University of Minho
School of Engineering

Simão José Silva Leite

Study and selection of several artificial intelligence algorithms for gas detection in systems with scarce resources

January, 2023

UMinho | 2023

Simão Leite

Study and selection of several artificial intelligence algorithms for gas detection in systems with scarce resources

University of Minho
School of Engineering

Simão José Silva Leite

Study and selection of several artificial intelligence algorithms for gas detection in systems with scarce resources

Master's Dissertation
Master's in Industrial Electronics and Computer Engineering

Work supervised by
Jorge Miguel Nunes Cabral

January, 2023

Determination is nothing without dedication and hard work.

Agradecimentos

Eu gostaria de agradecer ao professor Jorge Cabral por me orientar ao longo da minha dissertação e por me dar a oportunidade de realizar este projeto. Queria também agradecer aos Professores João Carvalho, Rui Machado e Sofia Paiva do DTx que sempre se mostraram disponíveis para me ajudar neste projeto. Um grande abraço a todos os meus professores do departamento de Engenharia Eletrónica da Universidade do Minho que fizeram a diferença no meu percurso de aprendizagem durante estes últimos cinco anos.

Um agradecimento especial a todos os meus amigos que sempre me ajudaram a dar o melhor de mim e que se mostraram disponíveis para me ajudar em tudo que fosse preciso.

À minha família queria agradecer do fundo do meu coração por me apoiarem em todas as minhas decisões, pois sempre me indicaram o melhor caminho a seguir, profissional e pessoal. Um muito obrigado às três mulheres da minha vida (a minha mãe, a minha irmã e a minha madrinha) e um grande abraço ao meu pai, ao meu padrinho e ao meu avô.

Por fim, um obrigado a todos que me ajudaram neste percurso académico.

Project "(Link4S)ustainability - A new generation connectivity system for creation and integration of networks of objects for new sustainability paradigms [POCI-01-0247-FEDER-046122 | LISBOA-01-0247-FEDER-046122]" is financed by the Operational Competitiveness and Internationalization Programmes COMPETE 2020 and LISBOA 2020 under the PORTUGAL 2020 Partnership Agreement, and through the European Structural and Investment Funds in the FEDER component.

Abstract

Study and selection of several artificial intelligence algorithms for gas detection in systems with scarce resources

The main contribution of this thesis is the study and selection of several artificial intelligence algorithms for gas detection developed to be used with BME688 gas sensor, in IoT (Internet-of-Things) end device. The proposed gas monitoring process provides the IoT end devices with gas leak detection, measuring the air quality and ensuring the correct gas concentration for some environments. Nowadays, with the development of the IoT technology, gas monitoring has been integrated into people's lives and the industry. One of the problems in gas monitoring is the lack of cross-sensitivity and low selectivity of the gas sensors. So, as a way to solve these problems, the idea of implementing artificial intelligence algorithms capable of overcoming such issues has arisen. Most artificial intelligence algorithms require a large processing capacity by the systems in which they are running, leading to mass energy consumption. This high energy consumption brings disadvantages, since not all systems have a significant number of resources to handle it. The operation time of a gas monitoring device without requiring maintenance is another crucial aspect. To achieve such a goal, the devices must consume little energy. Therefore, this thesis's goal is the selection of an artificial intelligence algorithm that is best suited for resource-scarce systems.

Keywords: Artificial intelligence algorithms; Resource scarce systems

Resumo

Estudo e seleção de vários algoritmos de inteligência artificial para a deteção de gases em sistemas com recursos escassos

O principal contributo desta dissertação é o estudo e seleção de vários algoritmos de inteligência artificial para a deteção de gases em sistemas com recursos escassos. A monitorização de gases é um processo bastante importante em alguns sistemas, pois permite detetar fugas de gases que possam ocorrer, medir a qualidade do ar e garantir uma correta concentração de gases em determinados sistemas. Atualmente com o desenvolvimento da tecnologia Internet-of-Things (IoT), a monitorização de gases, tem sido integrada na vida das pessoas e na indústria. Um dos problemas que reside na monitorização de gases é a falta de sensibilidade cruzada e a baixa seletividade por parte dos sensores de gases, existindo atualmente em funcionamento. Assim, como forma de resolver estes problemas surgiu a ideia de implementar algoritmos de inteligência artificial que possam colmatar tais problemas. A maioria dos algoritmos de inteligência artificial requerem grande capacidade de processamento por parte dos sistemas nos quais estão inseridos, levando a um grande consumo de energia dos mesmos. Este elevado consumo de energia traz grandes desvantagens, pois nem todos os sistemas possuem um número elevado de recursos. Outro aspeto também importante a ter em consideração é o tempo de operação de um dispositivo de monitorização de gases sem necessitar de manutenção, este objetivo só é atingido se o consumo de energia for baixo. Por conseguinte, o objetivo desta dissertação é selecionar um algoritmo de inteligência artificial que melhor se adequa a sistemas de recursos escassos.

Palavras-chave: Algoritmos de inteligência artificial; Sistemas de recursos escassos

Contents

List of Figures

List of Tables

Listings

Acronyms

AI	Artificial Intelligence
AIC	Akaike Information Criterion
ANN	Artificial Neural Network
ASIC	Application-specific integrated circuit
AUC	Area Under Curve
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
GAM	Generalized Additive Models
GUI	Graphical User Interface
HITRAN	High-Resolution Transmission Molecular Absorption Database
IoT	Internet-of-Things
LP	Low Power
MEMS	Microelectromechanical systems
ML	Machine Learning
MOS	Metal Oxide Semiconductors
RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristic
SMBus	System Management Bus

SoC	System on a Chip
SVM	Suport Vector Machine
TN	True Negatives
TNR	Tue Negative Rate
TP	True Positives
ULP	Ultra Low Power
VOC	Volatile Organic Compound
VSC	Volatile Sulfur Compound

Chapter 1

Introduction

This chapter will present the contextualization, the general motivations, and the objective of this dissertation. The contextualization will frame the work presented in the reality of gas detection in resource-scarce systems. The general motivation's subchapter will describe the main reasons that led the author to choose the topic for his dissertation. Finally, the last subchapter will express the dissertation's goal and the methodology applied during its development.

1.1 Contextualization

Currently, the advancement of Internet-of-Things (IoT) technology has allowed the monitoring and control of gas concentration in systems whose proper functioning depends on the concentration of certain gases. One of the main problems in this type of gas monitoring system is the difficulty in measuring different types of gases with few resources. In other words, implementing artificial intelligence techniques to solve the problem of low selectivity of some gas sensors requires a lot of resources to achieve accurate results.

1.2 General Motivations

Gas detection has become essential in several fields and applications, preventing accidents, avoiding the malfunction of equipments, ensuring the correct gas mixture for patients, and alerting the occurrence of gas leaks in industrial environments. Gas leaks can be significant, causing environmental impacts and problems in people's health. With the development of Internet-of-Things (IoT) technology, applying gas sensors in IoT devices has demonstrated efficiency in monitoring gases in several environments. Sometimes in very complex detection scenarios, gas sensors demonstrate poor cross sensitivity efficiency and low selectivity. Therefore, new gas detection methods were proposed to address these issues, such as developing artificial intelligence algorithms to address the lack of efficiency given by the sensors used. Another very important aspect in some gas detection systems is the choice of a sensor with low power consumption, thus providing gas monitoring over long periods without requiring maintenance. The BME688, according to Bosch, is the first gas sensor with artificial intelligence and low energy consumption [26]. This project seeks to select several artificial intelligence algorithms for gas detection, considering energy consumption. This project is part of the LINK4S project that aims to develop a new smart embedded connected device for monitoring applications.

1.3 Objective

This dissertation intends to study and select several artificial intelligence algorithms for gas detection in scarce resource systems. To this end, different artificial intelligence algorithms will be implemented to be compared, considering the system costs. Thus, the objectives proposed for this dissertation are:

- **BME688 sensor study**
  The BME688 sensor study goal is to understand how this sensor works and its possible application in some industrial environments. It is crucial to understand the sensor behavior since it will affect the data collection process, which is responsible for creating the dataset that will supply all Machine Learning models.

- **Study and development of classifiers for gas detection**
  As mentioned earlier, detection of gas has become essential in many fields and applications, such as preventing industrial accidents, avoiding equipment malfunctions, etc. Consequently, the research for techniques that improve gas sensor performance began to grow. One of these techniques was the development of machine learning models for classifying certain gases in a controlled environment (classifiers). One of the dissertation objectives is to develop such algorithms.

- **Study and development of regressors for gas detection**
  In some situations, predicting discrete class labels (Classification) is not the best approach, since

some systems require a model capable of predicting a continuous quantity (Regressor). In the gas detection field, it is crucial to have regressors to predict the gas concentration value in a specific controlled environment, such as providing the correct gas concentration to patients in hospitals.

- **Comparative analysis and evaluation of system performance for the several algorithms implemented for gas detection**
  After the Machine Learning (ML) models' implementation, the next step is to evaluate the model's performance, taking the system's resources and power consumption into account. The final objective is to provide a list of all models implemented, with their associated prediction errors, enabling an estimation of the quality of each model.

1.4 Methodology

Since one of the objectives of this dissertation is to study the BME688 sensor, the first step will be to analyze all the information made available by Bosch Sensortech (the sensor's developers). This first step will be studying the sensor datasheet and analyzing, in detail, the software provided to communicate with the sensor, allowing experiments with the sensor to understand its behavior when subjected to certain gases. Once the working principle of the sensor is understood, the next step will be to study and develop classifiers and regressors capable of detecting different types of gases. For that, it will be necessary to collect a training dataset using the BME688 sensor. After that, the final step will be to measure the performance of all models trained in the dataset collected. These tests will be executed to compare and discuss which model fits best in the user application, optimizing the resources and memory processing time used to implement such a model.

1.5 Dissertation Structure

This document is divided in six chapters. After this introductory chapter, the literature review of some topics covered in this dissertation is presented. The literature review is one of the most crucial parts of every scientific work since it refers to what has already been discovered about the researched subject, avoiding wasting time with unnecessary investigations. In the following chapter will be described all the basic theoretical concepts behind some AI methods used in this dissertation. Some of these concepts are the operation principle of an SVM classifier, a decision tree, a random forest, and an ANN. Chapter four presents the system specification, which explains the workflow of the system that makes up this dissertation, describing the hardware and software used. After that, the implementation chapter is shown to describe the whole implementation process. The implementation chapter will detail all the steps used to implement the system mentioned in the previous chapter, including code fragments and configurations, as well as tests and performance results of the ML models used to build the system. The last chapter presents all the conclusions drawn about the development of this dissertation and the future steps to improve the final solution.

Chapter 2

Literature Review

2.1 Background on Environmental Gases

Some gases are fundamental to some systems and industries, being the main ones responsible for many of the problems that may occur in the environments where they are used. In this subsection, the principle gases that are used in most industries will be listed and explained.

- **Oxygen** (O_2) is the principal gas for life. In a hospital, for example, in some cases, it is essential to maintain the oxygen concentration administered to a particular patient who is recovering from surgery or disease. The decrease in the oxygen concentration in enclosed spaces is sometimes related to the escape of other gases, leading people inside these spaces to asphyxiation. Several industrial processes depend on the correct oxygen concentration to ensure the best performance of the systems. Regardless of the fuel level or the speed at which they are running, engine systems depend on the oxygen level to deliver the desired performance [14].
- **Carbon dioxide** (CO_2) is a colorless, odorless gas that results from the oxidation and combustion of hydrocarbons, just as from the breathing process of living things. This gas is the principal cause of the greenhouse effect, and the increase in the level of its concentration in the presence of other gases is related to air pollution. Carbon dioxide also plays a significant role in the oxygen production process through photosynthesis. The increased concentration level of this gas in enclosed spaces can lead to asphyxiation, which can become deadly if its concentration reaches values above 3% [14].
- **Volatile organic compounds** (VOCs) are carbon-based organic compounds that exist in the vapor state at room temperature and result from the combustion of fossil fuels or natural emissions. Some

of these compounds are toxic, affecting the health of humans by causing respiratory diseases, heart disease, or even cancer [14].

- **Hydrogen** (H_2) is a colorless, odorless, non-toxic, and very flammable gas. This gas becomes flammable above a certain concentration level. Its density is 14 times less than the air density. Hydrogen is used in the production of chemicals and intermediate products. The mixture of hydrogen and carbon monoxide is part of the production processes for methanol and other products, such as aldehydes [16].
- **Methane** (CH_4) is a colorless and odorless gas. It is also known as marsh gas or methyl hydride. The vapors are lighter than the air, and under prolonged exposure to fire, the CH4 containers may rupture violently and rocket. It is used to make other chemicals. Methane is the main component of natural gas, used mainly to produce light and heat [22].

Monitoring and controlling the concentration levels of these gases have become very important in industry and human welfare, preventing accidents. This monitoring also makes it possible to reduce pollution levels in the air, improving people's quality of life and reducing the number of cases of respiratory diseases.

2.2 Sensing Technologies

Methods used to detect different types of gases depend on the physical or chemical changes of a given material in the presence of a gas. Some of these methods use the reaction between the gas sensor material and the target gas to determine the gas concentration. Other methods compare physical properties such as propagation speed and wave propagation between an ideal medium and a medium with gas. The gas detection techniques that yield the best results are listed below.

- **Electrochemical**
  The working principle of electrochemical sensors consists of chemical reactions between the sensor material and the target gas. The product resulting from this reaction and its speed is proportional to the concentration level of the target gas. These types of sensors consist of two electrodes (working electrode and counter electrode) and an ion conductor that separates the two electrodes (figure 1). When a gas such as carbon dioxide (CO_2) approaches the working electrode, it will oxidize the electrode through chemical reactions with the water molecules in the air, generating protons (H^+) in the working electrode (equation 1 and 2).

$$CO + H_2O \rightarrow CO_2 + 2H^+ + 2e^- \tag{1}$$

$$2H^{+} + \frac{1}{2}O_2 + 2e^{-} \rightarrow H_2O \qquad (2)$$

Short-circuiting the two electrodes will allow the protons (H$^+$) to travel between the working electrode and the counter electrode through the ion conductor that separates them. This flow of protons will generate current in the circuit between the working electrode and the counter electrode. This flow of electrons will be proportional to the gas concentration level in the working electrode [14, 33].

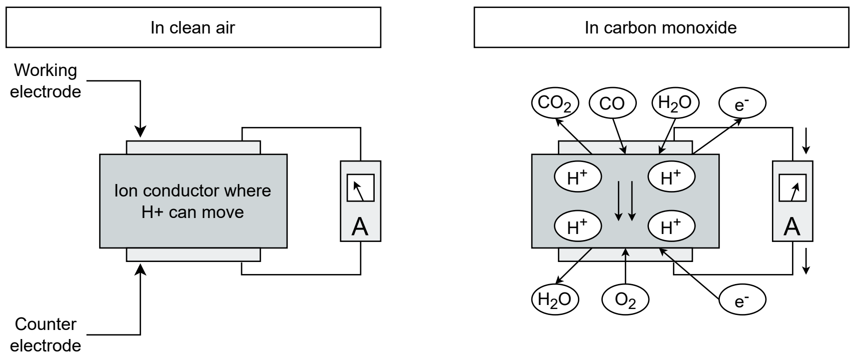


Figure 1: Working principle of electrochemical gas sensors (based on [33])

- **Metal Oxide Semiconductors (MOS)**

  MOS gas sensors mainly consist of tin dioxide semiconductor particles. In clear air, donor electrons of this semiconductor are attracted by external oxygen molecules, remaining on the surface of the semiconductor. This phenomenon prevents the flow of electric current in the sensor. When the sensor surface is subjected to reducing gases, the oxygen molecules will decrease due to the reactions between the new gases and the oxygen in the air, releasing the tin dioxide donor electrons and consequently creating an electron flow in the sensor. Finally, the gas concentration level can be measured by the current flow in the sensor [34].

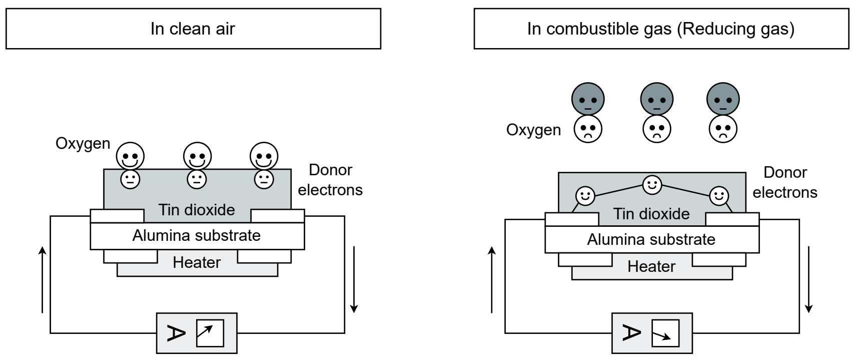


Figure 2: Working principle of MOS gas sensors (based on [34])

When the semiconductor particles are heated in air at high temperatures, oxygen is absorbed onto the surfaces of these particles by capturing free electrons. The depletion layer formed depends on the size of the semiconductor particles. If the particles are small (conventionally used in gas sensors), the depletion layer will extend over the entire area of each particle (High sensitivity). On the other hand, if the particle size is much larger, the depletion layer will be located on the periphery of the particles (Low sensitivity) [34].

These types of sensors are used in hospitals to ensure the correct mixture of oxygen (O_2) and nitrogen (N_2) for some patients. These sensors are also simple to manufacture and low cost relative to other gas detection technologies [14].

- **Acoustic**
  Sound propagates differently in the different propagation mediums. The speed at which sound travels through a medium containing gases depends on the temperature, pressure, humidity, and gas properties. Acoustic waves can detect the presence of a specific gas in the environment. Gases can be analyzed using sound speed, signal attenuation, acoustic impedance, or a combination of these features. Figure 3 demonstrates the schematic of an acoustic sensor.

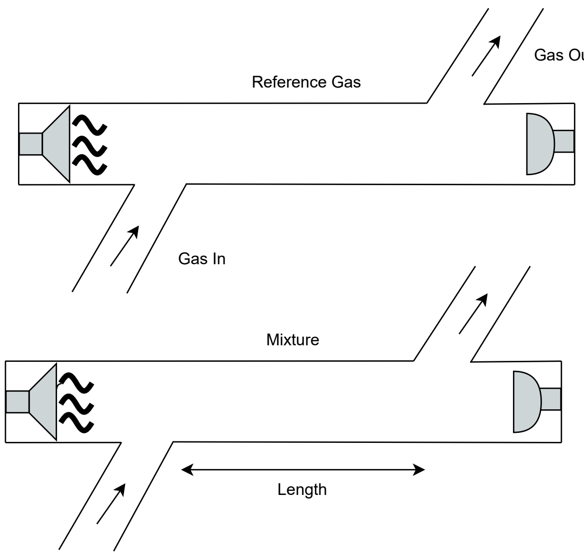


Figure 3: Ultrasonic speed-based gas sensor, which measures the target gas by a comparison of the sound speed in the air and the sound speed in the presence of the target gas (based on [14])

The most used and researched method is based on the sound speed difference in the different propagation mediums, using the propagation time in a defined distance to identify the composition of a particular gas mixture. The following equation (3) shows the relationship between sound speed and the gaseous propagation medium.

$$c = \sqrt{\frac{kRT}{M}} \tag{3}$$

Here c is the sound velocity, in meters per second, k the average specific heat ratio, R the gas constant, T the temperature in Kelvin, and M the average molecular weight. This method helps to measure gas concentration in systems with only two types of gases. When more gases come into such systems, they can indicate false positives due to the presence of another gas in proportions that would produce the same sound velocity in the propagation medium. A sensor that combines this technology with another detection approach can improve detection accuracy and selectivity. The attenuation of the acoustic signal refers to the scattered energy over a certain propagation distance. When a signal travels through a gaseous medium, the gaseous equilibrium is disrupted, and the molecules collide, exchanging energy and generating heat energy. The amount of energy generated is proportional to the amount of energy consumed by the gaseous molecules that cause the collisions. The system absorbs the acoustic energy, and the amount absorbed is proportional to the concentration of the gases that make up the propagation medium. Equation 4 is used to calculate the signal's attenuation, where P_O and P are the acoustic pressure in two different points, α is the attenuation coefficient, and X is the distance from P_O to P [14].

$$P = P_O e^{(-\alpha X)} \tag{4}$$

Acoustic impedance analysis is a less well-studied approach for sensing gases, with no commercial applications yet discovered. Equation 5 is used to determine the acoustic impedance (Z), ρ is the gas density, and c is the sound speed. This approach can lead to detection mistakes, since a particular gas mixture can be confused with another due to its different density [14].

$$Z = \rho c \tag{5}$$

- **Catalytic**

  Catalytic gas sensors are used to detect the presence of combustible gases in environments with O_2 concentrations of at least 15%. The working principle of these sensors is based on the measurement of the internal resistance variation of the sensor. This gas detection process consists of a chemical reaction between the sensor's catalytic element and the gas present in the environment, causing an increase in the sensor's temperature and a subsequent change in its internal resistance. These sensors are quick to react, but are not particularly good at detecting the gas concentration to which they are subjected. These sensors can be installed in environments with variations in humidity and temperature without losing their ability to detect the target gases. Under normal conditions, their operating time is approximately ten years, but if they are in the presence of corrosive gases in high concentrations, their lifetime will drastically reduce [14].

