rho}_{\kappa_C}$. In order to illustrate the notion behind $Q_m(\Delta^{\rho}_{\kappa_C})$, consider the next example.

Example Provided a representation ρ and the center of mass κ_C of the representation of a class C, assume that the set of distances $\Delta^{\rho}_{\kappa_C}$ is:

$$\Delta^{\rho}_{\kappa_C} = \{0.068, 0.262, 0.282, 0.36, 0.404, 0.408, 0.444,$$

$$0.446, 0.548, 0.570\}$$

Then, for m = 4, the set $Q_4(\Delta^{\rho}_{\kappa_C}) = \{0.303, 0.406, 0.446\}$, represent the set of quartiles of the previously mentioned set of distances. In figure 4.1, a graphic representation of the previously descripted example can be seen.

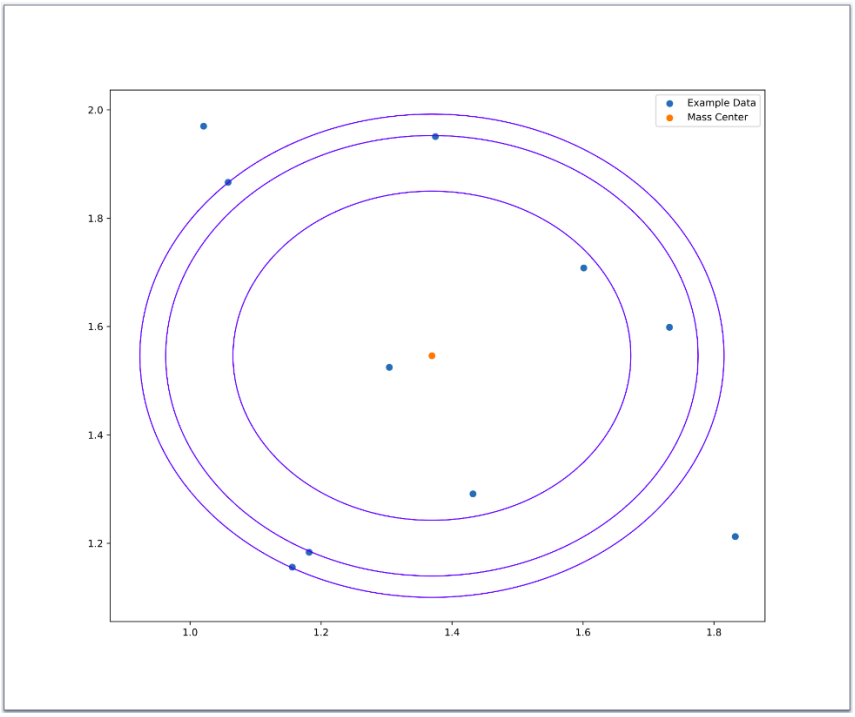

Figure 4.1: In this figure, a graphic representation of the example previously descripted can be seen. The data has been generated by means of a bidimensional uniform distribution which has been displace along the vector (1,1). The radius of the blue circles are the values of the quartiles depicted in $Q_4(\Delta^{\rho}_{\kappa_C})$.

Now, from the previous definition of $Q_m(\Delta^{\rho}_{\kappa_C})$, include the minimum and the maximum of the set of distances to the set of m-quantiles:

$$\overline{Q_m(\Delta^{\rho}_{\kappa_C})} = Q_m(\Delta^{\rho}_{\kappa_C}) \cup \left\{ \min \Delta^{\rho}_{\kappa_C}, \max \Delta^{\rho}_{\kappa_C} \right\}.$$

We will define a special notation for the minimum and the maximum,

namely, $q_0 = \min \Delta^{\rho}_{\kappa_C}$ and $q_m = \max \Delta^{\rho}_{\kappa_C}$. Besides, q_i will stand for the i-th element in the m-quantile set.

It is important to note that, for datasets with a huge amount of samples in high-dimensional vector spaces the behaviour of the distances become fuzzy [2]. Thus, instead of basing the approach in a quantitative computation of distances and geometric properties, an approach based on the measurement of the uncertainty associated to the probability of finding data points of the dataset at specific distances is provided. For this approach, the concept of the differential entropy is necessary.

Definition Let X be a random variable with probability density function f whose support is a set $\mathcal{X}$. The differential entropy $h(X)$ is defined as

$$h(X) = -\int_{\mathcal{X}} f(x)\log(f(x))dx,$$

where log denotes the natural logarithm.

Thus, from the previous definition, a density function is required in order to compute this entropy. We will define a probability density function which represents the density of elements at different distances from the centre of mass of the set $\rho(I_C)$. More specifically, this density function is given by

$$f_{\rho(I_C)}(x) = \begin{cases} 0 & x < q_0 \\ \frac{1}{m\cdot(q_i - q_{i-1})} & q_{i-1} \leq x \leq q_i \\ 0 & q_m < x, \end{cases} \quad (4.1)$$

with $1 \leq i \leq m$. Note that $f_{\rho(I_C)}(x)$ is, indeed, a probability density function. It is easy to see that $f_{\rho(I_C)}(x)$ is Lebesgue Integrable, and

$f_{\rho(I_C)}(x) \geq 0$ for all $x \in \mathbb{R}$. Moreover

$$\begin{aligned}\int_{-\infty}^{\infty} f_{\rho(I_C)}(x)dx &= \int_{q_0}^{q_m} f_{\rho(I_C)}(x)dx \\ &= \sum_{i=1}^{m} \int_{q_{i-1}}^{q_i} \frac{1}{m \cdot (q_i - q_{i-1})} dx \\ &= \sum_{i=1}^{m} \left[\frac{x}{m \cdot (q_i - q_{i-1})} \right]_{q_{i-1}}^{q_i} \\ &= \sum_{i=1}^{m} \frac{1}{m} = 1\end{aligned} \tag{4.2}$$

Now, the definition of differential entropy can be applied to the previously defined density function in order to measure the uncertainty of finding data points at specific distances.

Definition

Let $I_C \subset I$ be a set of items belonging to an unique class C. Let the pair $(V, ||\cdot||)$ be a normed vector space and consider the representation, $\rho : I_C \longrightarrow V$. We define the spread entropy as

$$h(\rho(I_C)) = -\int_{-\infty}^{\infty} f_{\rho(I_C)}(x) \log(f_{\rho(I_C)}(x))dx,$$

where log denotes the natural logarithm and $f_{\rho(I_C)}$ stands for the density function associated to the distances to the centre of mass for the image of the class I_C by the representation ρ.

Example Now, it will be shown an example that illustrates how the previously defined spread entropy works. More precisely, this example is intended to show how the way in which a representation spreads along the vector space is captured by the spread measure, and moreover, how higher

levels of entropy are associated to representations where the represented data are not as precisely located as in other cases. Also, negative values can be taken by this measure (since it is a measure of entropy).