Catalytic sensors consist of a sensing element (catalytic material sensitive to combustible gases) and a passive material (inert material) (figure 5). Combustible gases will be consumed only by the sensing material, causing a temperature increase and, thus, the internal resistance change of the sensor. The passive material will not change in the presence of combustible gases. The Catalytic sensors structure has a circuit that connects the two materials (catalyst and passive). When the sensor is in the same environment of combustible gases, the resistance of the sensing material will increase, causing the output signal of the circuit to vary. This signal is directly proportional to the concentration level of the combustible gas surrounding the sensor [32].

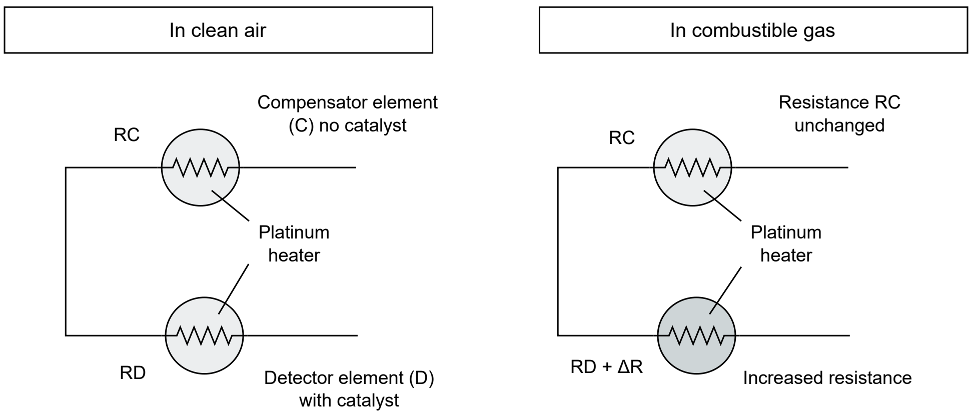
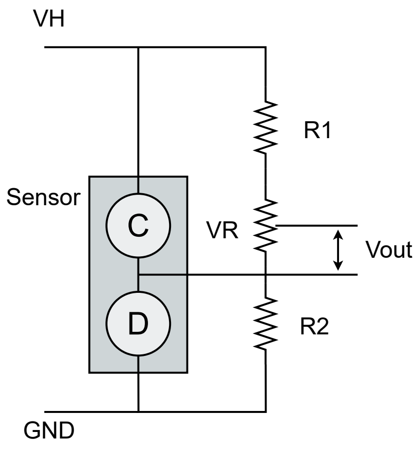
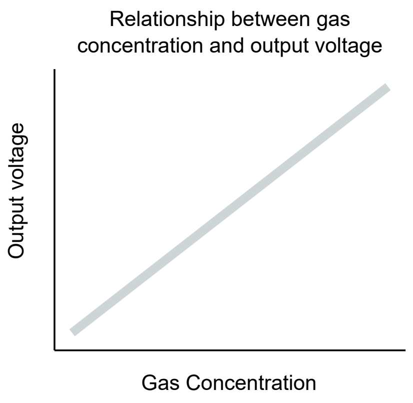


Figure 4: Working principle of Catalytic gas sensors (based on [32])



(a) Gas circuit, Wheatstone bridge (VR is responsible for maintaining a state of balance of the bridge circuit in clear air)



(b) Catalytic relationship between gas concentration and output voltage

Figure 5: Catalytic gas sensor circuit and the relationship between gas concentration and output voltage (based on [32])

- **Optic**

  One of the properties of gases is their ability to absorb different wavelengths. Many of the gases and the corresponding absorbed wavelengths are listed in the High-Resolution Transmission Molecular Absorption Database (HITRAN). Many gas detection techniques use this optical principle to determine the concentration of a specific gas. On the other hand, some methods rely on adding materials that react with gases in the presence of light, emitting different wavelengths, and absorbing or reflecting the emitted light.

Nanomaterials, such as polymers or metals, are typically employed to create these effects. Fluorescence-quenching sensors (figure 6) are usually composed of a matrix of nanomaterials permissible and sensitive to the target gas nanoparticles, primarily polymers or metals, coupled to the tip of a fiber optic using a 50:50 Y-type optical coupler. A light source is connected to one end of the coupler, while an optical spectrometer is attached to the other. With the appropriate wavelength and energy, the polymer matrix will react and emit a distinct wavelength, with the intensity of the light corresponding to the gas concentration [14].

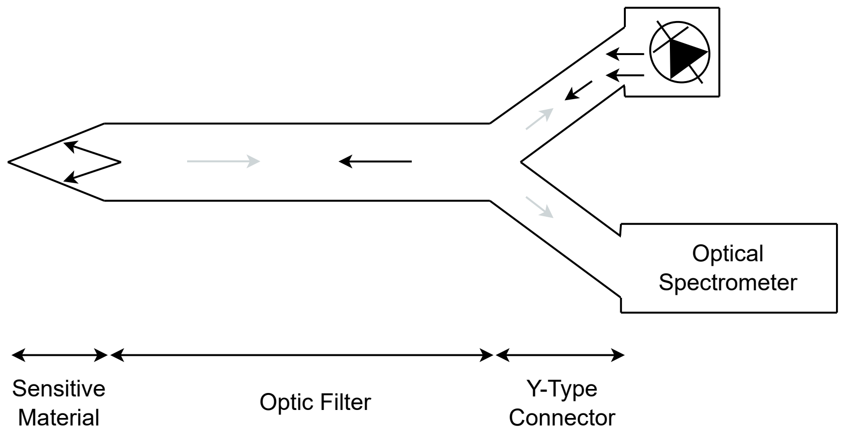


Figure 6: The fluorescent-quenching effect gas detector, this measures the target gas by evaluating the reflected wavelength from the sensing material on the presence of the gases (based on [14])

2.3 Background on gas sensors

Gas sensors are devices developed to detect the presence and concentration levels of specific gases in the environment. The operating principle of this type of device is to measure internal changes that occur when exposed to certain gases.

2.3.1 MQ gas sensors

There are many gas sensors on the market today, and the most widely used in various applications are the MQ-type gas sensors. These sensors have several operating principles, but the most used is the MOS gas sensor. As mentioned earlier, these sensors consist of a semiconductor that has free electrons (the semiconductor usually used is tin dioxide). In the air, there is a higher concentration of oxygen than in combustible gases, so the oxygen molecules will attract the free electrons from the semiconductor, thus preventing the generation of an electron flow in the sensor. When the sensor is in the same environment as combustible gases, these will react with the oxygen molecules, releasing the electrons from the semiconductor. This phenomenon will cause a flow of electrons inside the sensor, which can be used to measure the gas concentration level.[4] The following table (table 1) lists the different types of MQ-type gas sensors and the gases that each one is responsible for detecting.

Table 1: List of different types of gas sensors and what gases they sense

Sensor Name	**Gas to measure**
MQ-2	Methane, Butane, LPG, Smoke
MQ-3	Alcohol, Ethanol, Smoke
MQ-4	Methane, CNG gas
MQ-5	Natural gas, LPG
MQ-6	LPG, butane
MQ-7	Carbon Monoxide
MQ-8	Hydrogen Gas
MQ-9	Carbon Monoxide, flammable gases
MQ131	Ozone
MQ135	Air Quality
MQ136	Hydrogen Sulphide gas
MQ137	Ammonia
MQ138	Benzene, Toluene, Alcohol, Propane, Formaldehyde gas, Hydrogen
MQ214	Methane, Natural gas
MQ216	Natural gas, Coal gas

2.3.2 Bosch BME688 gas sensor

The BME688 gas sensor, according to Bosch, is one of the first gas sensors having artificial intelligence integrated with pressure, humidity, and temperature sensors. This sensor was developed for mobile applications and systems where energy consumption is a crucial requirement. The BME688 was also designed to detect volatile organic compounds (VOCs), volatile sulfur compounds (VSCs), and other gases such as carbon monoxide [26]. The following table (table 2) shows the technical characteristics of the BME688 sensor.

Table 2: BME688 Technical data (from [26])

Parameter	**Technical data**
Package dimensions	• No measurements are performed
Operation range (full accuracy)	• Pressure: 300 ... 1100 hPa
	• Humidity: 0 ... 100%
	• Temperature: -40 ... 85°C
Supply voltage VDDIO	• 1.2 ... 3.6V
Supply voltage VDD	• 1.71 ... 3.6V
Interface	• I_2C and SPI
Average current consumption	• 2.1 μA at 1 Hz humidity and temperature
	• 3.1 μA at 1 Hz pressure and temperature
	• 3.7 μA at 1 Hz humidity, pressure, and temperature
	• 90 μA at ULP mode for p/h/T & air quality
	• 0.9 μA at LP mode for p/h/T & air quality
	• 3.9 mA in standard gas scan mode
Gas Sensor	
Sensor-to sensor deviation (IAQ)	• +/-15% +/-15 IAQ
Standard scan speed	• 10.8 s/scan
Electric charge for standard scan	• 0.18mAh (5 scans 1 min)
Major sensor outputs	• Index for Air Quality (IAQ), bVOC- & CO_2-equivalents (ppm), Gas scan result (%) & Intensity level
Humidity sensor	
Response time	• 8 s(τ 0-63%)
Accuracy tolerance	• ±3% relative humidity
Hysteresis	• ≤1.5% relative humidity
Pressure sensor	
RMS Noise	• 0.12 Pa (equiv. to 1.7 cm)
Sensitivity Error	• ±0.25 %
Temperature coefficient offset	• ± 1.3 Pa/K
Temperature sensor	
Absolute accuracy	• +/-0.5 °C (25 °C)

The BME688 is housed in a metal lidded 8-pin LGA box. Its dimensions are 3.0 × 3.0 × 0.93 mm, with an 8.4 mm^3 container volume. Figure 7 shows the photographs of the top and bottom packages. The sensors are exposed to the environment through a round aperture in the left upper corner of the metal lid top surface. The package surface interconnection has eight contact pads and metal traces on the bottom surface [19].

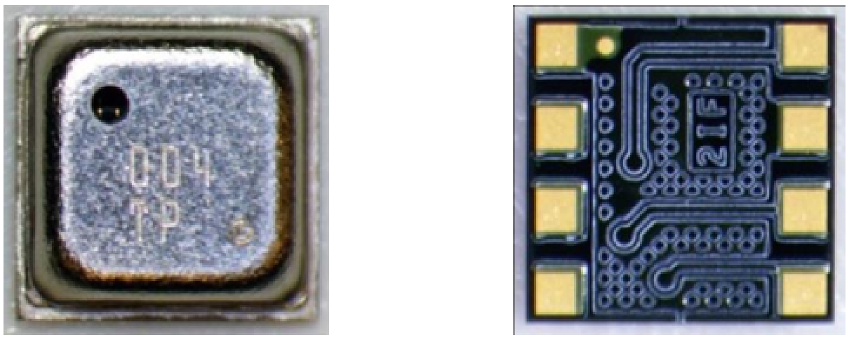


Figure 7: Top and bottom package images (from [19]

Figure 8 shows a shot of the BME688 after the metal lid has been removed. There are two sensor dies visible. On the left, it is possible to see how the pressure, temperature, and humidity sensors are all incorporated into one substrate (integrated MEMS). The gas sensor is located on the right-hand side of the second die. Below the left die, i.e., integrated MEMS is the SoC controller or ASIC [19].

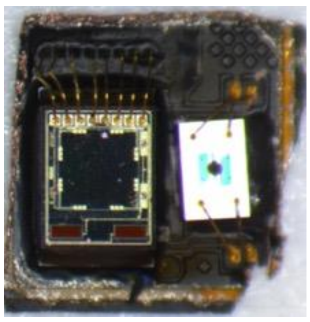


Figure 8: A BME688 assembly after the metal lid is removed (from [19])

Figure 9 is an image of the integrated MEMS die. The pressure sensor is likely piezoelectric-based and is situated in the middle of the die. The temperature sensor is at the top of the integrated MEMS, above the pressure sensor, and is most likely a semiconductor diode. To the right of the pressure sensor is the humidity sensor. This sensor is a pair of capacitor plates with a porous polymer-based dielectric. The embedded MEMS are connected to the metal traces of the package substrate via eight bond wires [19].

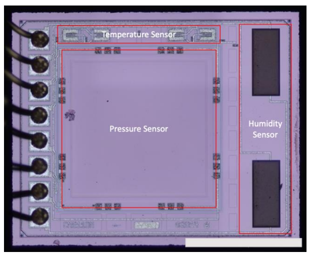


Figure 9: Die photo of integrated MEMS sensor (from [19])

Figure 10 illustrates the gas sensor. It is based on a gas-sensitive layer (dark blue circle in figure 10) that sits above two electrodes in the middle of the die. This sensor is bounded to metal traces on the package substrate via four bond wires [19].

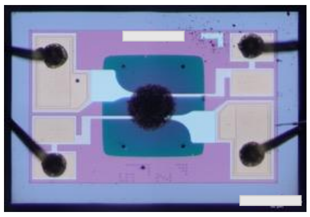


Figure 10: Gas sensor (from [19])

2.4 Machine learning applied to gas detection

Unlike other sensing types, such as force sensing or temperature sensing, which detect a single variable, the principle, and the tasks of gas sensing are complex. Therefore, gas sensing encounters many challenges:

- **Cross sensitivity**
  A gas sensor that detects a given gas based on its chemical properties can be affected by gases with identical chemical characteristics to the target gas. In other words, a gas sensor may show readings of gases that are not the target gas. An example is the case of Hydrogen Sulfide (H_2S) sensors that react to Hydrogen (H_2) [12].

- **Low selectivity**
  The accuracy of a gas sensor depends on the temperature and humidity to which it is subjected. Since gas activity is affected by temperature, the absorption capacity of a gas sensor is dependent on that. On the other hand, since humidity affects the interaction of gases with the sensor, the results obtained by the sensor become unpredictable. Most gas sensors use changes in their internal resistance to detect the presence of certain gases. These changes can sometimes result from the high temperatures at which the sensor is located, leading to errors in the sensor's reading [12].

The concept of "smart gas sensing" has been proposed as a way to solve all the problems mentioned above, in other words, the use of machine learning to get around such obstacles. Several machine learning models may be used in the gas detection field. The following list will discuss the most well-known ML models used by most data scientists.

- **Linear Regression**

  By fitting a linear equation to observable data, linear regression seeks to model the connection between two variables. One variable is regarded as an explanatory variable, while the other is considered a dependent variable [36].

- **Support Vector Machines (SVMs)**

  A Support Vector Machine (SVM) is a strong and flexible Machine Learning model that can do linear and nonlinear classification, regression, and even outlier detection. It is one of the most widely used Machine Learning models. SVMs are particularly well adapted to classifying complex datasets that are small or medium. SVM aims to draw a line or a hyperplane that divides data into classes. An SVM classifier takes the data as input and outputs a hyperplane that separates the input data into classes [15].

- **Decision Trees and Random Forests**

  Decision Trees and Random Forests are versatile Machine Learning algorithms that can perform both classification and regression tasks. They are influential algorithms, capable of fitting complex datasets. The fundamental components of Random Forests are the Decision Trees, which are among today's most powerful Machine Learning algorithms [15].

- **Artificial Neural Networks (ANNs)**

  Node layers in artificial neural networks (ANNs) include an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, is linked to another and has its weight and threshold. If the output of any particular node exceeds the given threshold value, that node is activated and sends data to the network's next tier. Training data is essential for neural networks to develop and enhance their accuracy over time [17]. Figure 11 shows a deep neural network example with one input layer, three hidden layers and one output layer.

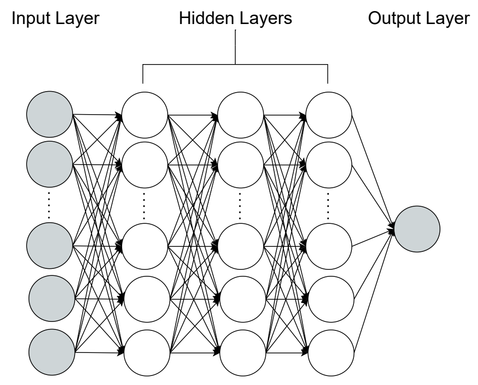


Figure 11: Deep Neural Network example (based on [17])

2.5 Selection Criteria

There are several ways to evaluate the performance of a machine learning model on a given dataset. One of these is the calculation of RMSE (Root Mean Square Error).

2.5.1 Root Mean Square Error (RMSE)

The Root Mean Square Error (RMSE) is a commonly used measure of the variations between predicted values and observed values. Training set and test set creation are crucial in machine learning model development. The latter will be used to evaluate the performance of the final model by calculating the RMSE. The RMSE measures the standard deviation of the errors that the ML model makes in its prediction. Equation 6 shows the mathematical formula to compute the RMSE, where n is the number of instances in the train set, Y_i is the vector of observed values, and $\hat{Y}$ is the predicted values [15].

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y})^2} \tag{6}$$

2.5.2 Akaike Information Criterion (AIC)

AIC is a single numerical score that can be used to identify the Machine Learning model that best fits a dataset among several others. AIC scores are only helpful when compared to other AIC scores for the same dataset because it estimates models relativistically. A lower AIC score is better. AIC operates by determining how well the model fits the training set and then adding a penalty term for model complexity [37]. Equation 7 shows how AIC score can be calculated. AIC uses a model's maximum likelihood estimation (log-likelihood) as measures of fit. In equation 7 L represents the likelihood and k represents the number of parameters.

$$AIC = -2ln(L) + 2k \tag{7}$$

Chapter 3

Theory

3.1 Training Models

The next chapter aims to understand how most machine learning algorithms/models work, such as the linear regression model, the gradient descent algorithm, a polynomial regressor, and a logistic regressor. All these algorithms are essential for developing more complex Machine Learning models, so it is necessary to have a good understanding of how they work.

3.1.1 Linear Regression

As mentioned in the literature review, linear regression seeks to model the connection between two variables. The dependent variable is the one that will be predicted, and the independent variable is the one that is used to forecast the value of the dependent variable. So, in other words, linear regression fits a linear equation to the observed data to model the relationship between two variables. Generally, a linear model computes the weighted sum of the input features plus a variable called the bias term to create a prediction, as given in equation 8.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \theta^T \mathbf{x} \tag{8}$$

where $\hat{y}$ represents the predicted value, n represents the number of features, x_i means the i^{th} feature value, and θ_j is the j^{th} model parameter.

To train the linear regression model, it is necessary to measure how well the model fits the training data. The most common performance measure for a regression model is the Root Mean Square Error (shown in equation 9).

$$
\begin{aligned}
RMSE &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2} \\
&= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta^T \mathbf{x}^{(i)})^2}
\end{aligned} \tag{9}
$$

In the training process, the goal is to find the value of θ that minimizes the RMSE. Figure 12 represents an example of three linear models. Two (gray lines) have a higher RMSE than the one presented in black.

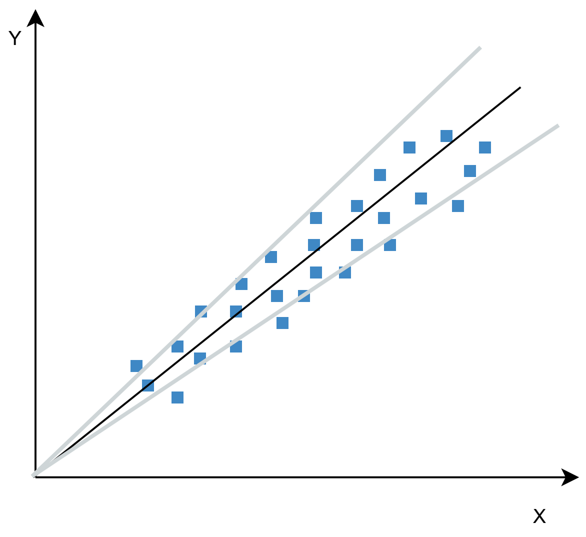


Figure 12: Linear Regression Example (based on [15])

3.1.2 Gradient Descent

Gradient Descent is a general optimization algorithm that can locate the best answers for many issues. Gradient Descent's goal is to iteratively adjust parameters to reduce the cost function, which measures how wrong a model is, in finding a relation between the input and output. In other words, it tells how badly a model predicts. Gradient Descent is used to find a local minimum/maximum of a given function. This method is commonly used in Machine Learning [15, 20]. The Gradient Descent algorithm does not work in all functions. There are two criteria that a function must respect:

- Differentiable
- Convex

Before explaining how the Gradient Descent algorithm works, it is first necessary to understand what a gradient is. The slope of a curve at a given location in a particular direction is the gradient. It is the first derivative at a chosen point for a univariate function. In the case of multivariate functions, is a vector of derivatives in each main direction (along variable axes) called partial derivatives.

The Gradient Descent algorithm (shown in equation 10 and in figure 13) iteratively calculates the next point, measuring the local gradient of the model cost function, scales it with the learning rate, and subtracts the obtained value from the current position. The cost function gradient is represented by the letter θ in the following graphs [20].

$$\theta_{n+1} = \theta_n - \eta \nabla f(\theta_n) \tag{10}$$

The parameter η represents the learning rate, which scales the gradient and controls the step size.