Therefore, let I_C be a specific class within a set of items I. Let ρ_1, ρ_2 and ρ_3 be representations verifying

$$\rho_k : I_C \longrightarrow \mathbb{R}^2,$$

with $k \in \{1, 2, 3\}$. Now, suppose that the behaviour of $\rho_k(I_C) \subset \mathbb{R}^2$ can be described by three different random variables. More precisely,

$$\begin{aligned} \rho_1(I_C) &\sim E(\lambda) + l_1 \\ \rho_2(I_C) &\sim N(\mu, \Sigma) - l_1 \\ \rho_3(I_C) &\sim U(a, b) - l_2 \end{aligned} \tag{4.3}$$

where E, N and U stand for an Exponential, Normal and Uniform random variables. In order to have a suitable statistical representativeness, one thousand samples have been generated for each random variable. The values assigned to the parameters in the example represented in Figure 4.2 can be seen in the caption of the figure.

For those distributions whose elements spreads along the space with higher density levels, the maximum distance where a significantly high number of elements can be found will be bigger. This can be seen through the associated spread density function. For this example, it has been computed the sets $\overline{Q_{100}(\Delta_{\kappa_C}^{\rho_k})}$, and the graphical representation of the associated density functions $f_{\rho_k(I_C)}$ can be seen in figure 4.3. Please note that, for this example, percentiles have been used in the computation of $\overline{Q_m(\Delta_{\kappa_C}^{\rho})}$. The

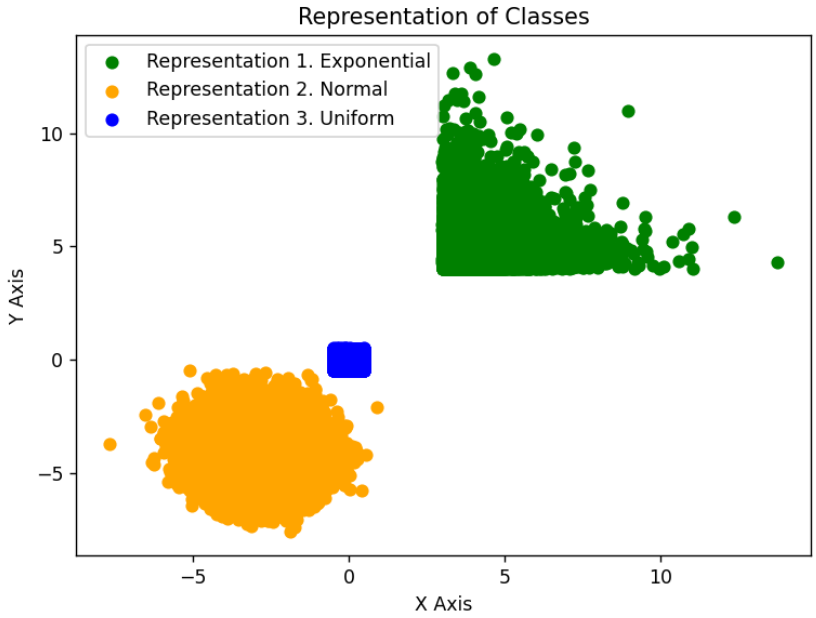

Figure 4.2: Examples of the representations ρ_1, ρ_2, ρ_3. The specific values of the parameters are $\lambda = 1$, $\mu = 0$, $\Sigma = 1$, $a = (0, 0)$, $b = (1, 1)$, $l_1 = (3, 4)$ and $l_2 = -\left(\frac{1}{2}, \frac{1}{2}\right)$

motivation for using percentiles is that, due to the number of samples generated by the random variables, using percentiles will provide both, a good resolution in spatial terms and a fairly high statistical representativeness.

As it can be seen, this entropy measure *punishes* those representations whose images contain elements that can be found with a fairly high probability at a wider range of distances. These ranges of distances can be understood as the concreteness of the representation. The less specific the

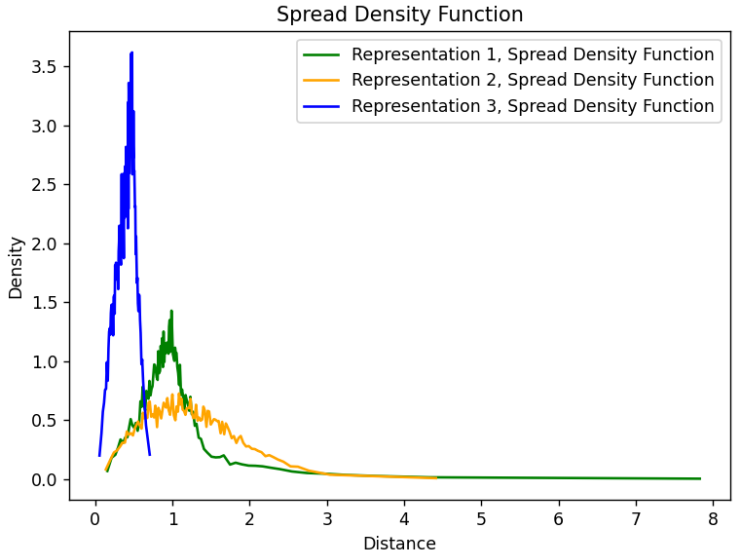

Figure 4.3: Spread density functions associated to the representations ρ_1,ρ_2 and ρ_3 . The spread entropies for the three representations are $h(\rho_1(I_C)) = 0.799$, $h(\rho_2(I_C)) = 0.94$ and $h(\rho_3(I_C)) = -0.57$.

representation, the higher the entropy.

4.2.2 Empirical Overlaping Measure

In the present subsection, some of the concepts defined in subsection 4.2.1 are going to be used for measuring the intersection of different classes. Later, this way of computing the intersection can be used for measuring how confusing a dataset is for a classification task.

It is important to note that datasets are of discrete nature. Thus, due to its *non-continuous* nature, a direct measurement of the intersection of

the representation of two classes of a dataset makes no sense, since it would probably be void even in the case that the support (i.e. those points where the associated distribution function is not null) of the statistical distributions from which the samples are drawn had no void intersection. However, if the statistical distributions that rule the behaviour of the representation of the classes were known, it could be possible to provide a measure of the intersection or overlaping of the support of the associated distribution functions but, in practice, the analytic expression of these distribution functions is unknown. From this point, a natural question arises: Is there any way for approaching the underlying distribution function taking the information provided by the available samples?. In Measure Theory there is a way for defining an empirical distribution function based on the observations provided by the available samples. The important point is that this empirical distribution function converges to the analytical distribution function.

Therefore, let us take a concept from [40] that will allow us to compute Empirical Probability Measures.

Definition

Let (S, Ω) be a measurable space and let $\{x_1, \cdots, x_l\} \subset S$ be a sample of elements drawn from a random variable X on S. We define the empirical distribution function associated to X and the sample $\{x_1, \cdots, x_l\} \subset S$ as a function $\mu_{\{x_1,\cdots,x_l\}}$ such that for each $A \in \Omega$:

$$\mu_{\{x_1,\cdots,x_l\}}(A) = \frac{\xi_A(\{x_1, \cdots, x_l\})}{l},$$

where ξ_A stands for the counter function of set A, i.e.

$$\xi_A(\{x_1, \cdots, x_l\}) = \#(\{x_1, \cdots, x_l\} \cap A).$$

The previous definition allows us to define a measure for the intersection of two classes,

Definition

For a class of items $I_C \subset I$, a representation $\rho : I \longrightarrow V$ and a measurable set $A \subset V$, we can define an associated empirical distribution function $\mu_{\rho(I_C)}$ given by

$$\mu_{\rho(I_C)}(A) = \frac{\xi_A(\rho(I_C))}{\#(I_C)},$$

where the symbol # stands for the cardinal of a set.

It is important to note that, in general,

$$\mu_{\rho(I_{C1})}(A) \neq \mu_{\rho(I_{C2})}(A),$$

for a measurable set $A \subset V$ and two classes I_{C1}, I_{C2}.

For the specific case on which we are working, we are going to work on the measurable spaces of the form $(\mathbb{R}^n, \Omega)$ with Ω the Borel $\sigma-$algebra, which can be generated by compact sets. Indeed, for the next definition, we are going to consider the closed balls centered in arbitrary points, which are measurable sets for this $\sigma-$algebra.

Definition

Let V be a n-dimensional real vector space and let $c \in V$ and $r \geq 0$. Let us denote by $B(c, r)$ the n-ball of centre c and radius r. Finally, consider a representation $\rho : I \longrightarrow V$ for a set of items I. For any two classes $I_{Ci}, I_{Cj} \subset I$ we define the overlaping measure of I_{Ci} with respect I_{Cj} as

$$\mu_{\rho(I_{Ci})}\big(B(\kappa_{Cj}, q_m^j)\big),$$

where q_m^j denotes the element $q_m \in \overline{Q_m(\Delta_{\kappa_C}^{\rho})}$ for the class I_{Cj}. For sake of simplicity, we will denote $\mu_{\rho(I_{Ci})}\big(B(\kappa_{Cj}, q_m^j)\big)$ as μ_{ij} in order to simplify the notation.

There exists several important properties to note:

1. $\mu_{ij} = 0$ whenever the number of samples of each class is big enough, means that the images of the classes I_{Ci}, I_{Cj} by means of the representation ρ are disjoint.

2. $\mu_{ij} = 1$ implies that $\rho(I_{C_i}) \subset \rho(I_{Cj})$.

3. In general, $\mu_{ij} \neq \mu_{ji}$.

The agnostic class differentiation measure defined in Section 4.2.4 endowed with this measure will be tested in Section 4.3.1. During the experiments it will be seen pathological cases where the geometric configuration of the dataset may yield a lack of predictive capability of the measure presented in this section.

4.2.3 Montecarlo Based Measure

In this section, another way for computing a measure of the intersection between two classes is presented. In this case, the method is based on the selection of elements randomly distributed within the class. More precisely, provided a representation ρ, two classes I_{Ci} and I_{Cj} and a random sample extracted from $\rho(I_{Ci})$, it will be counted how many points of this random sample are very close to a point in $\rho(I_{Cj})$. First, the notion of close need to be formalized:

Let $(V, ||.||)$ be a n-dimensional real normed vector space and consider a structured representation $\rho : I \longrightarrow V$ for a set of items I. Next, for any

two classes $I_{Ci}, I_{Cj} \subset I$, it is wanted to compute a quantity that stands for the measure of the intersection of these two sets. For this, it will be taken a subset $\{x_1, x_2, \cdots x_M\}$ of random elements in $\rho(I_{Ci})$ and see how many of them are close to some element that belongs to $\rho(I_{Cj})$. Hence, for every $x_m \in \{x_1, x_2, \cdots x_M\}$ define the quantity r_m as

$$r_m = \min\{||x_m - \rho(i)|| \text{ such that } i \in I_{Ci} \text{ and } x_m \neq \rho(i)\}.$$

This value stands for the distance from the data point x_m to its closest data point in set $\rho(I_{Ci})$. Now define the average distance among elements as r, which is computed as

$$r = \sum_m \frac{r_m}{M}.$$

This average distance among elements will allow us to have a radius inside which we can look for elements coming from other class in the dataset.

Next, it is necessary to define the occurrences of balls in $\rho(I_{Ci})$ that contains points in $\rho(I_{Cj})$. Formally,

Definition

Let $(V, ||.||)$ be a n-dimensional real normed vector space and consider a structured representation $\rho : I \longrightarrow V$ for a set of items I. For any two classes $I_{Ci}, I_{Cj} \subset I$ we define the Montecarlo-based intersection measure as

$$\mu_{ij} = \sum_m \frac{\max \chi_{B(x_m,r)}(\rho(I_{Cj}))}{M}, \tag{4.4}$$

where $x_m \in \{x_1, x_2, \cdots x_M\} \subset \rho(I_{Ci})$ is a random sample within the target class and r stands for the average distance among elements in $\rho(I_{Ci})$.

Please note that for a set $B \subset A$, the function $\chi_B : A \longrightarrow \{0, 1\}$ stands

for the characteristic function, which is defined for all $x \in A$ as $\chi_B(x) = 1$ if $x \in B$ and $\chi_B(x) = 0$ if $x \in A \setminus B$.

There exists several important properties to note:

1. Note that $\mu_{ij} = 0$ whenever the number of samples of each class is big enough, implies that $\rho(I_{Ci}) \cap \rho(I_{Cj}) = \emptyset$.

2. Also, note that $\mu_{ij} = 1$ can be interpreted as $\rho(I_{C_i}) \subset \rho(I_{Cj})$.

Next subsection will be focused on the use of the intersection measures defined in the present and previous section in order to determine the goodness of a dataset for a classification task.

4.2.4 Class Differentiation Measure

Once it has been defined different measures for understanding how the image of the items of a class are distributed by the action of a representation ρ and, moreover, the intersection of any two classes can be computed, it is time to focus on the definition of a measure devoted to evaluate how well the images by ρ of a set of classes $\{I_{C1}, I_{C2}, \cdots I_{Cm}\}$ can be differentiated. This will allow us to provide a measure of the quality of the dataset for a classification task. With this target in mind, next properties are needed to be satisfied for this function:

1. This function must take into account the average intersection values for all existing classes.

2. This function must take the value 0 in case of total disjointedness among classes.

3. This function must take higher values whenever the overlaping rate increases.

4. This function must take the infinity value whenever there exists a class whose representation is completely included within some other class.