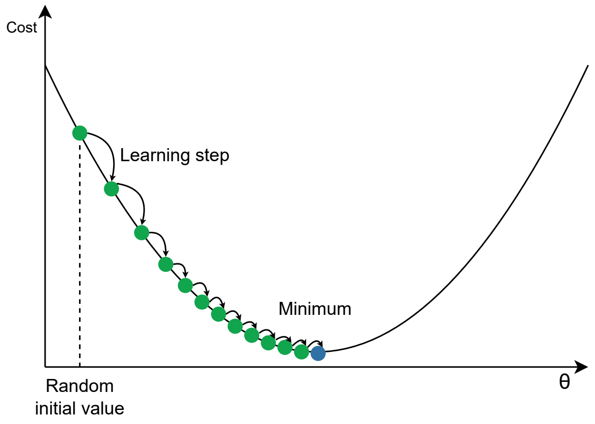


Figure 13: Gradient descent algorithm

The learning rate has a strong influence on the performance of the Gradient Descent algorithm. For example, if the learning rate is too short, the algorithm will have to go through many iterations to converge, taking a long time to do it (figure 14).

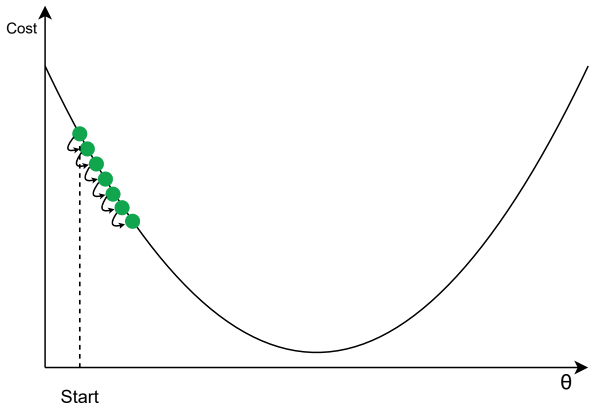


Figure 14: Learning rate too small

Meanwhile, if the learning rate is too big, the algorithm may not converge to the optimal point (it will jump around). Figure 15 represents the Gradient Descent algorithm with a learning rate too big.

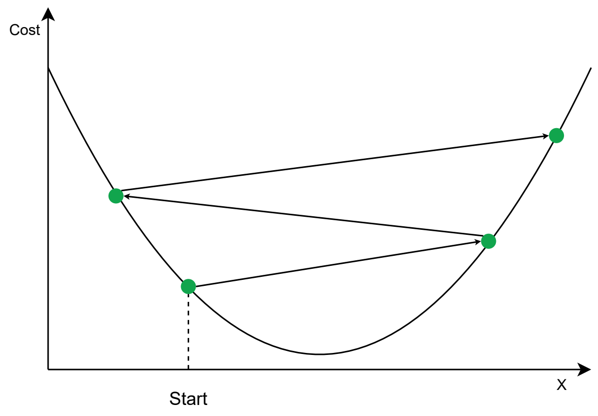


Figure 15: Learning rate too large

Not all cost functions are convex; some may feature ridges, plateaus, or holes that make the convergence process to the minimum very difficult. Figure 16 shows some problems with the Gradient Descent algorithm. The method will converge to a local minimum, which is inferior to the global minimum, if the random initialization starts the algorithm on the left. On the other side, if it starts the algorithm on the right, it will take a very long time to cross the plateau. If the algorithm quits too soon, it will never reach the global minimum [15].

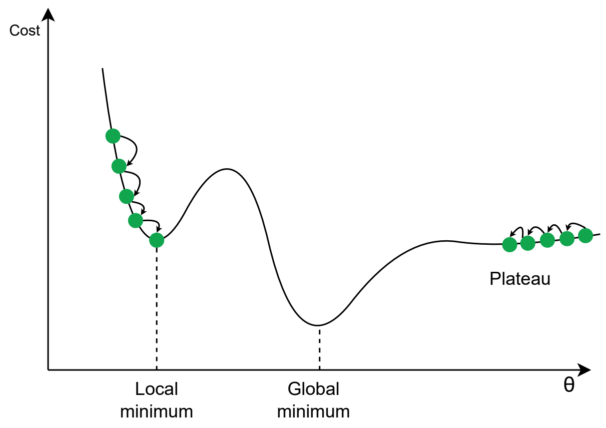


Figure 16: Gradient descent algorithm in a non-convex function

3.1.3 Polynomial Regression

If the data is more complex than a simple straight line, it is impossible to use a linear model to fit the data. One way to overcome this, is to add new features to the original data space. For example, in the case of a 2-dimensional nonlinear dataset, sometimes it is necessary to add another dimension to make the dataset linear.

Figure 17 represents a nonlinear dataset, and figure 18 shows the dataset in a new data space, where it is possible to see the mapping function responsible for the new feature (x_poly[1]), the plane resulted from the linear regression applied to the new dataset and the interception between these two. The resulting intersection line corresponds to a polynomial equation in the plane of x_poly[0] and y (axes of the original space 18b). In this new polynomial equation, it will be the model responsible for predicting instance in the original space.

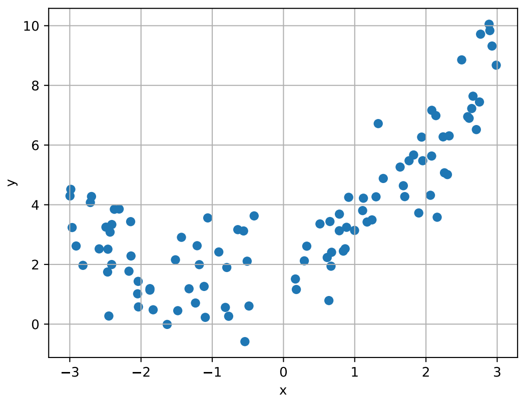
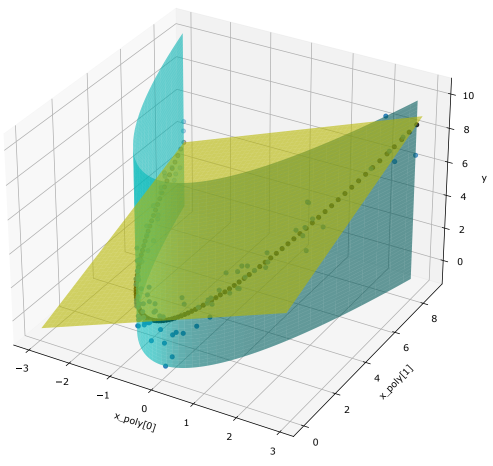
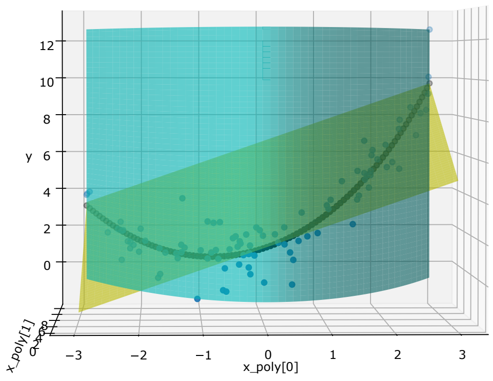


Figure 17: Nonlinear data (based on [15])



(a) New data space (mapping function x^2)



(b) New data space (mapping function x^2) in another viewing angle

Figure 18: Polynomial Regression Example

3.1.4 Logistic Regression

A machine learning technique called logistic regression, based on the idea of probability, is used to solve classification problems. This technique computes a weighted sum of the input features using a logistic regression model, just like a linear regression model. Rather than directly displaying the result as the linear regression model does, it shows the logistic of this result [15].

The logistic function is a sigmoid function, that outputs a number between 0 and 1. It is defined as shown in equation 11 and figure 19.

$$\sigma(t) = \frac{1}{1 + exp(-t)} \tag{11}$$

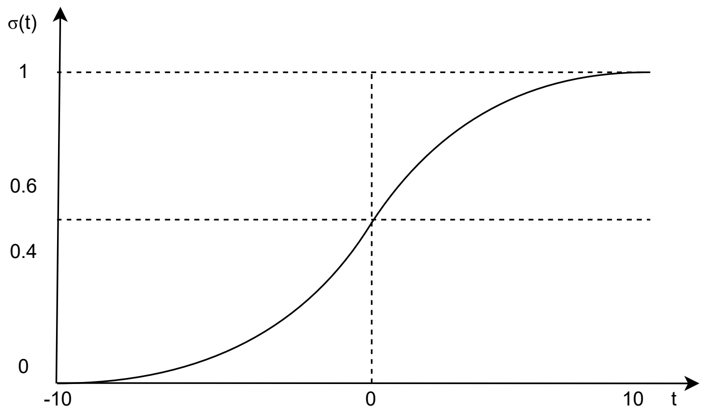


Figure 19: Nonlinear data

The Logistic Regression model may readily make its prediction once it has calculated the probability $\hat{p} = \sigma(\theta^T \mathbf{x})$ that an instance x belongs to a class. If $\sigma(t) < 0.5$ when $t < 0$, and $\sigma(t) \geq 0.5$ when $t \geq 0$, so logistic regression model predicts 1 if $\theta^T \mathbf{x}$ is positive, and 0 if it is negative [15].

The training objective of a logistic regression model is to find the vector θ, which makes the model estimate a high probability for positive instances (y=1) and a low probability for negative (y=0). The cost function for a single training instance is shown in equation 12.

$$c(\theta) = \begin{cases} -log(\hat{p}), & \text{if } y = 1, \\ -log(1 - \hat{p}), & \text{if } y = 0. \end{cases} \tag{12}$$

In the cost function, it is possible to see that $-log(\hat{p})$ grows when $\hat{p}$ approaches 0. In this scenario, the result of the cost function will be larger if the model estimates a probability close to zero for a positive instance. The cost function result will also be large if the model estimates a probability close to 1 for negative instances ($-log(1 - \hat{p})$). On the other hand, $-log(\hat{p})$ is close to 0 when $\hat{p}$ is close to 1, which means that, for negative instances, the model must predict a probability close to zero. The equation 13 shows the cost function over the whole training set, which is the average cost over all training instances. This equation is called *log loss*.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} log(\hat{p}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{p}^{(i)})] \tag{13}$$

Because equation 15 is a convex function, the Gradient Descent is guaranteed to find the global minimum [15].

3.2 Classification

Predicting the class of a set of data points is the classification process. Targets and labels are all common names for categories. Approximating a mapping function (f) from input variables (x) to discrete output variables is the task of classification predictive modeling (y). There are many applications in classification in many domains such as credit approval, medical diagnosis, target marketing, etc. [2].

3.2.1 Performance measures

There are several ways to measure a classifier's performance, some of which are listed below.

- **Accuracy score**
  Accuracy score is a value that measures the classification performance. This value is calculated through equation 14.

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i) \tag{14}$$

  where $\hat{y}_i$ is the predicted value of the i^{th} sample and y_i is the corresponding true value. In other words, the fraction between the correct predictions and the $n_{samples}$ is used to calculate the accuracy score of a respective classification model.

- **Cross-Validation**
  A good way to evaluate a model is to use cross-validation. Cross-validation is the process of making predictions and assessing them on each fold (a fold is a subset of the training dataset) using a model learned on the other folds after splitting the training set into K-folds.

- **Confusion Matrix**
  Another way to evaluate the performance of a classifier is analyzing the confusion matrix. The general idea of a confusion matrix is to count the number of times instances of class A are classified as class B (In machine learning, a class is the output category of the data). Each row in a confusion matrix represents an actual class, while each column represents a predicted class. Figure 20 represents a confusion matrix of a classifier trained to classify if an image is a number 5 or not.

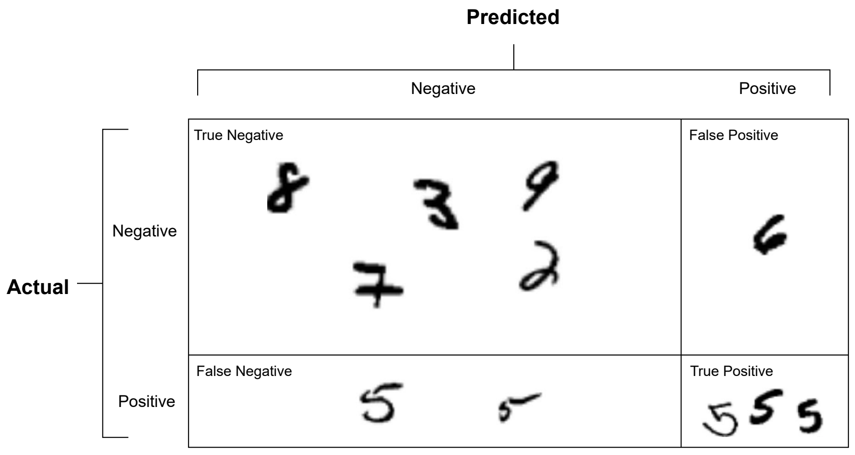


Figure 20: Illustration of a confusion matrix for 5-images classifier

The first row of the confusion matrix considers non-5 images (negative class), and the second row considers 5 images (positive class). In the negative class, the first column represents the number of instances that were correctly classified as non-5s (true negatives), and the second column shows the number of instances wrongly classified as 5s (false positives). On the other hand, the second row considers the 5s images (positive class), where in the first column are represented the images that were wrongly classified as non-5s (false negatives), and in the second column the images that were correctly classified as 5s (true positives) [15].

- **Precision and Recall**

  The confusion matrix gives a lot of information about a classifier, but sometimes most of that information is not needed, and a more concise metric can be used. One of those metrics can be the accuracy of the positive predictions, called precision. Equation 15 shows how to calculate a classifier precision, where TP is the number of true positives, and FP is the number of false positives.

$$precison = \frac{TP}{TP + FP} \tag{15}$$

  Another metric that can be useful is the recall method, also called the classifier sensitivity. This metric is responsible for indicating the ratio of positive instances that are correctly detected by the classifier (Equation 16) [15].

$$recall = \frac{TP}{TP + FN} \tag{16}$$

  FN is the number of false negatives.

3.3 Machine Learning Models Background

This section explains in detail all the models that will be used throughout this dissertation. Starting with the classifiers and finishing with the regressors.

3.3.1 Support Vector Machine Classifier

A Support Vector Machine (SVM) is a strong and flexible Machine Learning model that can do linear and nonlinear classification. It is one of the most widely used Machine Learning models. SVMs are particularly well adapted to classify complex datasets that are small or medium. The goal of SVM is to draw a line or a hyperplane that divides data into classes. An SVM classifier takes the data as input and outputs a hyperplane that separates the input data into classes.

The fundamental idea of an SVM model can be explained using the dataset shown in figures 21 and 22 (For simplicity, it will be discussed in cases of datasets with only two classes). The two classes presented in this dataset can be separated by a line, i.e., these two classes are linearly separable. In figure 21 it is possible to verify the presence of several classifiers that separate the two classes. None of these classifiers will work correctly in the presence of new instances, since these lines are not as far as possible from the instances of the two classes.

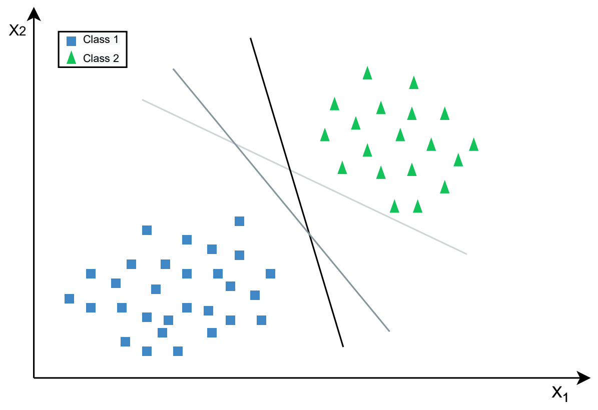


Figure 21: Dataset with two classes linearly separable

The line in figure 22, on the other hand, indicates an SVM classifier's decision boundary. This line not only separates the two classes, but also remains away from the closest training instances as possible.

Conforming to an SVM algorithm, the points closest to the line from both classes are called support vectors. The distance between the line and the support vectors is called the margin. The main objective in an SVM algorithm is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane [25].

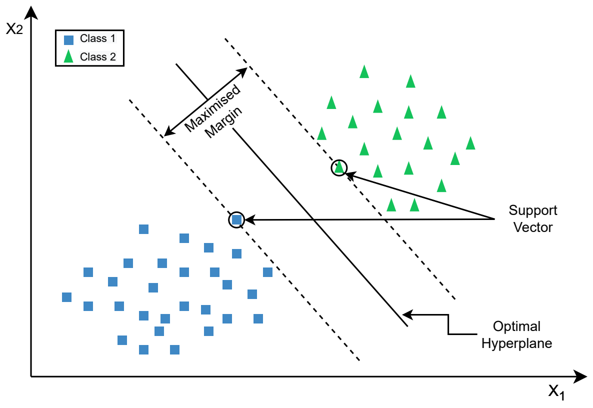
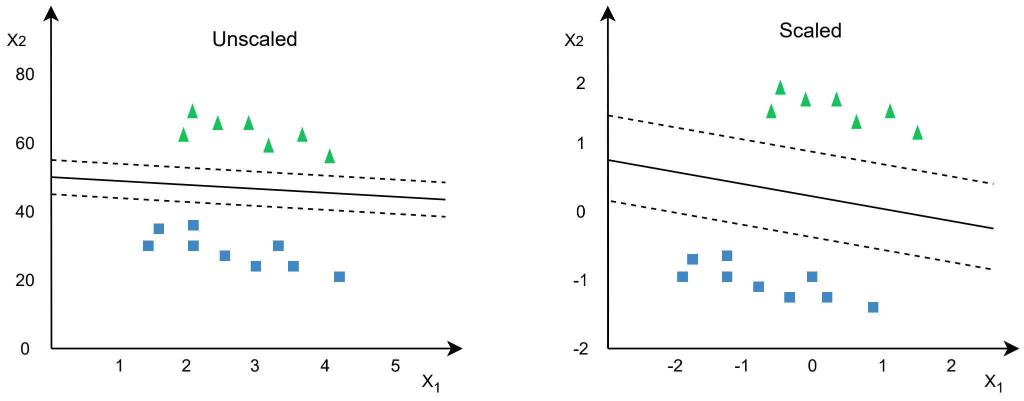


Figure 22: Dataset with two classes linearly separable and the optimal hyperplane

SVMs are sensitive to the feature scales, as it can be seen in figure 23, the left plot (figure 23a) has a vertical scale much larger than the horizontal scale, leading to a greatly reduced margin. In the opposite side, the decision boundary looks much better on the right plot (figure 23b) [15].



(a) Dataset unscaled

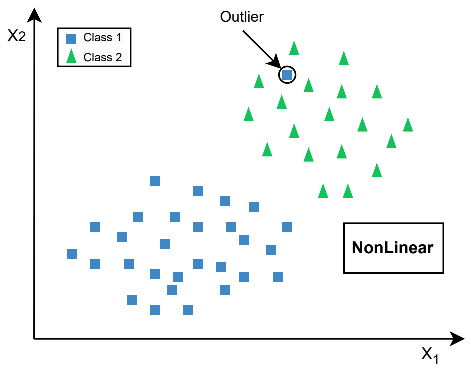
(b) Dataset scaled

Figure 23: Dataset unscaled and scaled

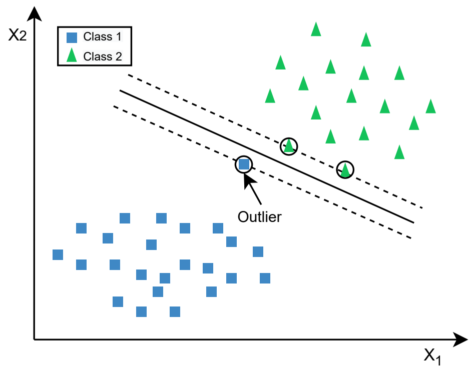
- **Hard Margin Classification**

  Hard margin classification imposes that all instances (data) must be off the street (i.e., the area inside the parallel dashed lines in figure 22) and on the right side (i.e., instances that belong to a particular class must be on one side of the margin while instances of the other class must be located on the other side of the margin). Hard margin classification has two major drawbacks. It only works if the data is linearly separable, and it is quite sensitive to outliers.

  The plot on the left of figure 24 represents a non-linearly separable dataset, making it impossible to apply the hard margin classification method. On the other hand, the plot on the right corresponds to a linearly separable dataset with a decision boundary very close to the data of one class due to the presence of one outlier.



(a) Non-linearly separable dataset



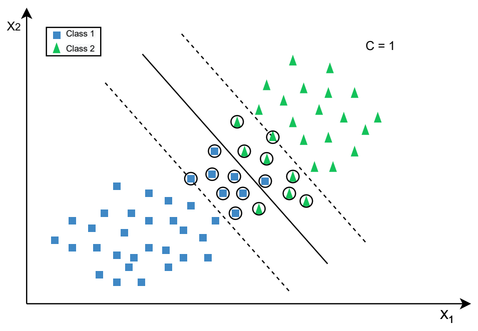
(b) Dataset with one outlier

Figure 24: Datasets with one outlier

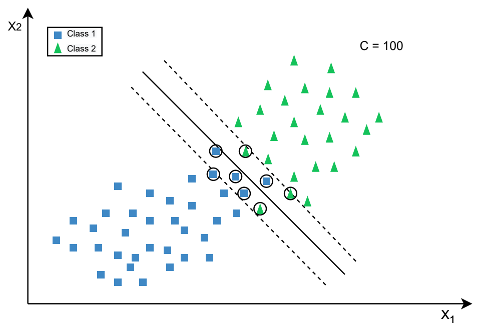
- **Soft Margin Classification**

  It is desirable to utilize a more flexible model to prevent these concerns. The idea is to find a balance between keeping the street as broad as possible while reducing margin violations (i.e., instances that end up in the middle of the street or even on the wrong side). This method is known as soft margin classification.

  The balance between keeping all the instances outside the street and on the right side is controlled by a hyperparameter called C.



(a) Dataset with large margin



(b) Dataset with few margin violations

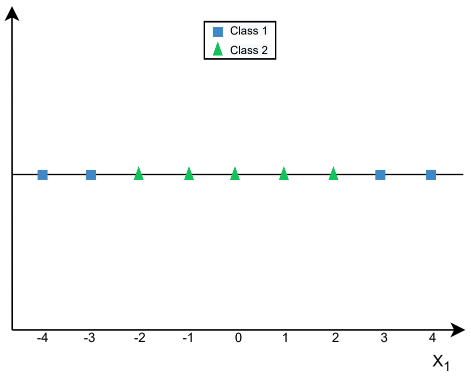
Figure 25: Dataset with large margin vs fewer margin violations

In figure 25a, it is possible to see that the classifier has a very wide margin, thus allowing for some margin violations (small hyperparameter C). In contrast to figure 25a, figure 25b has a very small margin and therefore contains few margin violations (large C hyperparameter).

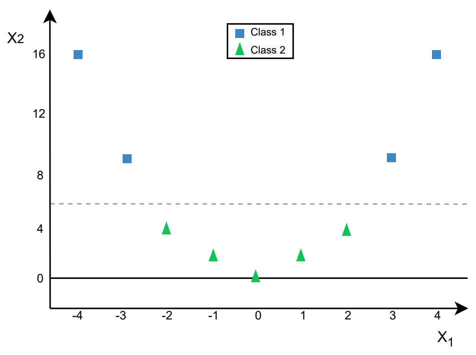
- **NonLinear SVM Classification**

  An SVM model works correctly in the presence of a linearly separable dataset. However, it is not always possible to work with such dataset. One way to get around this problem is to add more features to the dataset in question, such as polynomial features. These changes, in some cases, can lead to the presence of a linearly separable dataset.

In figure 26 it is possible to see that applying a polynomial transformation to a non-linearly separable dataset can sometimes yield a linearly separable dataset. The transformation function (mapping function) used is x^2, so the final result is a curve.



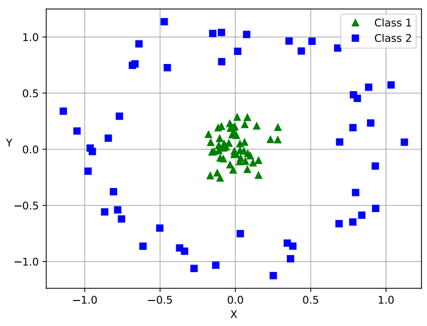
(a) Non-linearly separable dataset



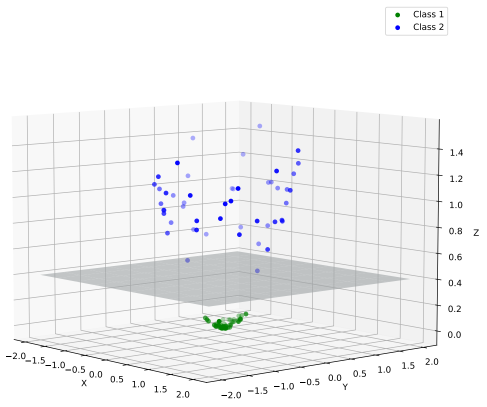
(b) Linearly separable dataset, due to new features (Polynomial features)

Figure 26: Adding features to make a dataset linearly separable

Figure 27a shows another non-linearly separable dataset. This dataset can become a linearly separable dataset if another dimension ($z = x^2 + y^2$) is added. The resulted dataset is shown in figure 27b.



(a) Non-linearly separable dataset

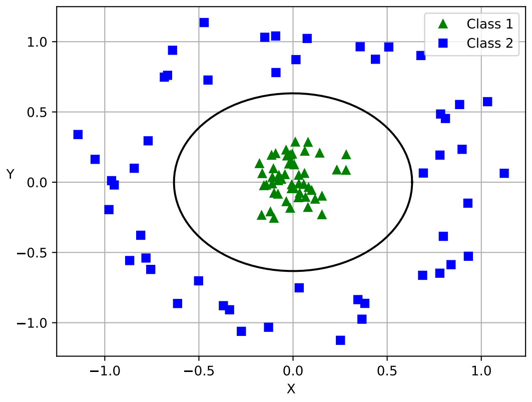


(b) Linearly separable dataset, due to new features (Polynomial features)

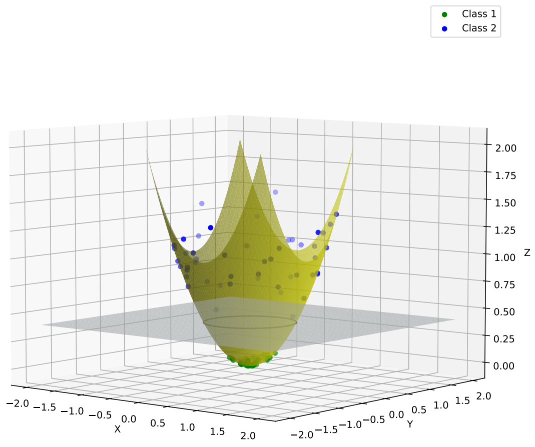
Figure 27: Adding features to make a dataset linearly separable

If we project the plane that separates the data of the two classes, in the original space, we will obtain the circle represented in the figure 28a.

Figure 28b, on the other hand, shows the mapping function, responsible for transforming the original dataset (figure 27a) into a linearly separable dataset (capable of being split by a plane)



(a) Dataset separated by the circle



(b) Linearly separable dataset, with the mapping function representation

Figure 28: Polynomial features

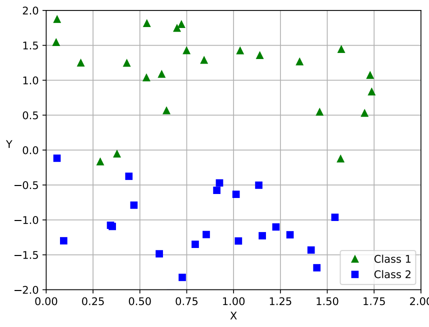
SVMs Classifiers technical details

This subsection is intended to explain the principle of how SVM's classifiers work. It starts by clarifying the importance of a decision function in an SVM algorithm, as well as what is the training objective in the developing of an SVM classifier. Finally, still within this section, it is possible to understand what are the mathematical methods involved in the development of an SVM model (in this case an SVM classifier)

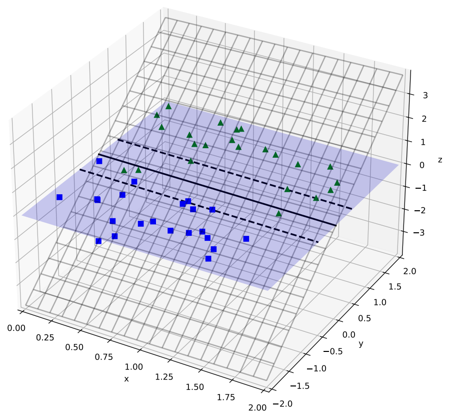
- **Decision Function**

  The linear SVM classifier model predicts the class of a new instance by simply computing the decision function. If the result of the decision function is positive, it means that the new instance belongs to a class, otherwise it belongs to another class. In other words, the decision function is responsible for classifying the new instance.

Figure 29 shows an example of a decision function in a random dataset, where it is possible to see the positions where the decision function is equal to -1 or 1. These positions are represented by the dashed lines in figure 29b.



(a) Random dataset

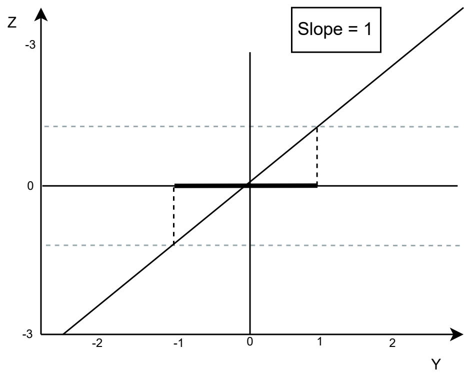


(b) Decision function representation

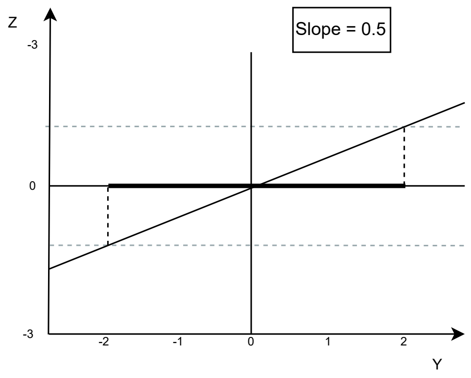
Figure 29: Decision function implementation example

- **Training Objective**

  The main objective when training an SVM model is to minimize the slope of the decision function. For example, figure 30a shows that the higher the slope, the smaller the margin responsible for separating the two data classes in the dataset, leading to an algorithm (classifier) that is not very robust, since the zone separating the two data classes is too small.



(a) Slope = 1



(b) Slope = 0.5

Figure 30: Decision function of figure 29b in 2D

In addition to minimizing the slope of the decision function, it is also necessary to take into account margin violations (i.e., all instances must be off-street and on the correct side of the margin). In the case of hard margin classification, no margin violation is allowed, while in the soft margin classification, some violations are allowed.

- **Kernel Trick**

  Due to the fact that most of the datasets collected from real scenarios correspond to non-linearly separable datasets. It may occasionally be necessary to execute operations with the higher dimensional vectors in the modified feature space when training an SVM classifier. In real applications, there may be several features in the data, and applying transformations that include numerous polynomial combinations of these features will result in expensive and unfeasible computational costs. A solution for this problem is provided by the kernel trick.

  The "trick" can be explained by looking at equation 17. This equation shows a couple of two-dimensional vectors, a and b, where a 2nd degree polynomial mapping function was applied together with the computation of the dot product between the transformed vectors.

$$\begin{aligned}\phi(a)^T \cdot \phi(b) &= \begin{pmatrix} a_1^2 \\ \sqrt{a_1 a_2} \\ a_2^2 \end{pmatrix} \cdot \begin{pmatrix} b_1^2 \\ \sqrt{b_1 b_2} \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2 \\ &= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)^2 = (a^T b)^2\end{aligned} \tag{17}$$

  As can be seen in equation 17, the dot product of the transformed vectors is equal to the square of the dot product of the original vectors. This equation (17) represents the kernel trick for a 2nd degree polynomial mapping. A benefit of the kernel trick is that the decision function, mentioned earlier, whose purpose is optimizing it to fit the higher dimension decision boundary, only includes the dot product of the transformed features vectors. Therefore, it is possible to substitute the dot product terms with the kernel function and does not even use the mapping function ($\theta(x)$). [35]

3.3.2 Decision Tree Classifier

Decision trees are adaptable Machine Learning algorithms that can carry out classification and regression tasks, as well as multioutput tasks, just as SVMs. They are powerful algorithms that can fit complex datasets. The fundamental elements of Random Forests, one of the most effective Machine Learning algorithms accessible today, are decision trees. In this subsection, it will be explained how decision trees make predictions, and how they are trained.

- **Visualizing**

  For reasons of simplicity, the following code was chosen to build a decision tree and to see or understand how a decision tree works and makes predictions. All this code and the respective explanation of it was based on the Aurélien Géron book, "Hands on Machine Learning with Scikit-learn and TensorFlow".

Listing 1: Decision Tree Classifier-Scikit

```python
iris = load_iris()
X = iris.data[:, 2:] # petal lenght and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X,y)
```

The code in listing 1 trains a decision tree on the iris dataset. This dataset is available in the Scikit-learn library. The features used to train the model are the petal length and the petal width. After the model training (*DecisionTreeClassifier* function), it is possible to create an image that shows how the model makes predictions. This figure is generated through the *export_graphviz* module of Scikit-learn library. Figure 31 represents the result from the model trained, and as can be seen, the tree has three nodes, since it was specified in the *DecisonTreeClassifier* function that the *max_depth* parameter is equal to two. The reason the number of nodes is three and not two comes from the fact that the first node is number zero [15].

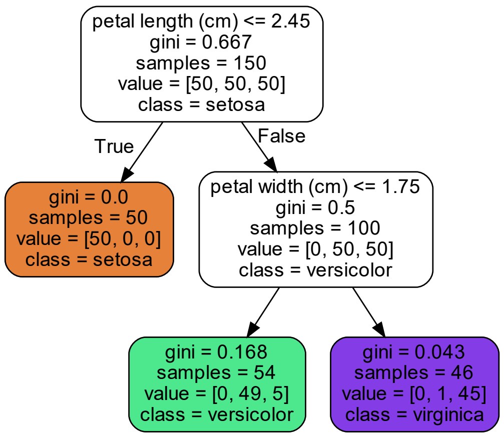


Figure 31: Iris Decision Tree

- **Making Predictions**

  Following the tree depicted in figure 31, it is possible to check that there are three different types of iris flowers (setosa, versicolor, and virginica). In the classification process of an iris flower, it is necessary analyzing the root node (depth 0, at the top). This node asks if the new iris flowers have a petal length equal to or lower than 2.45 cm. If it has, the flower is classified as setosa. If it does not, the model will ask another question. This last question will decide if the new instance is a versicolor (petal width ≤ 1.75) or a virginica (petal width > 1.75).

Each node has some attributes, such as the *samples* attribute indicating how many training instances are applied to the respective node. For example, 54 of the 100 training cases (depth 1, right) had petal widths lower than 1.75 cm. These 100 training instances have petal lengths of more than 2.45 cm (depth 2, left).

Another attribute in some nodes is the *value* attribute, which tells how many training instances of each class the respective node applies (the bottom-left node applies to 0 iris-setosa, 49 iris-versicolor, and 5 iris-virginica). The last attribute presented in some nodes is the *gini* attribute, which measures the impurity of the node. A node is "pure", if its *gini* attribute is equal to 0 (all the training instances that the node applies to belong to the same class, depth 1 left node). Equation 18 shows how to compute the *gini* score G_i of the i^{th} node.

$$G_i = 1 - \sum_{k=1}^{n} (p_{i,k})^2 \tag{18}$$

where, p$_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node. For example, the depth-2 left node has a *gini* score of $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$ [15].

- **The CART Training algorithm**

  Scikit-learn uses the "Classification And Regression Tree" (CART) algorithm to train decision trees. The algorithm first splits the training set into two subsets using a single feature k and a threshold t_k. The choice of the k, t$_k$ pair is done through the cost function that the algorithm is trying to minimize during the training process. The cost function is represented in the equation 19, where G$_{left/right}$ measures the impurity of the left/right subset, and m$_{left/right}$ represents the number of instances in the left/right subsets.

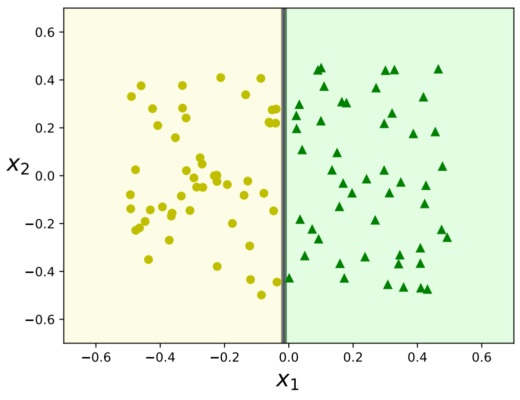
$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \tag{19}$$

  After splitting the dataset into two subsets, the algorithm splits the resulting subsets using the process to divide the original dataset. The algorithm does this recursively. It will stop when it reaches the maximum depth (defined by the *max_depth* hyperparameter) or if it is not possible to find new subsets with lower impurity [15].

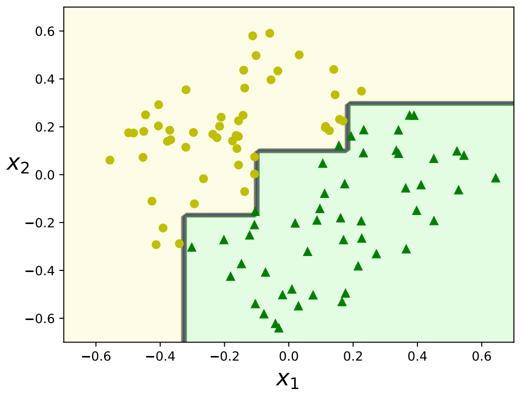
- **Instability**

  Decisions Trees have some restrictions due to their preference for orthogonal decision boundaries (all splits are perpendicular to an axis). As a result, decision trees are susceptible to changes in the training set.

Figure 32 shows an example of a rotation change in the training set used to train a decision tree classifier, and as can be seen, the model becomes more complex just by a slice change in the orientation of the training set.



(a) Training set without rotation



(b) Training set with rotation

Figure 32: Sensitivity to training set rotation (Figure based on [15])

A decision tree model can be different if, for example, some instances are removed from the training set. This modification will imply another behavior in the training process, leading to a new and different decision tree model [15].

3.3.3 Ensemble Learning and Random Forests

A group of predictors is called an ensemble, and the technique that aggregates the predictions of an ensemble is named Ensemble Learning. A classifier group prediction is better than the prediction of the best classifier in the group. For example, figure 33 represents four different types of classifiers (Two SVM Classifiers and two Decision Tree Classifiers), each one with about 80% of accuracy. The selection of the most voted prediction will lead to a creation of a better classifier. In other words, a classifier with higher accuracy.

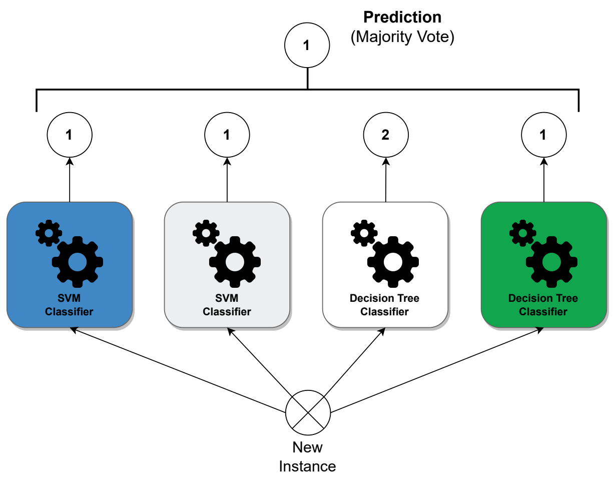


Figure 33: Ensemble Learning technique (Figure based on [15])

- **Bagging and Pasting**

  Utilizing a wide range of training techniques is one method for obtaining a diversified group of classifiers. Another way to get a diverse set of classifiers can be by using the same algorithm for every predictor, but the training process is done in different subsets of the training set. In this case, if the sampling process is executed with the subset replacement, it is called bagging, and if the sampling is performed without the subset replacement, it is named pasting. The difference between bagging and pasting is that bagging allows training instances to be sampled several times for one predictor, unlike pasting. Figure 34 shows an example of bagging training set sampling and training [15].

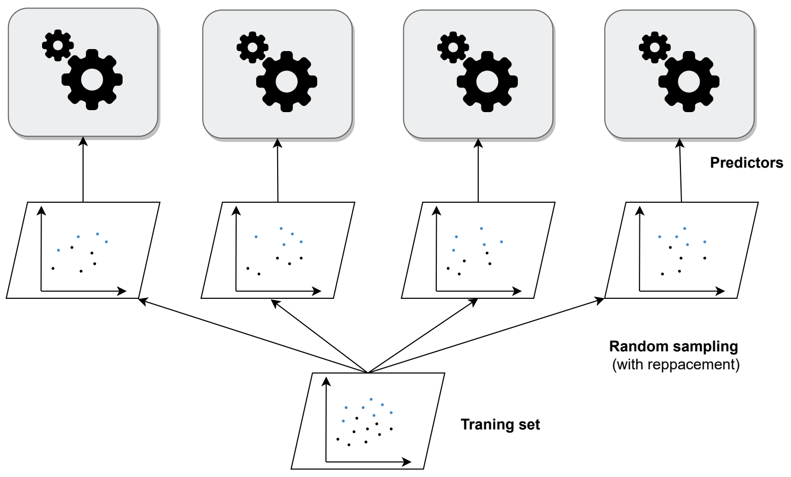


Figure 34: Bagging training set sampling and training (Figure based on [15])

- **Random Forest**
  The Decision Tree classifier group, for instance, could be trained using various random subsets of the training set. It is required to get the predictions of each tree before predicting the class that would receive the most votes. A random forest is a collection of decision trees, and despite its apparent simplicity, it is one of the most effective Machine Learning algorithms nowadays. A random forest is generally trained via the bagging method [15].