Now, the next definition is provided:

Definition

Let $I = \cup_i^k I_{C_i}$ be a set of items with k classes, let V be a n-dimensional real vector space and let $\rho : I \longrightarrow V$ be a structured representation. For a intersection measure μ and a class $I_{Ci} \subset I$, define its associated Class Differentiation Measure as

$$\phi_i = \sum_{j \neq i} \frac{-\log\left(1 - \mu_{ij}\right)}{k-1} \tag{4.5}$$

Please note that in case that all the classes representations are disjoint the previous function yields 0. Also, note that the previously defined function increases if the values μ_{ij} increase. Also, note that in the case that a class $\rho(I_{Ci})$ is contained in $\rho(I_{Cj})$ then $1 - \mu_{ij} = 0$, and thus the logarithm is not defined. In this case, we define ϕ_i to be equal to infinity. Now, we can compute the Mean Class Differentiation Measure as

$$\phi = \sum_{i=1}^{n} \frac{\phi_i}{n} \tag{4.6}$$

Let us give an example that will clarify the behaviour of the Class Differentiation Measure.

Example

As an example, consider a set of items I made out of three classes and

a representation ρ that projects I on $\mathbb{R}^2$. Suppose that we are on the conditions presented in Figure 4.4.

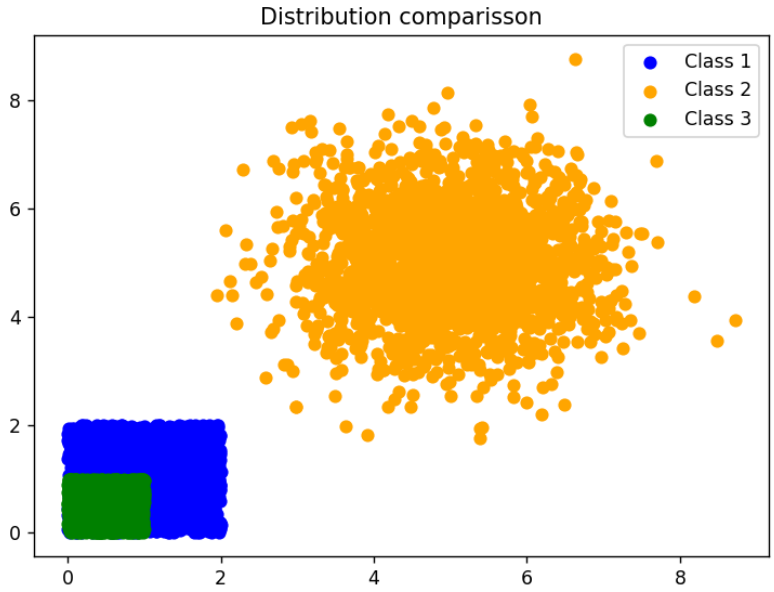

Figure 4.4: Images associated to three classes by means of representation ρ.

As it can be seen, class 2 can be easily differentiated from the others. However, class 3 is completely contained within class 1. Items in class 2 could be easily classify, however, a data point belonging to $\rho(I_C1) \cap \rho(I_{C3})$ is not classifiable. By computing the Class Differentiation Measure and using the Empirical Overlaping Measure defined in subsection 4.2.2, we obtain that $\phi_1 = 0.194$, $\phi_2 = 0$ and $\phi_3 = \infty$.

Thus, this definition provides a tool that can be used for measuring how

different a set of classes is under a certain representation.

4.3 Results

This section contains the description of the obtained results for the previously defined measures. These results have been obtained by testing the measures by means of experiments. More precisely, two experiments have been conducted in order to test the predictive capability of the Class Differentiation Measure as a evaluator for the suitability of the provided dataset for classification tasks. These experiments consists on the implementation of a series of steps on two different datasets. These are two public datasets (Iris Dataset and Mobile Price Classification Dataset) on which a classification task is going to be run. For each experiment, both, the Empirical Overlaping Measure (EOM) and Montecarlo Based Measure (MBM) for computing the intersection between classes have been used. This has yielded some pathological cases where the EOM defined in subsection 4.2.2 is not properly working. However, in these pathological scenarios, MBM defined in subsection 4.2.3 has returned satisfactory results.

The two experiments follow the same structure:

1. First, the Control dataset is imported and normalized and divided on training and testing. No modifications on the original structure of this dataset are going to be implemented on this step.

2. Next, we compute the Class Differentiation Measure of the control dataset. The measure is computed by using the two available intersection measures, i.e. EOM and MBM.

3. Then, in order to test the performance of the Class Differentiation Measure on models of different nature we use two of the most extended

and used models, namely, a classifier MLP and a Classifier XGB. These two models are trained and tested on the same dataset.

4. The models are scored on the testing dataset.

5. Now, we check that the performance of the models after the scoring step is coherent with the output of the Class Differentiation Measure. In this first part, models should have obtained fairly good results and the obtained Class Differentiation Measure should be not too high and, more specifically, not ∞.

6. Next, the structure of the control dataset is altered in two different ways. After each type of alteration, steps from 1 to 5 are repeated. First kind of alteration consists in making a random assignation of labels to the elements in the training dataset. Second kind of alteration consists in separately applying the standard scaling (i.e. substracting the mean and divide by the standard deviation) to each single class. It is important to note that these alterations on the control datasets actually are modifications of the representation that was initially chosen for representing the reference datasets. After applying any of these two alterations the Class DifferentiationMeasure values are expected to reach ∞ and the performance of the two models is expected to be poor.

4.3.1 Obtained Results

In this subsection, the obtained results will be presented through the results of the experiments depicted in the introduction of Section 4.3.

Experiment on Iris Dataset

This is a well known dataset usually used for learning how to proceed on classification tasks. The Iris Species can be classified within three categories, namely, Iris-virginica, Iris-versicolor and Iris-stosa and four are the features to be taken into account for doing that classification, namely, SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm. Table 4.1 summarizes the results of this experiment.

Dataset	ϕ E-O Meas.	ϕ M-B Meas.	MLP	XGB
CD	0.33	0.119	0.933	0.977
RAL	∞	∞	0.044	0.288
CSS	∞	∞	0.288	0.222

Table 4.1: In this table it can be seen the output value of the Class Differentiation Measure and the scoring value of the MLP and XGB for all the CD and the two modifications conducted on the Iris dataset. In this table, M-B stands for the ϕ values computed by the Montecarlo-based measure, R A L stands for Random Assignation of Label and, in the same way, C S S stands for Categorical Standard Scalation.

As it can be seen in equation 4.5, the Mean Class Differentiation Measure takes into account on its computation the intersection measures of the different categories, whose expresion can be found in equation 4.4. Thus, associated to it, there are different matrices with the intersection measures for the different categories. This gives us an idea of how the different representations of the categories intersect among them, which is a good measure fo how distinghishable they are. The values in next tables have been computed using the MBM.

Table 4.2 shows the intersection measures for the case of the Control

Dataset of the Experiment conducted on Iris Dataset.