3.3.4 Artificial Neural Network (ANN)

An artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain, artificial neural networks also have neurons that are linked to each other in various layers of the networks. Figure 35 represents the diagram of a biological neuron.

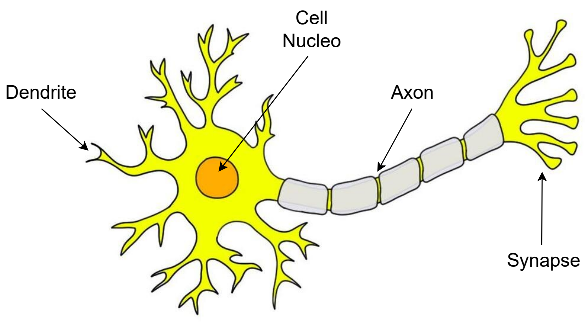


Figure 35: Diagram of biological neuron (Figure based on [17])

Figure 36 shows a typical Artificial Neural Network with just one neuron, and as can be seen, compared to figure 35, the inputs are represented by dendrites, the nodes by cell nucleus, the weights by synapses, and the outputs by axons.

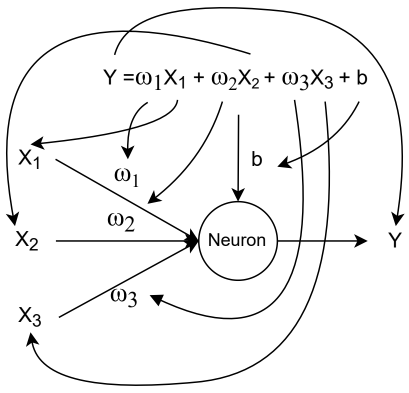


Figure 36: Diagram of artificial neuron (Figure based on [13])

Figure 36 also represents the mathematical form of an artificial neuron. In this representation, the neuron takes the inputs and computes the weighted sum of those, including a bias term (b). This example shows an ANN with just one neuron, but there are others with more than one. This number is proportional to the complexity of the problem that is trying to be solved by the ANN [18]. Just like any ML model, ANNs also need to be trained. So the ANN training process lies in finding the best weights and bias terms that fit the problem. The mathematical representation of a neuron, given by the previous figure, only works for modeling linear relationships between inputs and outputs. So in the presence of non-linear data, the neurons of an artificial neural network should use an activation function to calculate their outputs. There are different types of activation functions, but the most popular and most used activation function is the sigmoid function. The mathematical formula of this function is represented in equation 20.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{20}$$

In each neuron, the activation function accepts the sum of the product between the inputs and the weights, and then returns a single value that is able to non-linearly map between the inputs and the output(s) [13].

In an artificial neural network with more than one neuron, the neurons are arranged in a sequence of layers. Figure 37 shows an ANN with one input layer, two hidden layers, and one output layer. The input layer accepts the inputs in several formats provided by the programmer, unlike the hidden layers, which are responsible for performing all the calculations to find hidden features and patterns in the input values. The last layer is responsible for showing the result of the neural network [18].

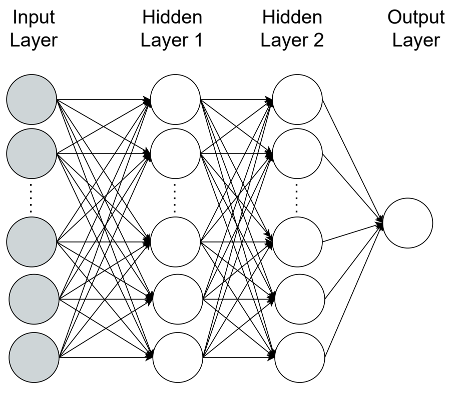


Figure 37: Artificial Neural Network with two hidden layers

As mentioned earlier, the training operation consists in finding the weights and the bias terms that best fit the data and reduce the error (the difference between the original data and the prediction data). This process requires the gradient descent algorithm application to the equation that relates the error and the ANN parameters.

For example, in the case of an ANN with one input and one output (Figure 38), the equation that associates the prediction error and the ANN parameters (weights) is represented in equation 21 [13].

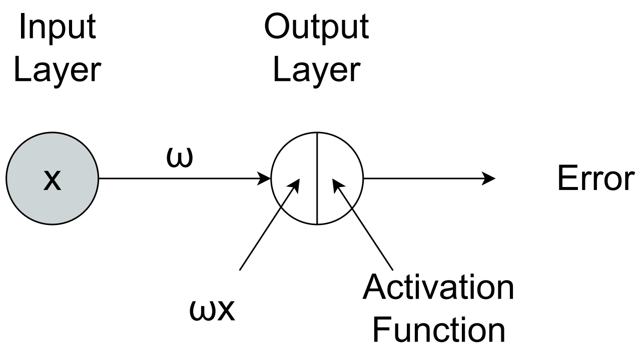


Figure 38: Artificial Neural Network with 1 input and 1 output

$$error = (predicted - target)^2 = \left(\frac{1}{1 + e^{-wx}} - target \right)^2 \tag{21}$$

Applying the gradient descent algorithm to this equation allows finding the value of ω that best fits the data used for training. The examples mentioned before correspond to Feed-Forward ANN. There are various types of ANN, the most famous and used types of ANN are listed below:

- **Feedback ANN**
  The output of this type of ANN is feedback into the network to achieve the best internal results. Feedback neural networks allow signal transmission in both ways, and the complexity of these neural networks can grow faster. Neural networks with feedback are dynamic. When such a network reaches an equilibrium point, the “state” will no longer change until the input changes. The architecture of a feedback neural network is also known as interactive or recurrent [29].

- **Feed-Forward ANN**
  In a feed forward Neural Network, signals only flow one way, from input to output. No loops or feedback exist. In such networks, the output of one layer does not affect another layer below it. Simple networks that link inputs and outputs are feed-forward neural networks [29].

3.3.5 Generalized Additive Models (GAM)

Generalized Additive Models (GAMs) are a kind of regression model used to model the relationship between a dependent variable and one or more independent variables. Unlike traditional regression models, which assume that the relationship between the dependent and independent variables is linear, GAMs allow for non-linear relationships between the variables, which makes them more flexible and able to capture complex patterns in the data. GAMs are based on additive modeling, where the response variable is modeled as a sum of multiple functions, each of which is a function of one of the independent variables (figure 39).

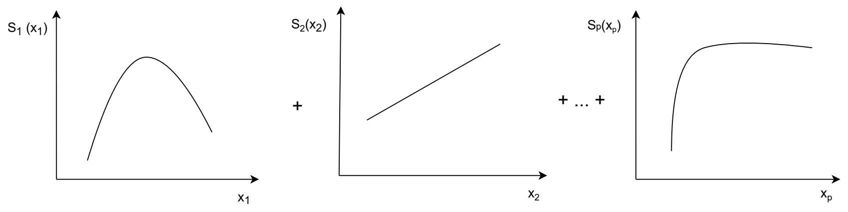


Figure 39: GAM structure example (Figure based on [21])

The GAM structure can be written as:

$$g(E(Y)) = \alpha + S_1(x_1) + ... + S_p(x_p) \tag{22}$$

where Y is the dependent variable (i.e., the predicted value), $E(Y)$ the expected value, and $g(E(Y))$ denotes the link function that links the expected value to the predictor variables $x_1, ..., x_p$. The terms $S_1(x_1), ..., S_p(x_p)$ indicate smooth, *nonparametric* functions. In contrast to *parametric* functions, which are defined by a typically small set of parameters, *nonparametric* function's shape is entirely determined by the data. This will enable a more flexible estimation of the underlying predictive patterns without requiring prior knowledge of the pattern's appearance [21].

GAM is a powerful algorithm because of its interpretability, flexibility, and regularization.

- **Interpretability**
  In an additive regression model, the interpretation of the marginal impact of a single variable (the partial derivative) don't depend on the values of the other variables in the model. Therefore, it is possible to make simple statements about the predictive variables, such as in figure 39 the expected value of Y increases linearly as x_2, holding everything else constant [21].

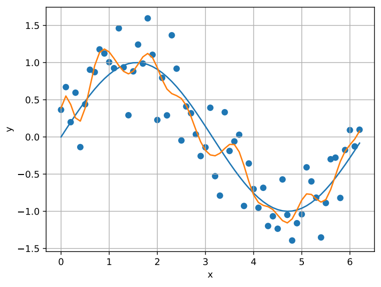
- **Flexibility**
  GAM can detect common nonlinear patterns that a classic linear model would miss. Therefore, fitting *parametric* regression models with nonlinear data will result in nonlinear effects captured through binning or polynomials, and a gawky model with many correlated terms. In addition, selecting the best model involves developing several models, followed by a search algorithm to choose the best one. In GAM, this type of problem doesn't exist, since the predictor functions are automatically derived during model estimation. It is not necessary to know upfront what type of functions best fits the data [21].

- **Regularization**
  GAM has the capacity of controlling the smoothness of the predictor functions to prevent overfitting. As a way to further explain regularization in the GAM algorithm, the following example is presented. This example consists of a two-dimensional dataset (x, y) in which the relationship between these

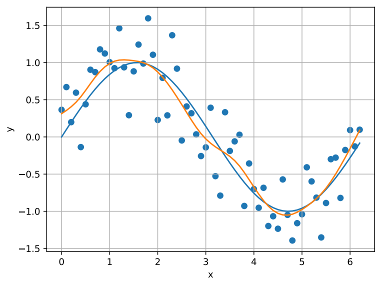
two variables follows a sine function (with random error). The purpose of this example is to predict y, giving, x by fitting the following model (equation 23).

$$y = s_{\lambda}(x) + e \tag{23}$$

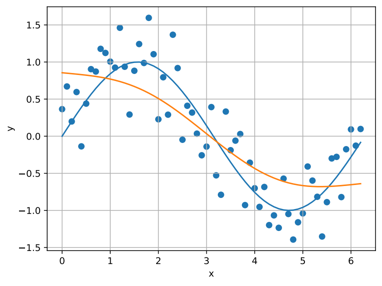
where $s_{\lambda}(x)$ is the smooth function, and the level of smoothness is determined by the smoothing parameter(i.e., λ). The higher the value of this parameter, the smoother the predicted function. Figures 40a, 40b, and 40c, represent three fitted models to the mentioned dataset, each with a different smoothness parameter. In the figures, the orange line represents the final model, while the blue line corresponds to the sine function, which the dataset follows. Finally, the blue dots indicate the constituent data of the dataset.



(a) $\lambda = 0$



(b) $\lambda = 0.6$



(c) $\lambda = 200$

Figure 40: Fitting a model with three different λ values (Figure based on [21])

As it is possible to check, the model with $\lambda = 0$ provides the best fit for the data, but with a resulting curve very prone to outliers. Oppositely, the model with $\lambda = 200$ presents a function that diverges significantly from the data. The best solution in this situation is the one in figure 40b, which seems to be the intermediate model between the other two.

For more information about the working principle of the GAM algorithm, you can refer to [21].

3.3.6 Lattice-based ML models

Lattice is an interpolated look-up table that can approximate arbitrary input-output relationships in the data. It overlays a regular grid onto the input space and learns values for the output in the vertices of the grid. The following figure, 41, shows an example of two lattices. The one on the left represents an example of a function with two input features and four parameters (θ), which are the function's values at the corners of the input space. The rest of the function's values are interpolated from these parameters.

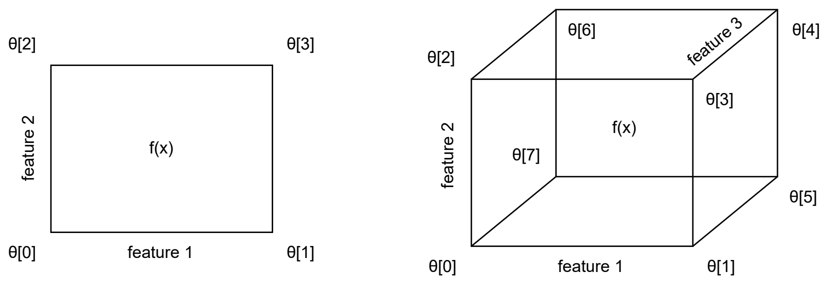


Figure 41: Lattices example (Figure based on [30])

In figure 41, the example on the right show an example of a lattice with three input features, and as can be seen, it has eight parameters ($2^{features\_number}$). To implement lattice-based machine learning models, TensorFlow Lattice library, provide some tools. (see more in [30])

Lattice models are a kind of interpretable machine learning model that can make accurate predictions while also providing insight into the relationships between the input features and the output. Lattice models consist of interconnected nodes, or lattice points, representing the input features and the output. The connections between the lattice points are called edges, and the strength of these connections is represented by the weights of the edges. These weights are learned during the training process and determine how much influence each input feature has on the output. To train a lattice model, a dataset is provided to the model along with the desired output. The model then adjusts the weights of the edges to minimize the difference between the predicted output and the desired output. This process is repeated for multiple epochs or passes through the dataset until the model reaches a satisfactory level of accuracy.One of the benefits of lattice models is their interpretability. Since the weights of the edges represent the strength of the relationship between the input features and the output, it is possible to analyze these weights to understand the relative importance of each attribute in predicting the final result. This can be useful for understanding the relationships between the features and the output, and identifying features that are not important for the prediction process (For more information about the working principle of the Lattice-based ML models, you can refer to [30]).

Chapter 4

System Specifications

4.1 System Overview

For the application proposed in this dissertation, the system overview is shown in figure 42. In this system, it is possible to check the data collection that will constitute the training dataset responsible for the overall machine learning models. These models, represented in figure 42 by the box whose legend is (classifier/regressor), are responsible for classifying or predicting new gas instances.

To collect the training data and the new data to be evaluated, the BME688 sensor was chosen. This sensor is present in the BME688 development kit, available from Bosch Sensortec.

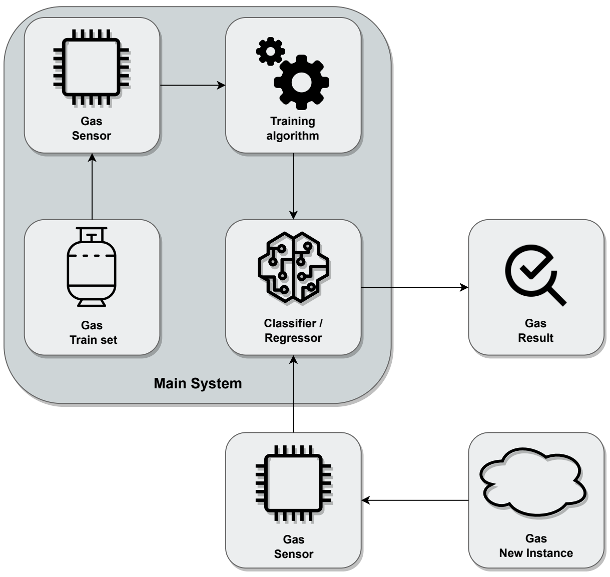


Figure 42: System overview

4.2 Requirements

System requirements are all the requirements that describe the functions that the system as a whole must fulfill. It is important to respect all the system requirements during its development so that it responds to the problem as intended.

4.2.1 Functional Requirements

- Collect data correctly
- Train robust machine learning models
- Classify a given instance or predict the concentration of a gas in the newly collected gas instance.
- Selection of the best machine learning model for a specific dataset based on selection criteria

4.2.2 Non-Functional Requirements

- Ensure a small classification/prediction error
- Fast response from the machine learning models

4.3 Hardware

The BME688 development kit from Bosch Sensortec enables the testing and creation of use cases for temperature, barometric pressure, humidity, and gas sensing. As can be seen in figure 43, this kit consists of an Adafruit HUZZAH32 feather board and a BME688 dev-kit board.

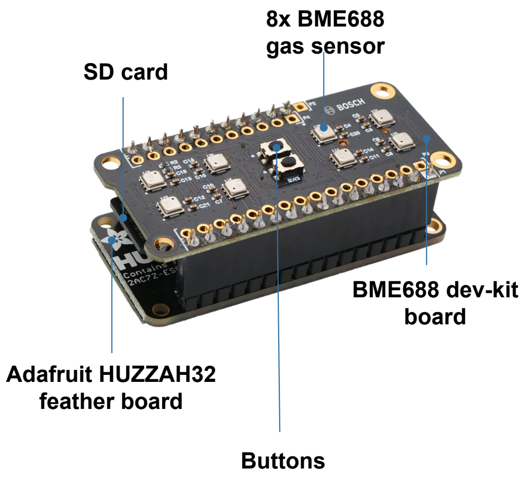


Figure 43: BME688 development kit

The Adafruit HUZZAH32 feather board presented in figure 44 is responsible for processing the data collected from the BME688 dev-kit board. This feather contains a dual-core ESP32 chip, 4 MB of SPI Flash, a tuned antenna, and other features. The following list shows some of the most significant specifications of this board.

Adafruit HUZZAH32 feather specifications:

- 240 MHz dual-core Tensilica LX6 microcontroller with 600 DMIPS
- Integrated 520 KB SRAM
- Integrated 802.11b/g/n HT40 Wi-Fi transceiver, baseband, stack and LWIP
- Integrated dual mode Bluetooth (classic and BLE)
- 4 MByte flash include in the WROOM32 module
- 3 x SPI Hosts
- 2 x I^{2}C Hosts
- PWM/timer input/output available on every pin

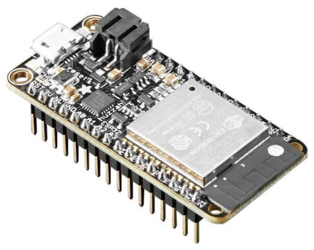


Figure 44: Adafruit HUZZAH32 feather board

The other board that makes up the BME688 development kit is the BME688 dev-kit board, which is responsible for taking temperature, pressure, humidity, and gas readings. This board features eight BME688 sensors, an RTC, and an SD Card slot.

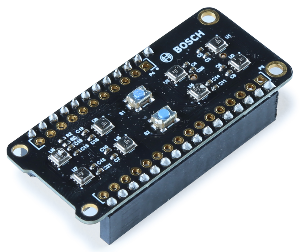


Figure 45: BME688 dev-kit board

BME688 development kit has 8 BME688 gas sensors, all responsible for collecting gas data. The BME688 sensor can discriminate between various gas compositions by detecting and measuring their distinct electronic fingerprint. Figure 46 shows an BME688 sensor.

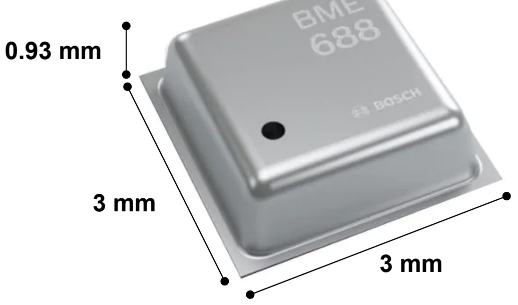


Figure 46: BME688 sensor

Another significant aspect of the BME688 development kit is that the Adafruit HUZZAH32 feather board, connected to the eight BME688 sensors, communicates with all the sensors by an SPI interface, sharing the same MISO and MOSI line. Because all sensors share the same resources, the BME688 dev-kit board has a device that allows the ESP32 (Adafruit HUZZAH32 feather board) to select the sensor that wants to communicate with, i.e., the microprocessor must choose which sensor wants to communicate, to avoid conflicts in communication and reduce the number of resources (in this case SPI hosts by the microcontroller). The device chosen by Bosch Sensortec to unpack such a function was the TCA6408A Low-Voltage 8-bit I^{2}C and SMBus I/O Expander. Its schematic is shown in the figure 47.

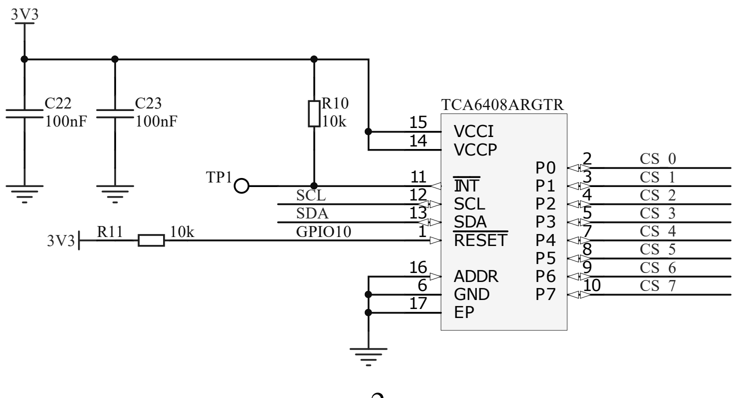


Figure 47: TCA6408A Low-Voltage 8-bit I^{2}C and SMBus I/O Expander Schematic

The output of the TCA6408A is connected to all eight sensors in the BME688 dev-kit, as shown in figure 47. Figure 48 shows the schematic of sensor number zero (CS_0 $\Rightarrow$ Chip Select 0).