Categories	C1	C2	C3
C1	-	0	0.428
C2	0	-	0
C3	0.142	0	-

Table 4.2: Matrix of intersection measures for the Control Dataset in Iris Dataset Experiment.

Analogously, in table 4.3 the matrix of intersection measures for the Random label assignation can be seen. In this case it can be observed a much higher values of intersection are obtained.

Categories	C1	C2	C3
C1	-	1	1
C2	0.833	-	0.833
C3	1	1	-

Table 4.3: Matrix of intersection measures for the Random Assignation of Labels in Iris Dataset Experiment.

Experiment on Mobile Price Classification Dataset

In this case, the experiment is conducted on the Mobile Price Classification Dataset. This is a public dataset where different models of mobile phones are categorized in different categories associated to different prices based on different features such as the clock speed, the battery power etc.

Table 4.4 summarizes the results of the experiment.

As in the previous experiment, we show the matrix of intersection measures for the Control Dataset case and the Random Assignation of Labels.

Dataset	ϕ E-O Meas.	ϕ M-B Meas.	MLP	XGB
CD	∞	0.135	0.913	0.91
RAL	∞	∞	0.185	0.253
CSS	∞	∞	0.241	0.265

Table 4.4: In this table it can be seen the output value of the Class Differentiation Measure and the scoring value of the MLP and XGB for all the control dataset and the two modifications conducted on the Mobile Price Dataset. In this table, M-B stands for the ϕ values computed by the Montecarlo-based measure, R A L stands for Random Assignation of Label and, in the same way, C S S stands for Categorical Standard Scalation.

The values in next tables have been computed using the Montecarlo based measure. In table 4.5 it can be seen the values for the Control Dataset.

Categories	C1	C2	C3	C4
C1	-	0	0	0.292
C2	0	-	0.214	0.357
C3	0	0.132	-	0
C4	0.309	0.368	0	-

Table 4.5: Matrix of intersection measures for the Control Dataset in Mobile Price Experiment.

Analogously, in table 4.6 it can be seen the values of intersection measures for the case of Random Assignation of Labels.

Categories	C1	C2	C3	C4
C1	-	0.958	0.958	0.986
C2	1	-	0.957	0.957
C3	0.971	1	-	1
C4	0.956	0.956	1	-

Table 4.6: Matrix of intersection measures for the Random Assignation of Labels in Mobile Price Experiment.

The conducted experiments have shown that, in both cases, the behaviour of the Mean Class Differentiation Measure computed with the MBM measure is coherent with the performance obtained by the different models on the different datasets (control and modified). Moreover, it can also be seen that there exists pathological cases where the EOM is not properly working. For instance, in the experiment conducted on the Mobile Price Classification Dataset, the control dataset was properly understood by the models, however, the Mean Class Differentiation Measure was $\phi = \infty$ for the Empirical Overlaping Measure computation.

This incoherent behaviour of the measure is related to the geometry of the representation of the different classes. In the case that the EOM is used for computing ϕ, and the image of a class $\rho(I_{Ci})$ is bigger, too close, and contained in an affine subspace parallel to the representation of some other class $\rho(I_{Cj})$ this can yield these kind of problems. An example of such situation can be seen in figure 4.5. Thus, it is concluded that the EOM could not be good enough.

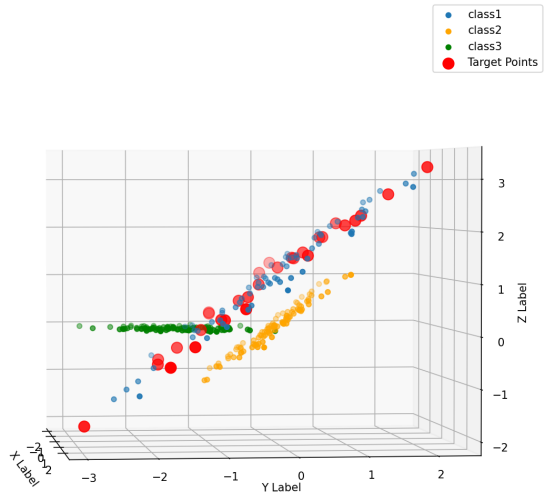

Figure 4.5: This image shows a pathological case where the ϕ value would be ∞ if the measure would have been computed with the Empirical Overlaping Measure. At the same time, it can be seen how the Montecarlo-based Measure solves this problem.

4.4 Conclusions and Future Works

The results obtained in the experiments conducted on section 4.3.1 show that the Class Differentiation Measure not only works as a reliable estima-

tor of the complexity and thus the quality of the dataset for classification tasks, but also the geometry of the chosen representation can be better understood thanks to proper measures defined for measuring the intersections of the projections of classes like, for instance, the MBM measure. In the future, the efforts will be aimed at two different targets: On the one hand, extending the definition of this measure to regression tasks and, on the other hand, applying the current definition to some other classification tasks from the context of NLP like, for instance, sentiment analysis. Also, in the future, it is intended the development of this kind of techniques in order to define strategies with the capability for improving the used representations by means of of the geometric information provided by the measuring models. These strategies might not provide a direct tool but intuition of how the chosen representation works, and how to take advantage of some of its properties. All the developed code for this manuscript can be found on Chapter A in Section A.1.

Appendix A

Appendix: Developed Code

A.1 Functions for Class Differentiation Measure

In the present Appendix, it is shown the code developed for computing the Class Differentiation Measure together with the different required auxiliary processes. All the developed functions make use of the numpy library.

distance_to_point

```
def distance_to_point(point, array):
    """
    This function computes the distance of one single point (with respect the
    ↪  Frobenious norm) to a set of
    (dimensioanlly coherent) array of points.
    Parameters
    ----------
    point: numpy array. Query point
```

```
    array: numpy array. Array of target points. The distance will be computed with
    ↪  respect each point contained in this
    array.

    Returns
    -------
    array of distances.
    """

    dist = []
    for i in range(array.shape[0]):
        d = np.linalg.norm(point - array[i])
        dist.append(d)
    return np.array(dist)

```

centroid_distance

```
def centroid_distance(sample):
    """
    This function computes the distance from the mass center of a given set to each
    ↪  other point

    Parameter
    ---------
    sample: sample of elements on which the method will be applied.

    Return
    ------
    d_v: distance vector
    """
    m = np.mean(sample, axis=0)
    d_v = distance_to_point(m, sample)
    return d_v
```

density_funct_estimation

```python
def density_funct_estimation(sample, density_factor=100):
    """
    This function estimates a density function for the distances with respect the
    ↪  centre of mass of a provided sample.

    Parameters
    ----------
    sample: numpy array. Target sample.
    density_factor: int. Steps or divisions in which the density function is
    ↪  computed. Default: 100

    Returns
    -------
    q: numpy array. x_values of the computed density function. These values are
    ↪  computed as the q-quantiles of the
    distances of the samples with respect the centre of mass. q is determined by
    ↪  the density_factor.
    h: numpy array. y_values of the computed density function. These values are
    ↪  computed so that (q_i - q_(i-1))*h_i =
    1/density_factor.
    """
    q = []
    for i in range(1, density_factor):
        q.append(np.quantile(sample, i / density_factor))
    q = np.array(q)
    q = np.concatenate((np.array([np.min(sample)]), q,
    ↪  np.array([np.max(sample)])))
    dq = np.diff(q)
    h = (1 / density_factor) / dq

    return q[1:], h
```

mutual_density

```
def mutual_density(target_random_sample, non_target_random_sample,
↪  non_target_quantiles, quantile_index = -1):
    """
    This function computes the empirical probability measure of the intersection of
    ↪  two
    categories within a dataset.

    Params
    --------
    target_random_sample: np.array. Samples that belongs to the target category.
    ↪  The
                          probability measure will be computed with respect the
                          ↪  empirical
                          probability distribution associated to this category.
    non_target_random_sample: np.array. Samples that belong to some other category.
    ↪  If
                          these two categories are not disjoint then the
                          ↪  intersection will
                          be a measurable set and its measure can be computed with
                          ↪  respect
                          the different probability measures.
    non_target_quantiles: np.array. Array that contains the quantiles of distances
    ↪  with respect
                          the non_target_random_sample.
    quantile_index: int.  Index of the quantile to be taken as radius from the
    ↪  non_target_quantiles array.

    Return
    -------
    m: float. Measure of the intersection of the two categories with respect the
    ↪  empirical prob
              bability measure defined for the target_random_sample-
    """

```

```
    r = non_target_quantiles[quantile_index]
    c2 = np.mean(non_target_random_sample, axis=0)
    distances_to_c2 = distance_to_point(point=c2, array=target_random_sample)
    categories_intersection = distances_to_c2[np.where(distances_to_c2 <= r)]
    m = categories_intersection.shape[0] / target_random_sample.shape[0]
    return m
```

quantile_entropy

```
def quantile_entropy(density_vector):
    """
    This function computes the entropy associated to a simple density function made
    ↪ out of constant values
    obtained for equally spaciated quantiles.

    Parameters
    ----------
    density_vector: array. Vector containing the densities obtained for a specific
    ↪ sample.

    Return
    ------
    qe: Scalar value associated to the entropy of the sample.

    """
    c = 1/density_vector.shape[0]
    qe = (-c)*np.sum(np.log(density_vector))
    return qe

```

mutual_density_divergence_eom

```
def mutual_density_divergence_eom(target_category, category_list):
    """
    This function computes a coefficient associated to the measure of
    the intersections of a measurable set with respect someother measurable sets
    for a set of probability measures. This coefficient intends to provide insights
    of the average 'size'of these intersections and, in that way, how
    ↪  distinguishable
    is a set from others.

    Parameters
    ----------
    target_category: array. Dataset associated to the target category for which the
    ↪  coe-
                    fficient will be computed.
    category_list: list of arrays. This list must contain all datasets associated
    ↪  to the
              different categories  in a classification problem.
    mode: int. Used mean method. Default:0 Arithmetic mean. 1, geometric mean.

    Return
    -------
    phi: float. Coefficient associated to the target category.

    """

    relative_measures = []
    for category in category_list:
        d = centroid_distance(category)
        q,h = density_funct_estimation(d)
        m=mutual_density(target_category,category,q)
        relative_measures.append(m)
    relative_measures = np.array(relative_measures)
    phi = np.mean(np.log(1/(1-relative_measures)))
    return phi, relative_measures
```

mutual_density_divergence_mcm

```
def mutual_density_divergence_mcm(target_category, category_list):
"""
    This function computes a coefficient associated to the measure of
    the intersections of a measurable set with respect someother measurable sets
    for a set of probability measures. This coefficient intends to provide insights
    of the average 'size'of these intersections and, in that way, how
    ↪  distinguishable
    is a set from others.

    Parameters
    -----------
    target_category: array. Dataset associated to the target category for which the
    ↪  coe-
                    fficient will be computed.
    category_list: list of arrays. This list must contain all datasets associated
    ↪  to the
              different categories  in a classification problem.
    mode: int. Used mean method. Default:0 Arithmetic mean. 1, geometric mean.

    Return
    -------
    phi: float. Coefficient associated to the target category.

    """
    cpr = 0.2#Control Points Rate
    relative_measures = []
    control_points = target_category[:int(cpr*target_category.shape[0])]
    reference_data = target_category[int(cpr*target_category.shape[0]):]
    mn,s = average_inner_distance(target_points = control_points, data =
    ↪  reference_data)
    for category in category_list:
        m = dataset_intersection_counter(target_reference_points = control_points,
        ↪  radius = mn+2*s, target_dataset=category)
        relative_measures.append(m)
```

```
    relative_measures = np.array(relative_measures)
    phi = np.mean(np.log(1/(1-relative_measures)))
    return phi, relative_measures

```

dataset_density_function_estimation

```
def dataset_density_function_estimation(dataset):
"""
    Computes the quantile entropy associated to a provided dataset.

    Parameters
    ----------
    dataset: numpy array. Target sample.

    Return
    ------
    q: numpy array. Quantile values.
    h: numpy array. Density associated to the quantile values in order to define a
    ↪  density function.
"""
    d = centroid_distance(dataset)
    q,h = density_funct_estimation(d)
    return q,h


```

dataset_quantile_entropy

```
def dataset_quantile_entropy(dataset):

    """
    This function computes  the entropy associated to a dataset.

```

```
    Parameters
    ----------
    dataset: numpy array. Target sample.

    Return
    -------
    e: float. Entropy
    """

    q,h = dataset_density_function_estimation(dataset)
    e = quantile_entropy(h)
    return e
```

mean_categorical_entropy

```
def mean_categorical_entropy(category_list):
    """
    This function computes the mean categorical entropy defined for n categories in
    a classification problem as (phi1*h1+...+phin*hn)/n

    Parameters
    ----------

    category_list:list of arrays. This list must contain all datasets associated to
    ↪ the
              different categories  in a classification problem.

    Return
    --------
    mce:float. Mean Cathegorical Entropy.
    """
    entropies = []
    mdd_coefficients=[]

    for i in  range(category_list.shape[0]):
```

```
        quantile_entropy = dataset_quantile_entropy(category_list[i])
        entropies.append(quantile_entropy)
        target_category = category_list[i]
        non_target_category = np.delete(category_list,i,axis = 0)
        #print(f'shape {non_target_category.shape}')
        phi,_relative_measures = mutual_density_divergence(target_category =
        ↪  target_category,category_list = non_target_category)
        print(f'Relative Measures {_relative_measures}')
        print(f'Coefficients {phi}')
        mdd_coefficients.append(phi)
    entropies = np.array(entropies)
    mdd_coefficients = np.array(mdd_coefficients)
    mce = np.mean(100*np.log2(mdd_coefficients)+entropies)
    return mce
```

category_finder

```
def category_finder(training_dataset,category_column):
"""
This function takes a pandas DataFrame training_dataset as
input and returns a set of pandas DataFrame each associated
to a specific category

Parameters
----------
training_dataset: Pandas DataFrame. DataFrame that contains the
                  training dataset
category_columns: str. Name of the column that contains the
categories

Returns: category_dict. Dictionary that contains the different
        categories dataframes
"""
category_dict = {}
categories = training_dataset[category_column].unique()
```

```
    #print(f'Unique Values: {categories}')
    for i in range(categories.shape[0]):
        #print(f'Categoria: {categories[i]}')
        name = categories[i]
        current_df = training_dataset.loc
        [training_dataset[category_column]==categories[i]]
        .copy()
        category_dict[name]= current_df
    return category_dict