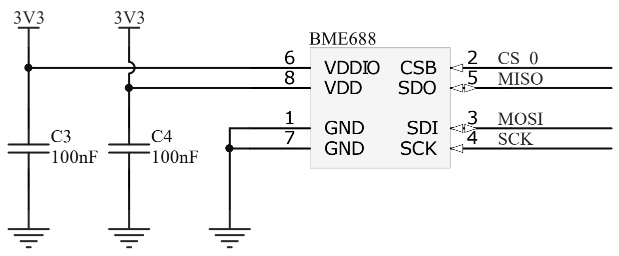


Figure 48: BME688 Schematic

According to Bosch Sensortec, the BME688 sensor contains a metal oxide layer. Operating this layer at different temperatures allows different gas concentrations with different sensitivities, creating unique fingerprints for each gas concentration. This sensor lets the definition of its heating profile through its internal registers, i.e., select the different temperatures at which the sensor will operate. Figure 49 shows a heater profile example divided into ten steps, each with a specific time duration [26].

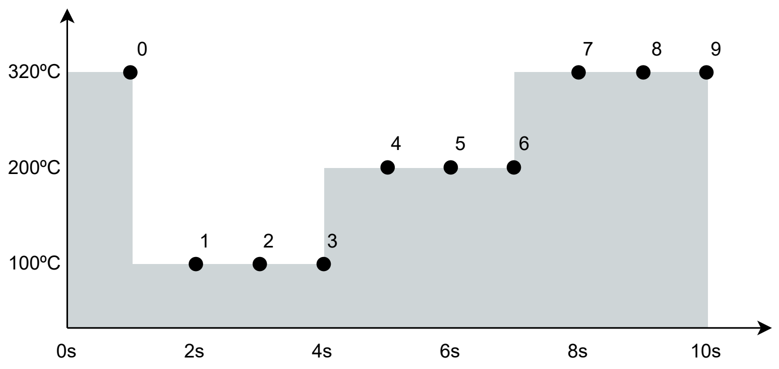


Figure 49: Heater profile example

The BME688 sensor supports low-level power modes, such as sleep, forced and parallel mode, each with a specific key feature. The key features of all operation modes are summarized in table 3.

Table 3: BME688 operation modes (from [26])

Operation	**Key feature**
Sleep Mode	• No measurements are performed • Minimal power consumption
Forced Mode	• Single TPHG cycle is performed • Sensor automatically returns to sleep mode afterwards • Gas sensor heater only operates during gas measurement
Parallel Mode	• Multiple TPHG cycle are performed • Sensor will not automatically return to sleep mode • Gas sensor heater operates in parallel to TPH measurement

A TPHG cycle, presented in the table above, corresponds to the execution of a heater profile, in which the sensor collects data on temperature, ambient pressure, humidity, and ambient gases. The BME688 sensor automatically enters sleep mode after a power-up procedure. Execution of mode switching commands is postponed until the end of the measurement period if the device is currently running one [26].

The BME688 dev-kit board is equipped with an external RTC, allowing the time logging of the data read by the gas sensors. Figure 50 represents the schematic of the external RTC, whose reference is PCF8523.

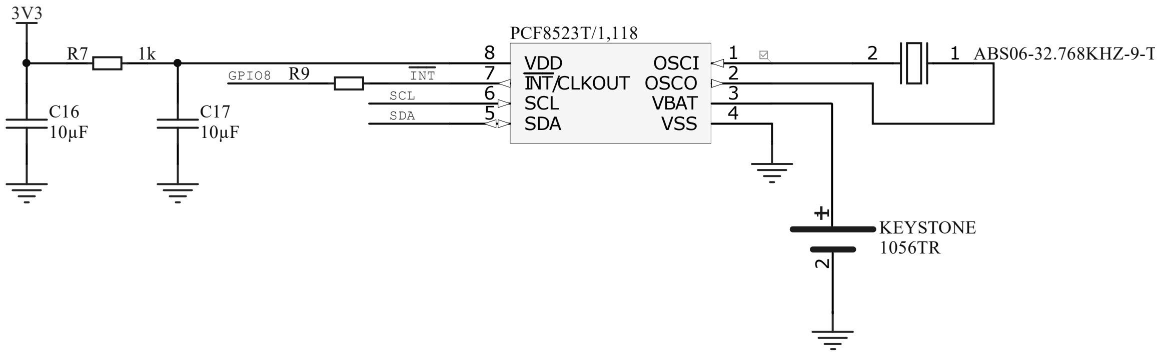


Figure 50: PCF8523 Real-Time Clock (RTC)

4.4 Software

4.4.1 Tools

- **Visual Studio Code**
  A streamlined code editor, Visual Studio Code, supports development activities like task execution and version management. It seeks to give developers the resources they require for a rapid cycle of code-build-debug [23].

- **Spyder**
  Spyder is a free and open source scientific environment created by and for scientists, engineers, and data analysts in Python. This tool has a distinctive combination of capabilities, including code editing, analysis, and debugging. This tool also allows advanced and detailed data analysis and visualization [9].

4.4.2 Software Tools Used

- **Scikit-learn**
  Unsupervised and supervised learning are both supported by the open source machine learning

package Scikit-learn. Additionally, it offers several tools for data preprocessing, model selection, model evaluation, and many other utilities [8].

- **TensorFlow**
  TensorFlow is an end-to-end platform for machine learning. It provides resources to speed up model building and create scalable ML solutions. TensorFlow offers multiple levels of abstraction, such as building and training models using the high-level Keras API [3].

- **NumPy**
  NumPy is a Python library that provides a multidimensional array object, and an assortment of functions for fast operations on arrays, including mathematical, logical, shape manipulation, sorting and much more [6].

- **Pandas**
  Pandas is a Python-based open-source data analysis and manipulation tool that is quick, strong, adaptable, and easy to use [7].

- **Matplotlib**
  Python's Matplotlib toolkit provides a complete tool for building static, animated, and interactive visualizations [5].

- **BME68x Driver**
  Bosch Sensortec's provide a sensor API to operate the BME688 gas sensor [27].

4.4.3 Software Architecture

The project proposed in this dissertation is divided into 3 phases: data collection process, training machine learning models based on the collected data, and execution of the trained models. For the data collection stage, the software architecture is represented in figure 51, and for the two other phases, the software architecture is shown in figure 52. The data collection process will be executed in the BME688 Development Kit, in contrast with the two others processes, which will be performed in a personal computer (capable to support the libraries in figure 52).

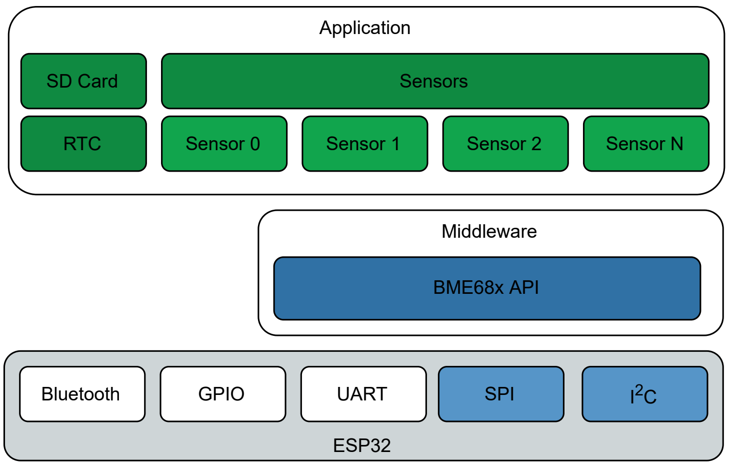


Figure 51: Software architecture for the data collection process

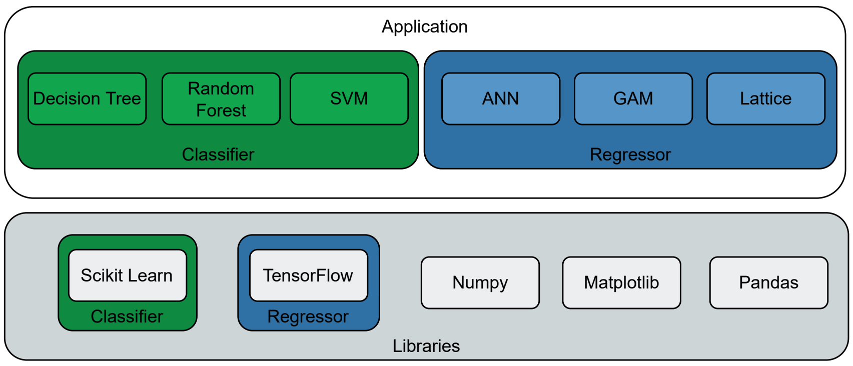


Figure 52: Software architecture for machine learning models

The software architecture, shown in figure 51, represents the organization of the data collection system. This organization includes all components, how they interact with each other and the environment in which they operate. The ESP32 layer has five modules. The SPI and I^{2}C modules enable, with the help of the BME68x API, the digital interface with the eight BME688 sensors in the application layer. This layer, beyond the sensors modules, has two other modules. One is for SD Card interface, and the other is for interfacing with the external RTC in the BME688 dev-kit board. The software architecture for the final two phases of the project is shown in figure 52. It has two layers. The library layer is used to help the ML model's implementation, and the application layer implements the respective ML models (three classifiers and three regressors). The Scikit-learn and TensorFlow modules are libraries used for classification and regression processes, respectively.

4.4.4 Software Class Diagrams

In this subsection, the class diagrams for the software of this dissertation will be described. The class diagram in figure 53 shows all the structures used in the data collection process. In this diagram, it is possible to see the bme68x_dev structure, responsible for representing the BME688 sensor. This structure is part of the API provided by BOSCH Sensortec that aims to simplify communication between the user and the sensor. This structure contains the attributes responsible for storing the addresses of the read and write functions for the BME688 sensor (bme68x_read_fptr_t read and bme68x_write_fptr_t write). In addition to this structure, the bme68x_heatr_conf and bme68x_data structures that are part of the BME68x API are delegated to configure the sensor's heater profile and store the sensor data, respectively.

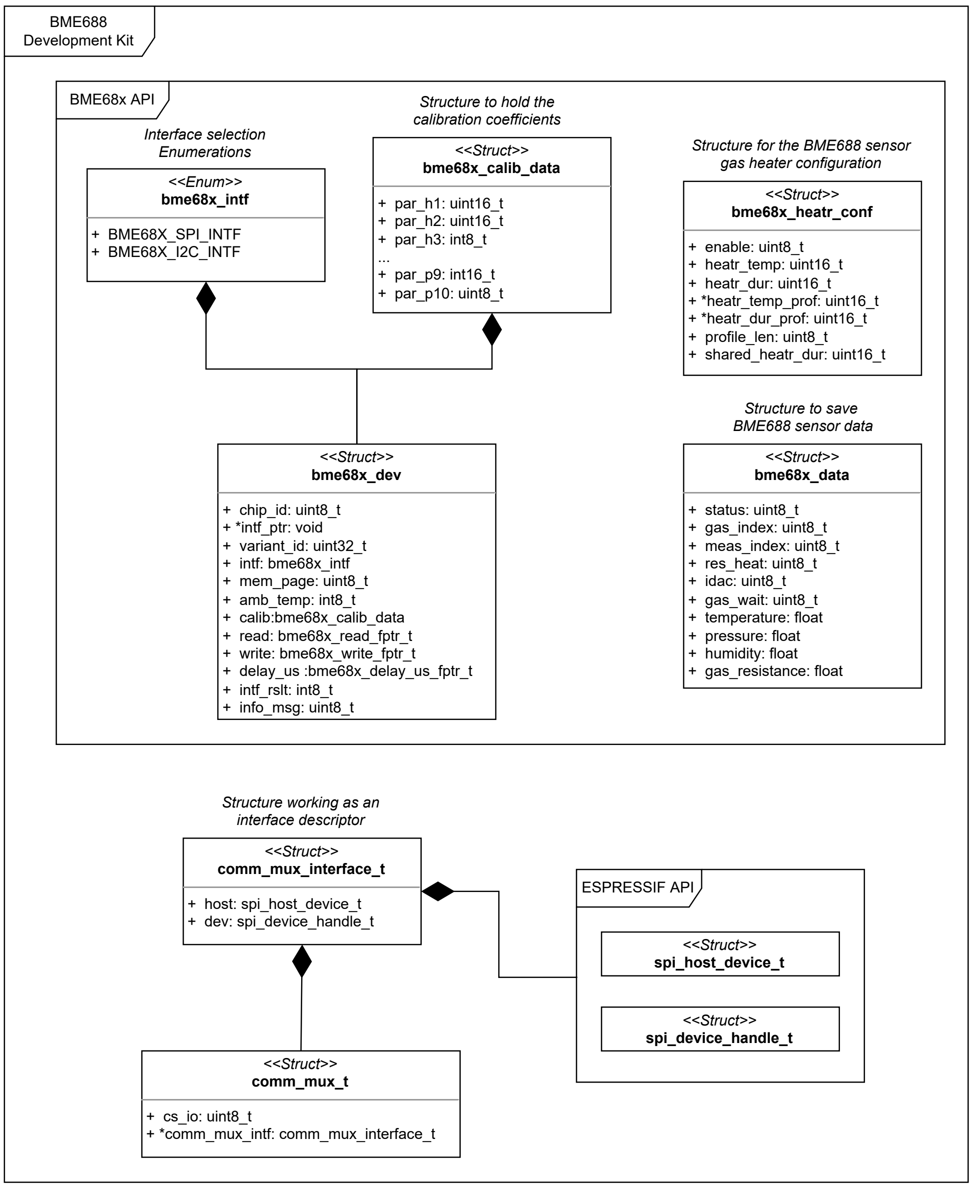


Figure 53: Class Diagram for the BME688 Development Kit data collection process

As mentioned earlier in the hardware subsection, the BME688 dev-kit board has an SMBus to control access to the various sensors presented in the kit. The structures represented in figure 53 with the name of comm_mux_intterface_t and comm_mux_t are responsible for establishing the communication between the SMBus, the ESP32 microcontroller, and the 8 BME688 sensors. The first structure (comm_mux_int-terface_t) will control the ESP32's SPI peripherals while working as masters, and the second structure (comm_mux_t), will command the communication process between each sensor in the board and the ESP32 microcontroller. This last one has the cs_io variable as an attribute that will take the sensor number as a value.

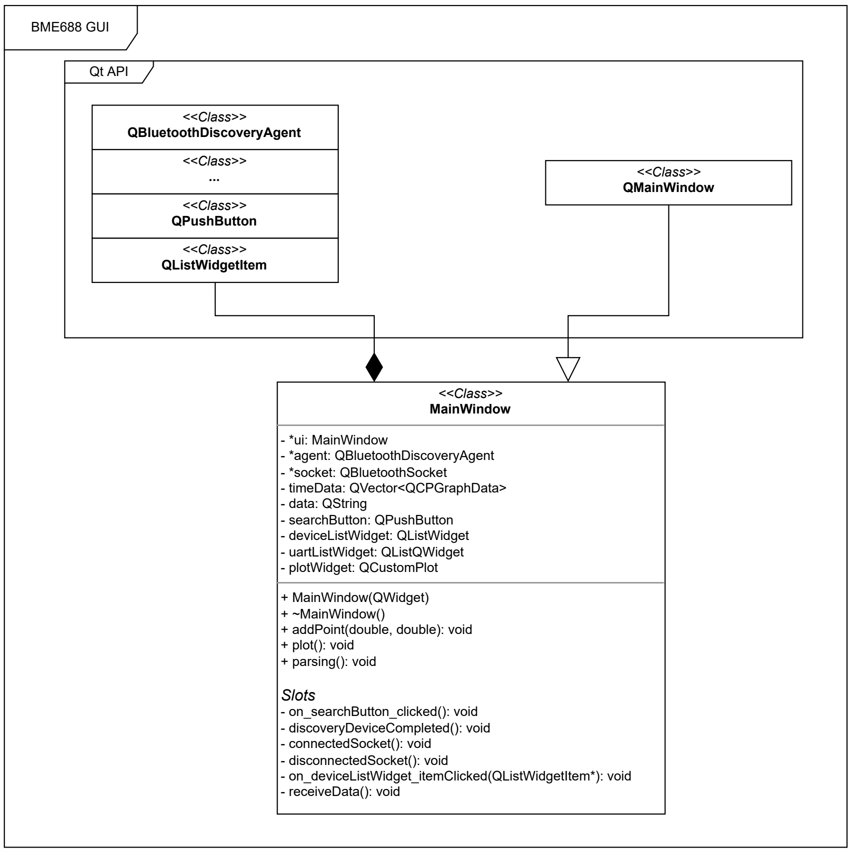

BME688 GUI
Qt API
<<Class>> QBluetoothDiscoveryAgent
<<Class>> ...
<<Class>> QPushButton
<<Class>> QListWidgetItem
<<Class>> QMainWindow
<<Class>> MainWindow
- *ui: MainWindow
- *agent: QBluetoothDiscoveryAgent
- *socket: QBluetoothSocket
- timeData: QVector<QCPGraphData>
- data: QString
- searchButton: QPushButton
- deviceListWidget: QListWidget
- uartListWidget: QListQWidget
- plotWidget: QCustomPlot
+ MainWindow(QWidget)
+ ~MainWindow()
+ addPoint(double, double): void
+ plot(): void
+ parsing(): void
Slots
- on_searchButton_clicked(): void
- discoveryDeviceCompleted(): void
- connectedSocket(): void
- disconnectedSocket(): void
- on_deviceListWidget_itemClicked(QListWidgetItem*): void
- receiveData(): void

Figure 54: Class Diagram for the BME688 Development Kit GUI

Since the data collection process will be done in a controlled environment to avoid gas leaks, access to the BME688 Development Kit (ESP32 microcontroller and the BME688 dev-kit board) will be constrained, making it necessary to develop an application that monitors the data reading process. This application was design to run in any personal computer that support Qt. The class diagram of this application is represented in figure 54. In this figure, it is possible to analyze the "MainWindow"class that contains the "agent", "socket", and "plotWidget"attributes, which will establish the Bluetooth connection with the BME688 Development Kit and display the read data to the user. The structures used in the ESP32 to create a Bluetooth host are not shown in figure 53 for simplicity reasons (see [11]).

4.4.5 Software Flowcharts

A flowchart is a diagram that describes a process, system, or computer algorithm. This type of diagram can be defined as a diagrammatic representation of an algorithm. The flowchart in figure 55 illustrates the algorithm flow responsible for the collection data process. This algorithm will run in the BME688 Development Kit, and because it owns two external buttons, one of which is used to control the data collection process. The Bluetooth host used to monitor the kit execution will work in the background, waiting for new connections (some computer) to send the data read by the sensors. If any device connects to this host, the program will keep saving the data into the SD Card. In other words, the BME688 Development Kit will save the sensor's data into the SD Card independently of a Bluetooth device connection.

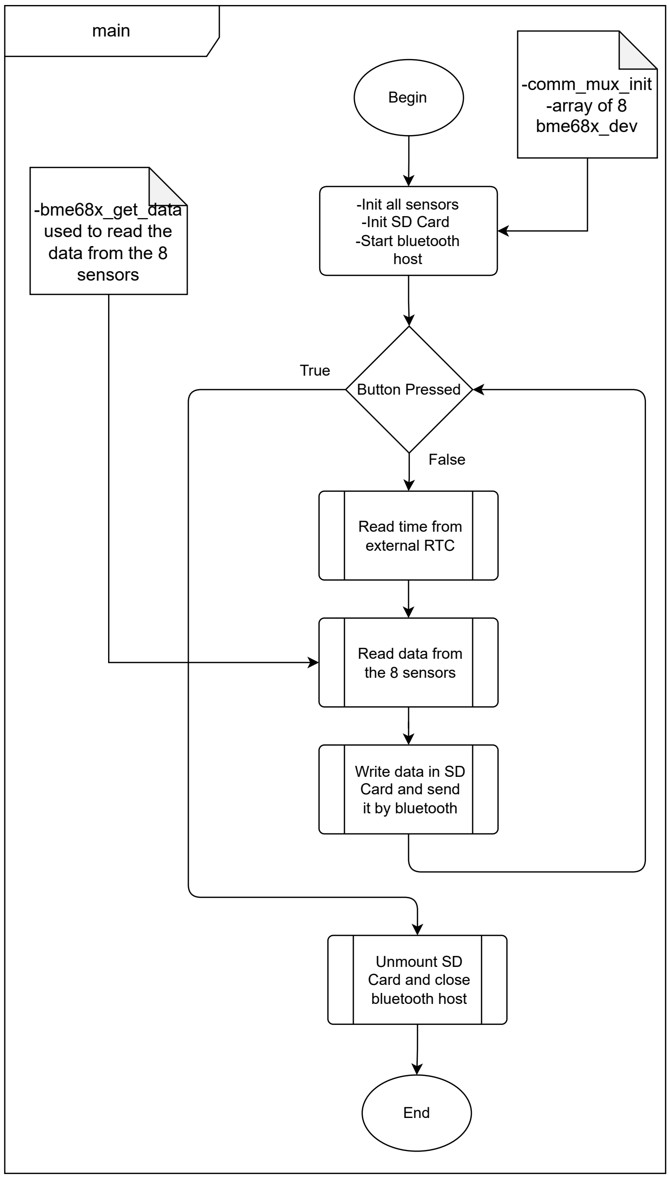


Figure 55: Software Flowchart for the data collection process

The comm_mux_init function represented in figure 56, and called at the beginning of the ”main function”above, is responsible for configuring the SMBus mentioned earlier, and the SPI bus (MISO, MOSI, and SCK) used in the communication between the sensors and the ESP32. The configuration process consists in putting all SMBus user peripherals as output pins, since they will work as the chip select for the 8 BME688 sensors in the board.