```

mean_mutual_density_divergence

```
def mean_mutual_density_divergence_eom
(dataset,representation_columns,category_column):
    """
    this function computes the mean mutual density divergence.

    Parameters
    ----------
    dataset: pandas DataFrame. dataset that is going to be used for training the
    ↪  model (
            all columns must be included)
    representation_columns: list of str. Names of the columns that stand for the
    ↪  representation,
                        i.e. the feature columns
    category_column: str. Name of the column that contains the categories

    Return
    ------
    mean_phi: float. Mean of the phi values associated to the different mutual
    ↪  density divergence

    phis: list of float. List of phi values associated to the different mutual
    ↪  density divergence
```

```
19
20      """
21
22      categories = category_finder(dataset,category_column)
23      categories_names = list(categories.keys())
24      phis = []
25      for i in range(len(categories_names)):
26          remaining_category_names = categories_names.copy()
27          target_category_name = categories_names[i]
28          remaining_category_names.remove(target_category_name)
29          #print(f'Target Category Name: {target_category_name}')
30          #print(f'Remaining Category Name: {remaining_category_names}')
31          #computing the divergence
32          target_X = categories[target_category_name]
33          [representation_columns].values
34          #print(f'This is target_X: { target_X}')
35          complementary_targets = [categories[name][representation_columns].values
            ↪  for name in remaining_category_names]
36          #print(f'These are complementary targets: {complementary_targets}')
37
38          phi,rm = mutual_density_divergence(target_category=target_X,
            ↪  category_list=complementary_targets)
39          phis.append(phi)
40          #print(f'Mean phi:{np.mean(phis)}\nList of phis: {phis}')
41
42          del remaining_category_names
43      return np.mean(phis),phis
```

mean_mutual_density_divergence_mcm

```
1  def mean_mutual_density_divergence_mcm
2  (dataset,representation_columns,category_column):
3      """
4      this function computes the mean mutual density divergence.
5
```

```
    Parameters
    ----------
    dataset: pandas DataFrame. dataset that is going to be used for training the model (
            all columns must be included)
    representation_columns: list of str. Names of the columns that stand for the representation,
                          i.e. the feature columns
    category_column: str. Name of the column that contains the categories

    Return
    ------
    mean_phi: float. Mean of the phi values associated to the different mutual density divergence

    phis: list of float. List of phi values associated to the different mutual density divergence

    """

    categories = category_finder(dataset,category_column)
    categories_names = list(categories.keys())
    phis = []
    for i in range(len(categories_names)):
        remaining_category_names = categories_names.copy()
        target_category_name = categories_names[i]
        remaining_category_names.remove(target_category_name)
        #print(f'Target Category Name: {target_category_name}')
        #print(f'Remaining Category Name: {remaining_category_names}')
        #computing the divergence
        target_X = categories[target_category_name]
        [representation_columns].values
        #print(f'This is target_X: { target_X}')
        complementary_targets = [categories[name][representation_columns].values for name in remaining_category_names]
        #print(f'These are complementary targets: {complementary_targets}')
```

```
37
38          phi,rm = mutual_density_divergence_2
39          (target_category=target_X, category_list=complementary_targets)
40          phis.append(phi)
41          print(f'Relative measures: {rm}')
42          #print(f'Mean phi:{np.mean(phis)}\nList of phis: {phis}')
43
44          del remaining_category_names
45      return np.mean(phis),phis
46
47
```

average_inner_distance

```
1  def average_inner_distance(target_points, data):
2  """
3      This function compute the average distance from a representation
4      of a set called data to a series of target points.
5
6      Parameters
7      ----------
8      target_points: np.array. Array of target points. These points are
9                     the reference points for computing the average distance.
10     data: np.array. Array associated to the data points for a specific
       ↪  representation.
11
12     Return
13     ------
14     mean_dist: float. Mean distance to the different points
15     std_dist:  float. Standard deviation of the distances.
16     """
17     distances_array = np.zeros(shape=(target_points.shape[0],data.shape[0],
18     target_points.shape[1]))
19     for i in range(target_points.shape[0]):
20         distances_array[i]= target_points[i] - data
```

```
    d = np.linalg.norm(distances_array,axis = 2)
    m_d = np.min(d,axis = 1)
    mean_dist = np.mean(m_d)
    std_dist = np.std(m_d)
    return mean_dist, std_dist

```

dataset_intersection_counter

```
def dataset_intersection_counter(target_reference_points, radius, target_dataset):
    """
    This function counts the number of points of a target dataset inside the
    ↪  covering
    defined by the balls of centers in target_reference_points (coming from a
    ↪  reference dataset)
    with radius radius.
    The idea is to detect if two given datasets the intersectionis void or not.

    Parameters
    ----------
    target_reference_points: np.array. Array of target points. These points come
    ↪  from the targe dataset of
                    reference.
    radius: float. Radius of the balls
    target_dataset: np.array. Array of points associated to someother dataset whose
    ↪  intersection
    with the reference dataset wants to be computed.

    Returns
    -------
    intersection_measure: float. This value is the number of target reference
    ↪  points that have points
                          of target dataset inside a ball of center a target_point
                          ↪  and radius radius.
    """
```

```
    distances_array = np.zeros(shape=(target_reference_points.shape[0],
    target_dataset.shape[0],target_reference_points.shape[1]))
    for i in range(target_reference_points.shape[0]):
        distances_array[i]= target_reference_points[i] - target_dataset
    d = np.linalg.norm(distances_array,axis = 2)
    n_intersection_indexes = np.unique(np.where(d <= radius)[0]).shape[0]
    intersection_measure = n_intersection_indexes/target_reference_points.shape[0]
    return intersection_measure
```

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, Corrado G., A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015, Software available from tensorflow.org.

[2] C. Aggarwal, A. Hinneburg, and D. Keim, *On the surprising behavior of distance metric in high-dimensional space*, First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973) (2002).

[3] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, *Unsupervised real-time anomaly detection for streaming data*, Neurocomputing **262** (2017), 134–147.