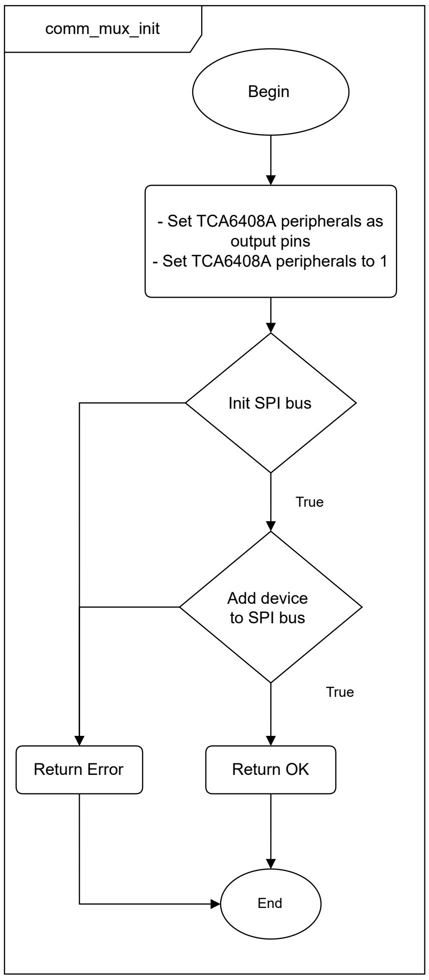
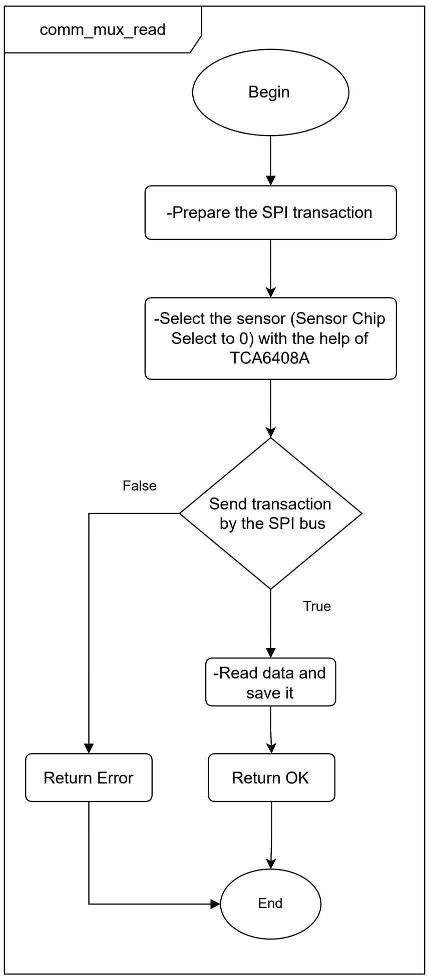
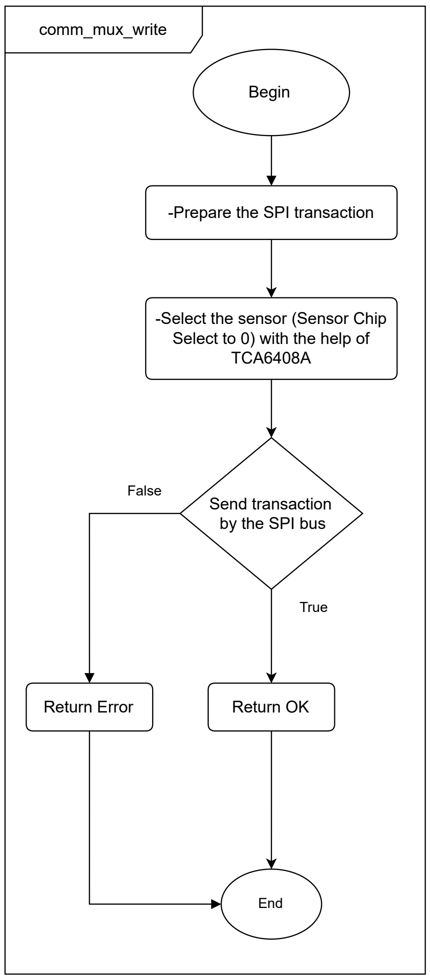


Figure 56: Software flowchart of comm_mux_init function

The comm_mux_read and comm_mux_write functions (represented in figure 57) are responsible for writing and reading information to a specific BME688 sensor. The address of these two functions will be assigned to the ”read”and ”write”attributes of the bme68x_dev structure (figure 53) to provide easy communication with the sensors. All these functions make part of the comm_mux module, which was developed to connect the ”main function”with the BME68x API.


comm_mux_read
Begin
-Prepare the SPI transaction
-Select the sensor (Sensor Chip Select to 0) with the help of TCA6408A
Send transaction by the SPI bus
False
True
-Read data and save it
Return Error
Return OK
End

(a) Software flowchart of comm_mux_read function


comm_mux_write
Begin
-Prepare the SPI transaction
-Select the sensor (Sensor Chip Select to 0) with the help of TCA6408A
Send transaction by the SPI bus
False
True
Return Error
Return OK
End

(b) Software flowchart of comm_mux_write function

Figure 57: Software flowchart of comm_mux module used to make the interface between the ESP32, SMBus and the sensors

To better explain the process of reading the internal registers of the BME688 sensor, the following flowcharts are presented. These flowcharts represent the workflow of the functions bme68x_get_data and bme68x_get_regs provided by BOSCH Sensortec. The bme68x_get_regs reads the registers from a given sensor via the comm_mux_read procedure passed as an argument through the bme68x_dev structure. On the other hand, the bme68x_get_data task will read the sensor's register fields, taking into account its operating mode (Parallel mode or forced mode). For more information about the BME688 sensor's register fields, which contain all the information read by the sensor, the sensor's datasheet provides more detailed information about these fields [26].

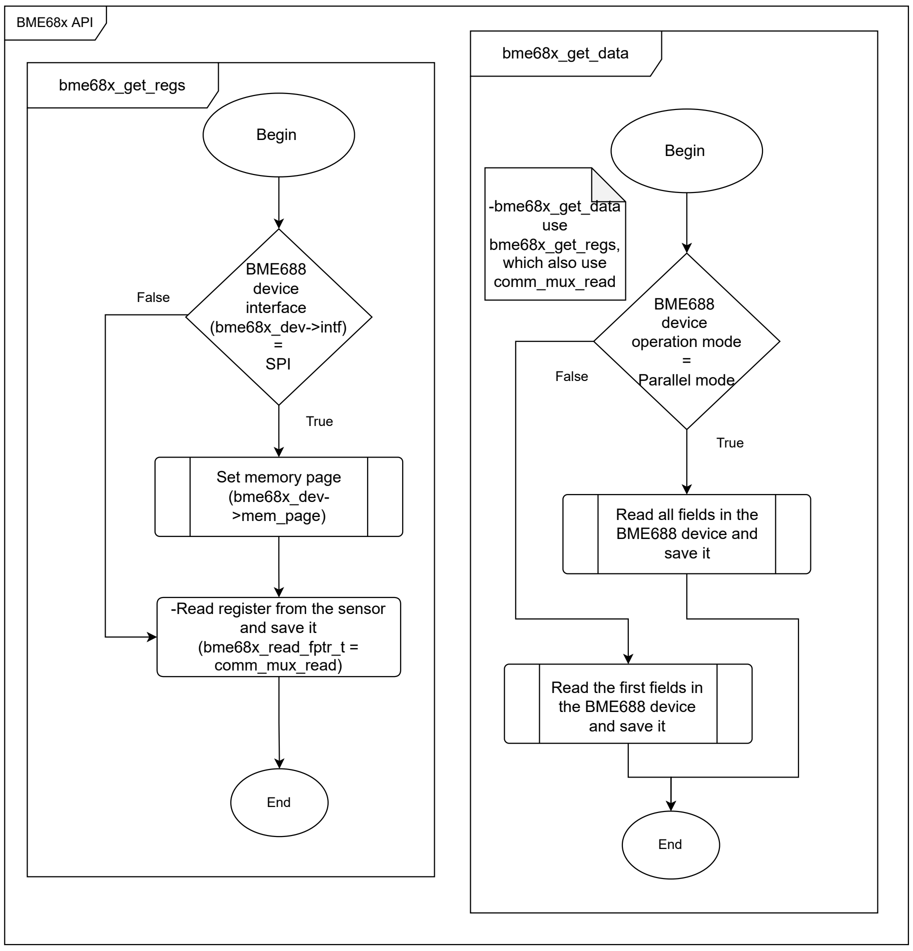
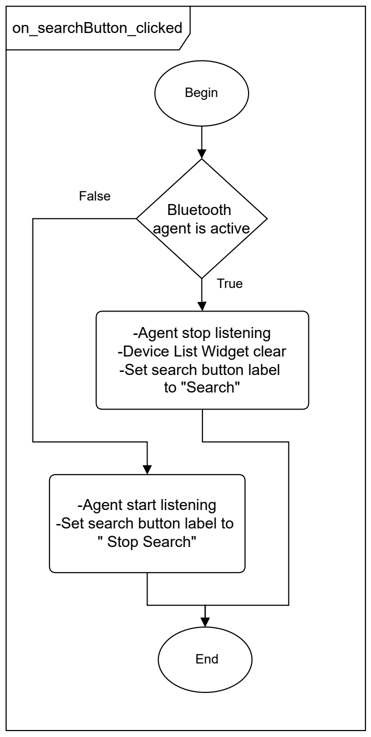
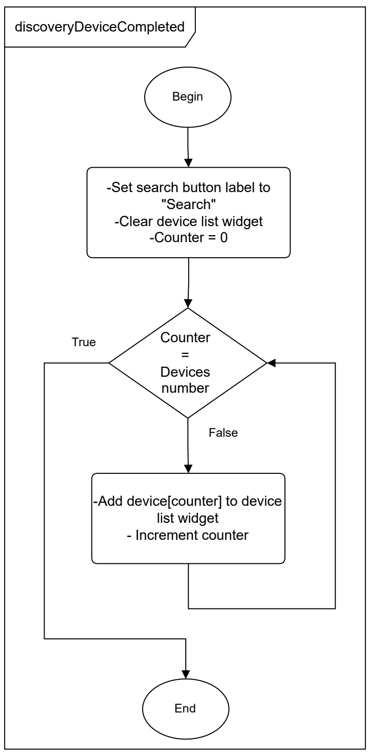

BME68x API
bme68x_get_regs
Begin
BME688 device interface (bme68x_dev->intf) = SPI
False
True
Set memory page (bme68x_dev->mem_page)
-Read register from the sensor and save it (bme68x_read_fptr_t = comm_mux_read)
End
bme68x_get_data
-bme68x_get_data use bme68x_get_regs, which also use comm_mux_read
Begin
BME688 device operation mode = Parallel mode
False
True
Read all fields in the BME688 device and save it
Read the first fields in the BME688 device and save it
End

Figure 58: Software Flowchart of BME68x API

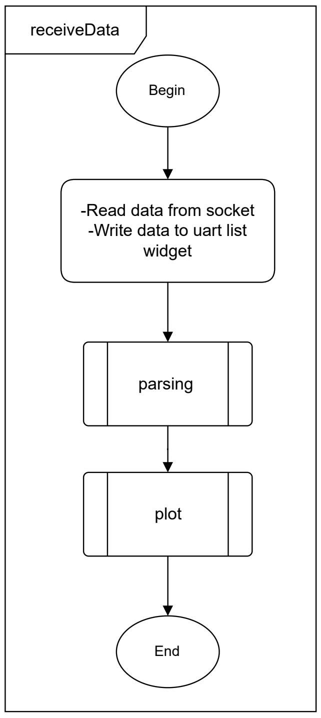
The monitoring process of the readings taken by the BME688 dev-kit board will be performed through an application external to the kit (In a computer that supports Qt). The flowcharts that present the workflow of this application are shown in figure 59. The flowcharts illustrated by 59a, and 59b, represent the process of searching and listing all the Bluetooth devices around the user to select the BME688 Development Kit Bluetooth host. The searching process will be triggered by a button available in the application interface. After establishing the connection between the monitoring application and the kit, the application will be on hold until it receives information from the sensors. After this information arrives to the application, it will execute the procedure described by the flowchart 59c, where it is possible to verify the parsing of the new data and subsequent plot of the same.


on_searchButton_clicked
Begin
Bluetooth agent is active
False
True
-Agent stop listening
-Device List Widget clear
-Set search button label to "Search"
-Agent start listening
-Set search button label to " Stop Search"
End

(a) on_searchButton_clicked function


discoveryDeviceCompleted
Begin
-Set search button label to "Search"
-Clear device list widget
-Counter = 0
Counter = Devices number
True
False
-Add device[counter] to device list widget
- Increment counter
End

(b) discoveryDeviceCompleted function



(c) receiveData function

Figure 59: Software flowcharts of BME688-GUI

The process that follows after collecting the data obtained from the sensors present in the BME688 Development Kit is the training process and subsequent execution of the ML algorithms. Therefore, in the training models process, the developed algorithm is shown in figure 60. This algorithm contains a few steps, since the model training step is performed with the help of the Scikit-learn and TensorFlow libraries. One step that is very important in this flowchart is the last, since it will be helpful in the result stage of this dissertation, allowing the comparison between all ML models.

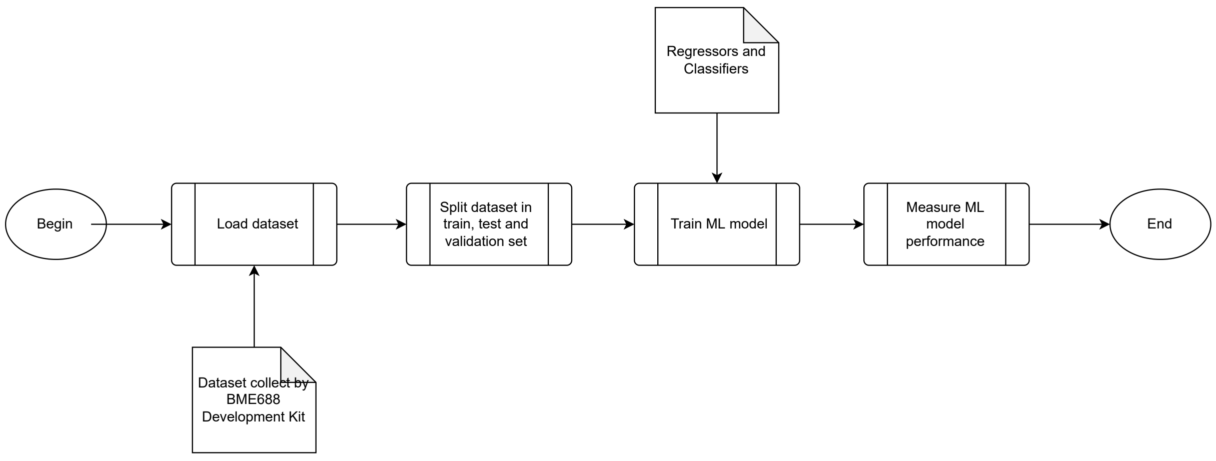

Regressors and Classifiers
Begin
Load dataset
Split dataset in train, test and validation set
Train ML model
Measure ML model performance
End
Dataset collect by BME688 Development Kit

Figure 60: Software Flowchart for machine learning models

4.4.6 Sequence Diagram

Since sequence diagrams are interaction diagrams that detail how operations work, the following sequence diagram (61) demonstrates how it will be the interaction between the BME688 Development Kit and the monitor application. This diagram shows that the BME688 Development Kit will work (save the sensor's data into the SD Card) independently of a Bluetooth device connection. After establishing a Bluetooth connection between the kit and the application, the kit will enter a loop to send information about the sensors to the application, which is then made available to the users by the application.

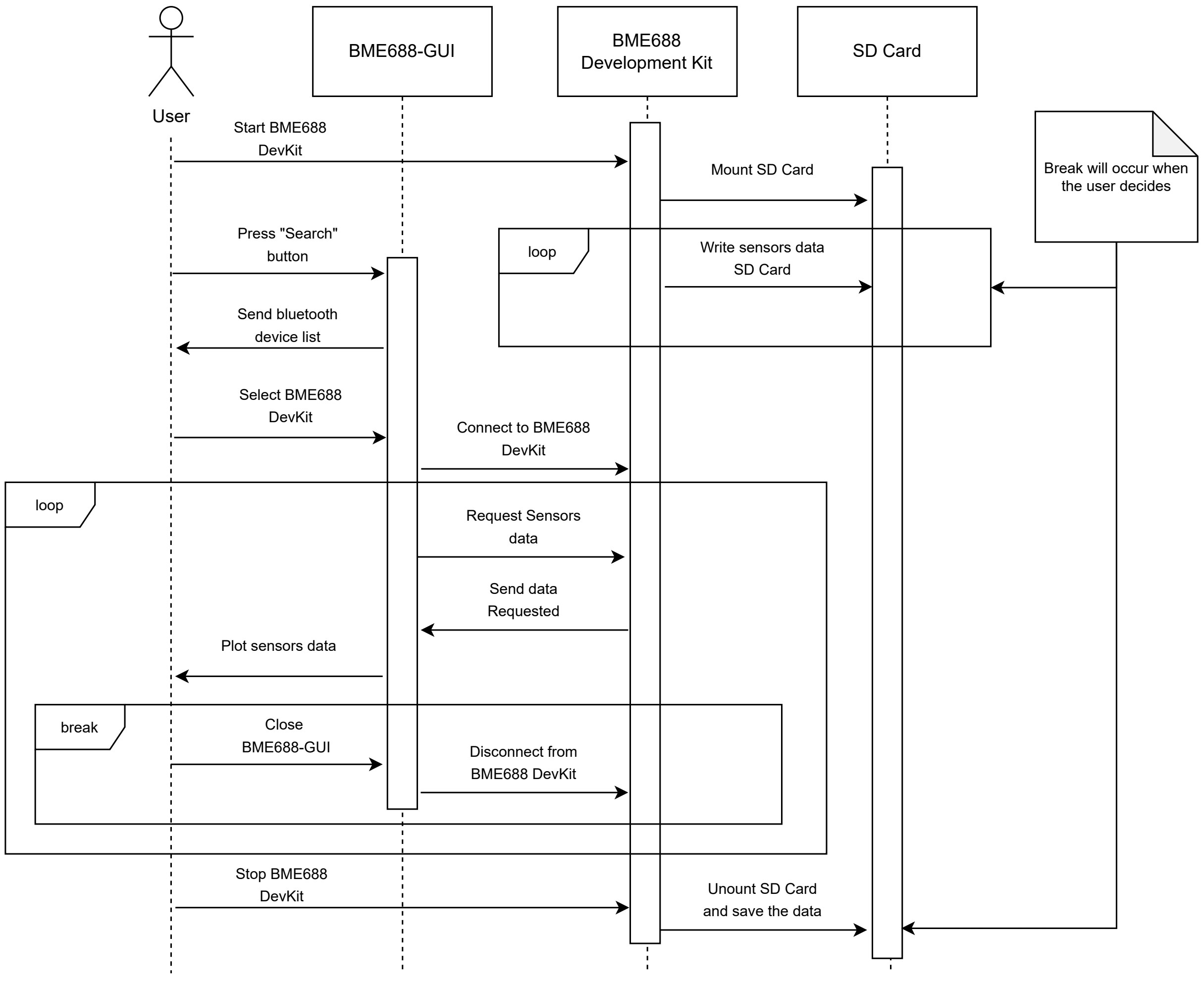


Figure 61: Sequence diagram for data collection process

Chapter 5

Implementation

5.1 BME688 Development Kit

The dataset collection process, as mentioned earlier, was implemented in the BME688 Development Kit. In this process, it was necessary to define the heater profile of each sensor in the kit. The heater profile implementation is represented in figure 49, and as can be seen, the sensors will start by heating to 320 °C, passing by 100 °C, and finishing at 320°C. Each heater step has a run time of one second (mult_prof buffer), which means that all sensors will be heating for one second at each heater step temperature.

Listing 2: Heater Profile buffer

```
// Heater profile
uint16_t temp_prof[10] = {320, 100, 100, 100, 200, 200, 200, 320, 320, 320};
uint16_t mul_prof[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
uint8_t idac_prof[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

uint16_t shared_heatr_dur;
bme68x_heatr_conf_t heatr_conf = {
   .enable = BME68X_ENABLE,
   .heatr_temp_prof = temp_prof,
   .heatr_dur_prof = mul_prof,
   .profile_len = 10
};
```

After defining the sensor's heater profile, the next step is each sensor configuration, including selecting the communication interface between the sensor and microcontroller (SPI), setting the communication functions to treat the sensor's data, and setting the heater profile defined earlier. All these configurations are represented in code 3.