[4] N. Alharbi and B. Soh, *Roles and challenges of network sensors in smart cities*, IOP Conference Series: Earth and Environmental Science, vol. 322, IOP Publishing, 2019, p. 012002.

[5] N. Altman and M. Krzywinski, *The curse(s) of dimensionality*, Nature Methods **15** (2018), 399–400.

[6] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A.C. Knoll, *A survey of robotics control based on learning-inspired spiking neural networks.*, Frontiers Neurobotics **12** (2018), no. 35.

[7] S. Bozinovski, *Reminder of the first paper on transfer learning in neural networks, 1976*, Informatica **44** (2020), no. 3.

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, R. Henighan, T.and Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, and D. Amodei, *Language models are few-shot learners*, Advances in Neural Information Processing Systems (H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, eds.), vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[9] E. J. Candes, J. K. Romberg, and T. Tao, *Stable signal recovery from incomplete and inaccurate measurements*, Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences **59** (2006), no. 8, 1207–1223.

[10] E. J. Candes and T. Tao, *Near-optimal signal recovery from random projections: Universal encoding strategies*, IEEE Transactions on Information Theory **52** (2006), no. 12, 5406–5425.

[11] E.J. Candes and T. Tao, *Decoding by linear programming*, IEEE Transactions on Information Theory **51** (2005), no. 12, 4203–4215.

[12] D. et al. Chappell, *Introducing the windows azure platform*, David Chappell & Associates White Paper (2010).

[13] J. P. Dominguez-Morales, A. F. Jimenez-Fernandez, M. J. Dominguez-Morales, and G. Jimenez-Moreno, *Deep neural networks for the recognition and classification of heart murmurs using neuromorphic auditory sensors*, IEEE transactions on biomedical circuits and systems **12** (2017), no. 1, 24–34.

[14] Y. C. Eldar and G. Kutyniok, *Compressed sensing: theory and applications*, Cambridge university press, 2012.

[15] K. Ethayarajh, C. Yejin, and S. Swayamdipta, *Information-theoretic measures of dataset difficulty*, ArXiv (2021).

[16] A. Géron, *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems*, " O'Reilly Media, Inc.", 2019.

[17] H. Ghorbani, *Mahalanobis distance and its application for detecting multivariate outliers*, Facta Univ Ser Math Inform **34** (2019), 583–95.

[18] D. Gunning, *Explainable artificial intelligence (XAI)*, Defense Advanced Research Projects Agency (DARPA), nd Web **2** (2017), no. 2.

[19] J. D. Hamilton, *Time series analysis*, 1 ed., Princeton University Press, 1994.

[20] J. Hawkins and S. Blakeslee, *On intelligence*, Macmillan, 2004.

[21] A. Jiménez-Fernández, E. Cerezuela-Escudero, L. Miró-Amarante, M.J. Domínguez-Morales, F. de Asís Gómez-Rodríguez, A. Linares-Barranco, and G. Jiménez-Moreno, *A binaural neuromorphic auditory sensor for fpga: A spike signal processing approach*, IEEE transactions on neural networks and learning systems **28** (2016), no. 4, 804–818.

[22] N. K. Kasabov, *Methods of Spiking Neural Networks BT - Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*, Springer, 2019.

[23] N. Ketkar, *Introduction to Keras*, Deep learning with Python, Springer, 2017, pp. 97–111.

[24] A. Kolmogorov, *Three approaches to the quantitative definition of information*, Problems of Information Transmission **1** (1965), 4–7.

[25] S. K. Lee, M. Bae, and H. Kim, *Future of iot networks: A survey*, Applied Sciences **7** (2017), no. 10, 1072.

[26] M. Li and P. Vitányi, *An introduction to kolmogorov complexity and its applications*, 01 1997.

[27] P. C. Mahalanobis, *On tests and measures of group divergence i. theoretical formulae.*, Proc. Nat. Inst. Sci. India (Calcutta) (1936), 49–55.

[28] Prasanta Chandra Mahalanobis, *On the generalized distance in statistics*, National Institute of Science of India, 1936.

[29] Chen-K. Mikolov, T., G.s Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, Proceedings of Workshop at ICLR **2013** (2013).

[30] S. Mishra, U. Sarkar, S. Taraphder, S. Datta, D. Swain, R. Saikhom, S. Panda, and M. Laishram, *Principal component analysis*, International Journal of Livestock Research (2017), 1.

[31] L.J. Muñoz-Molina, I. Cazorla-Piñar, J. P. Dominguez-Morales, L. Lafuente, and F. Perez-Peña, *Real-time detection of uncalibrated sensors using neural networks*, Neural Computing and Applications (2022), 1–13.

[32] Encyclopedia of Mathematics, *Mahalanobis distance*, 2001.

[33] Zurich Institute of Neuroinformatics, *Java tools for address-event representation (aer) neuromorphic processing.*, 2007, https://github.com/SensorsINI/jaer [Accessed: 2020-02-15].

[34] S. J. Pan and Q. Yang, *A survey on transfer learning*, IEEE Transactions on knowledge and data engineering **22** (2009), no. 10, 1345–1359.

[35] F. J. Pontes, G.F. Amorim, P. P. Balestrassi, A. P. Paiva, and J. R. Ferreira, *Design of experiments and focused grid search for neural network parameter optimization*, Neurocomputing **186** (2016), 22–34.

[36] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, *Zero-shot text-to-image generation*, Proceedings of the 38th International Conference on Machine Learning (Marina Meila and Tong Zhang, eds.), Proceedings of Machine Learning Research, vol. 139, PMLR, 18–24 Jul 2021, pp. 8821–8831.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788.

[38] Amazon Web Services, *Xgboost*, 2024, https://docs.aws.amazon.com/es$_e$*s/sagemaker/latest/dg/xgboost.html*[*Accessed* : 06/03/2024].

[39] K. Tin and M. Basu, *Complexity measures of supervised classification problems*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 3, 289–300.

[40] V.N. Vapnik and A.Y. Chervonenkis, *On the uniform convergence of relative frequencies of events to their probabilities*, pp. 11–30, 01 2015.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, Proceedings of the 31st International Conference on Neural Information Processing Systems (Red Hook, NY, USA), NIPS'17, Curran Associates Inc., 2017, p. 6000–6010.

[42] S. Walczak and N. Cerpa, *Artificial neural networks*, Encyclopedia of Physical Science and Technology (Third Edition) (Robert A. Meyers, ed.), Academic Press, New York, third edition ed., 2003, pp. 631–645.

[43] J. Wang, *Geometric structure of high-dimensional data and dimensionality reduction*, vol. 13, Springer, 2012.

[44] D. Wicaksono, *Learning from nature: biologically-inspired sensors*, Ph.D. thesis, Delft University of Technology, 2008.

[45] W. J. Zhang and Y. Lin, *On the principle of design of resilient systems–application to enterprise information systems*, Enterprise Information Systems **4** (2010), no. 2, 99–110.