Listing 3: Configuration of all sensors

```
// Config all 8 bme688 sensors
for (uint8_t i = 0; i < SENSORS_NUMBER; i++)
{
   comm_mux[i].comm_mux_intf = &comm_mux_intf;
   comm_mux[i].cs_io = i;
   bme688[i].intf_ptr = &comm_mux[i];
   bme688[i].intf = BME68X_SPI_INTF;
   bme688[i].read = comm_mux_read;
   bme688[i].write = comm_mux_write;
   bme688[i].delay_us = comm_mux_delay_us;
   bme688[i].amb_temp = 25;
   bme68x_init(&bme688[i]);
   bme68x_get_conf(&conf, &bme688[i]);
   conf.os_hum = BME68X_OS_1X;
   conf.os_temp = BME68X_OS_16X;
   conf.os_pres = BME68X_OS_2X;
   bme68x_set_conf(&conf, &bme688[i]);
   shared_heatr_dur = 1000 - (bme68x_get_meas_dur(BME68X_PARALLEL_MODE, &conf, &bme688[i])
       ↪ / 1000); // 1s
   heatr_conf.shared_heatr_dur = shared_heatr_dur;
   bme68x_set_heatr_conf(BME68X_PARALLEL_MODE, &heatr_conf, &bme688[i]);
   bme68x_set_regs(&reg_addr, idac_prof, 10, &bme688[i]); // Boost
}
```

As mentioned previously, BME688 sensor has three operations mode (sleep mode, forced mode, parallel mode), all represented in table 3. In this project, the operation mode required was the parallel mode since it is the only one that allows multiple TPHG (Temperature, Pressure, Humidity, and Gas Resistance) cycles of measurements. The code presented in 4 is responsible for the configuration of every sensor operation mode.

Listing 4: Operation mode configuration of all sensors

```
for (uint8_t i = 0; i < SENSORS_NUMBER; i++)
{
   bme68x_set_op_mode(BME68X_PARALLEL_MODE, &bme688[i]);
}
```

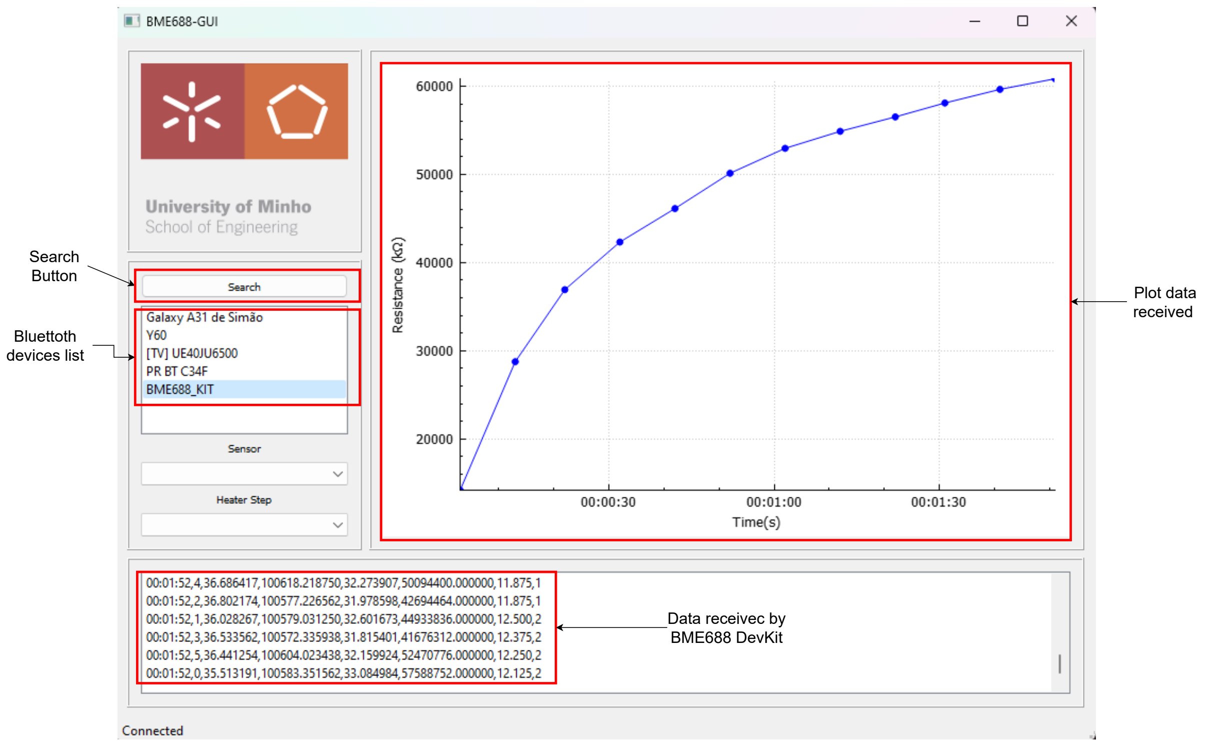
The BME68x API supplies a lot of functions that help the communication between the main program and the BME688 sensor. One of those functions is the bme68x_get_data, responsible for reading the pressure, temperature, humidity, and gas data, from the sensor. All this information is stored in the bme68x_data structure instance passed to bme68x_get_data as an argument. The code 5 shows the workflow of the BME688 Development Kit to collect the sensors data and save it in the SD Card.

Listing 5: Read data from of all sensors

```c
for (uint8_t i = 0; i < SENSORS_NUMBER; i++) {
   bme68x_get_data(BME68X_PARALLEL_MODE, data, &n_fields, &bme688[i]);
   for (uint8_t field = 0; field < 3; field++) {
      // New data
      if (data[field].status & BME68X_NEW_DATA_MSK) {
         f = fopen("/sdcard/data.txt", "a");
         if (f != NULL) {
            rtc_get_hour(&hour);
            rtc_get_minutes(&minutes);
            rtc_get_seconds(&seconds);
            idac = ((float)(data[field].idac) + 1) / 8;
            // Timestamp
            fprintf(f, "%02x:%02x:%02x", hour, minutes, seconds);
            // Sensor Index
            fprintf(f, "%d,", i);
            // Temperature
            fprintf(f, "%f,", data[field].temperature);
            // Pressure
            fprintf(f, "%f,", data[field].pressure);
            // Humidity
            fprintf(f, "%f,", data[field].humidity);
            // Gas Resistance
            fprintf(f, "%f,", data[field].gas_resistance);
            // DAC Current
            fprintf(f, "%.3f,", idac);
            // Heater Step
            fprintf(f, "%d,", data[field].gas_index);
            // Valid data or not
            if (data[field].status == 0xB0) {
               fprintf(f, "1\n");
            }
            else {
               fprintf(f, "0\n");
            }
            bluetooth_write(buffer); // buffer contain all BME688 DevKit data
            fclose(f);
         }
      }
   }
```

5.2 BME688 Graphical User Interface

The application used to monitor the entire data collection process is shown in figure 62. In this figure, it is possible to see the graph that will present the BME688 sensor's resistance subjected to the gases under study.



```
00:01:52,4,36.686417,100618.218750,32.273907,50094400.000000,11.875,1
00:01:52,2,36.802174,100577.226562,31.978598,42694464.000000,11.875,1
00:01:52,1,36.028267,100579.031250,32.601673,44933836.000000,12.500,2
00:01:52,3,36.533562,100572.335938,31.815401,41676312.000000,12.375,2
00:01:52,5,36.441254,100604.023438,32.159924,52470776.000000,12.250,2
00:01:52,0,35.513191,100583.351562,33.084984,57588752.000000,12.125,2
```

Figure 62: BME688 GUI

The monitoring application also displays a list of Bluetooth devices available to establish a connection, as well as a window that contains all the information received by the Bluetooth device connected to the application. This application works in real-time, since its goal is to update the user on the current state of the BME688 development kit while it collects the data needed to train various machine-learning models.

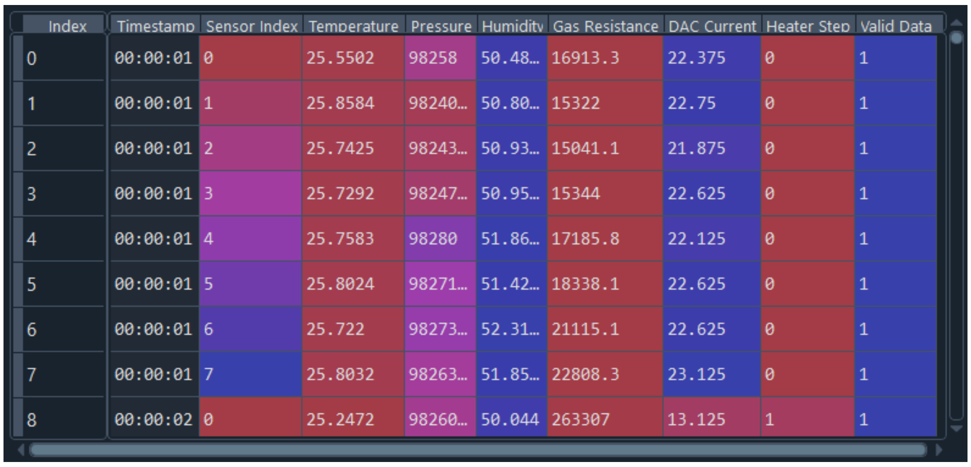
The graph in the application, used to plot the data received by the Bluetooth device, was developed with the help of the QCustomPlot API. This API is a Qt widget for plotting and data visualization. It is a customizable and powerful solution for displaying graphs and charts in Qt-based applications. QCustomPlot is built on top of the Qt graphics framework and provides a wide range of features for creating professional-quality plots and charts. Some features of QCustomPlot include ([10]):

- Support for multiple axes and plot layouts;
- A wide range of plot types, including line plots, bar plots, scatter plots, and more;
- Customization of plot elements, such as axis labels, tick marks, and grid lines;

- Support for user interactions, such as zooming, panning, and selecting data points;
- Export to various image formats, such as PNG, JPG, and PDF.

5.3 Data Structure

The data collected by the BME688 Development Kit is organized in a certain way, represented in figure 63.



Index	Timestamp	Sensor Index	Temperature	Pressure	Humidity	Gas Resistance	DAC Current	Heater Step	Valid Data
0	00:00:01	0	25.5502	98258	50.48…	16913.3	22.375	0	1
1	00:00:01	1	25.8584	98240…	50.80…	15322	22.75	0	1
2	00:00:01	2	25.7425	98243…	50.93…	15041.1	21.875	0	1
3	00:00:01	3	25.7292	98247…	50.95…	15344	22.625	0	1
4	00:00:01	4	25.7583	98280	51.86…	17185.8	22.125	0	1
5	00:00:01	5	25.8024	98271…	51.42…	18338.1	22.625	0	1
6	00:00:01	6	25.722	98273…	52.31…	21115.1	22.625	0	1
7	00:00:01	7	25.8032	98263…	51.85…	22808.3	23.125	0	1
8	00:00:02	0	25.2472	98260…	50.044	263307	13.125	1	1

Figure 63: Dataset Structure

This figure shows the sensor number, the data read by the respective sensor (Temperature, Pressure, Humidity, Gas Resistance, and DAC Current), the timestamp, the heater step (Temperature at which the sensor is working), and a value that indicates if the data collected is valid or not.

The DAC Current is a value that shows the current presented in the BME688 sensor that will make the internal sensor resistance achieve the requested heater temperature. The BME688 sensor contains a control loop that periodically measures the heater resistance value and adapts the value of current injected from a DAC [26].

5.4 BME688 Sensor Behavior

Before starting the training process of the machine learning models, it is necessary to understand the sensor operating principle when subjected to a particular gas and the influence of the heater profile on its response. To this purpose, an experiment was conducted, consisting in recording the response of a sensor configured to work at a single temperature (320°C) and configured to work at different temperatures, both performed in ambient air. This experiment intends to study the influence of the heater profile on the sensor response. The image 64 shows the experiment result, where it can be noticed that the sensor reacts much faster when set to work at different temperatures than at various temperatures, leading to the conclusion that using a heater profile helps reduce the response time of the BME688 sensor.

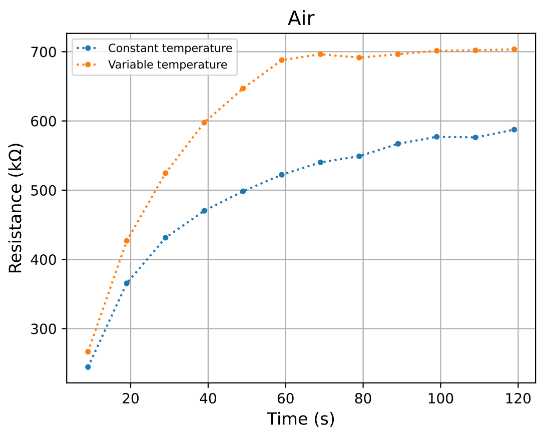


Figure 64: BME688 response time to ambient air

5.5 Dataset for the Classifiers

The gas chosen to train the classifiers was CO_2 because of its ability to cause shortness of breath and other harmful effects on human health when accumulated at high concentration levels. The dataset that will serve as the basis for training the classifiers, was collected from eight BME688 sensors, belonging to the BME688 development kit. Figure 65 shows the setup used for collecting the classifier's dataset, where it is possible to see the container responsible for keeping the data samples (air with and without CO_2) in a controlled environment. Inside the container, beyond the sample data, was also placed the BME688 Development Kit connected with a battery for power supply reasons.

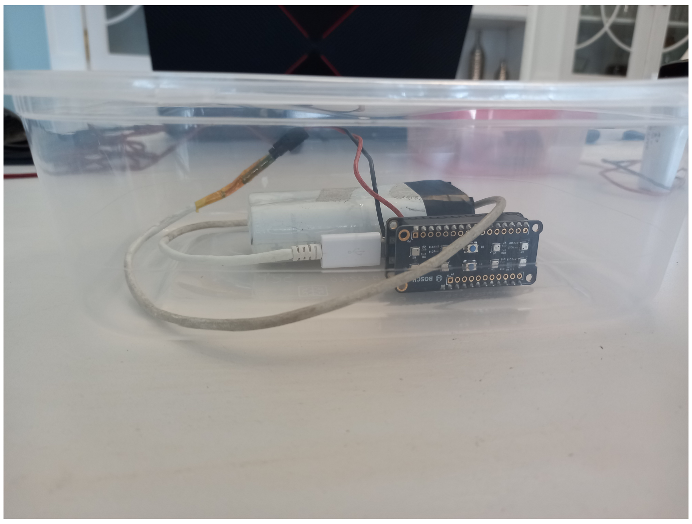


Figure 65: Setup used to collect the data for the classifiers models

The experimental procedure lasted 6 minutes for each sample (ambient air and CO_2), and during those 6 minutes, the gas resistance recorded is represented in figures 66 and 67. After analyzing those graphs, the gas resistance in each concentration has different behaviors when submitted to non-identical heater steps (higher the temperature, lower the gas resistance). Even though the sensor response is accelerated with the heater profile application, the sensor always takes some time to react to a specific gas. Therefore, for the total CO_2 and ambient air dataset, the initial gas instances were removed, resulting in a training and testing dataset with a time range between 1 minute and 6 minutes.

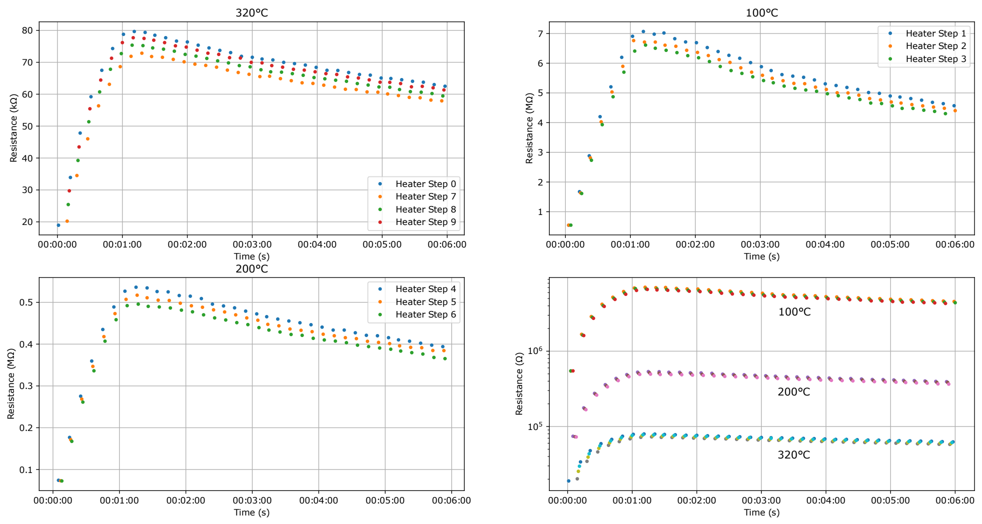


Figure 66: Gas Resistance collected in the ambient air environment

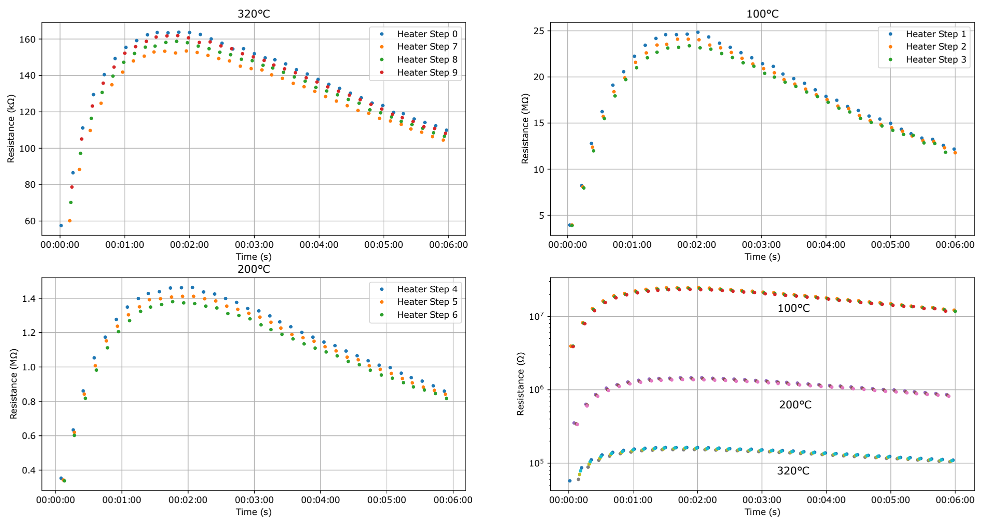


Figure 67: Gas Resistance collected in the CO_2 environment

5.5.1 Training and Test set

The only way to know how well a model will generalize to new cases is to try it out in other instances. The best solution to do it is splitting the data into two sets (training and test set). The training set will be used to train the model, while the test set will be helpful in the test procedure, indicating if the model is good. So, after collecting the data for the classifiers, it is necessary to split the dataset into two sets. For the creation of the test set was used the function *train_test_split* provided by the Scikit-learn library.

5.6 SVM Classifier

This subsection shows the implementation of the SVM classifier with the dataset collected earlier. Figure 68 shows the data collected by one sensor of the BME688 Development Kit. The SVM classifier developed was trained with this dataset since it is crucial to understand whether the sensor can distinguish between two different types of gases, in this case, CO_2 and ambient air. As can be seen in figure 68, the data collected is not linearly separable, requiring a transformation of the dataset space (i.e., adding new features to the dataset to create a 3D dataset)

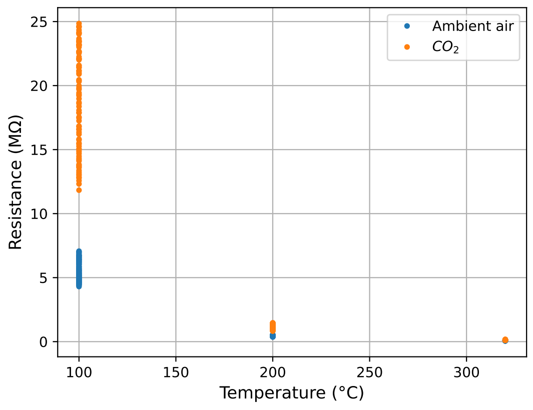


Figure 68: BME688 resistance in CO_2 and ambient air

To implement the SVM classifier, first a *Pipeline* containing a *PolynomialFeatures* was created, followed by a *LinearSVC*. The first method creates a new feature ($Z = X \cdot Y$) that will turn the nonlinear dataset into a linear one (Figure 70 shows the new dataset). The *LinearSVC* finds the best hyperplane that separates the two classes presented in the dataset (ambient air and CO_2). This hyperplane is also represented in figure 70. The training process of the SVM classifier is shown in code 6 and its result is represented in figure 69, which is a 2D representation of figure 70.

Listing 6: SVM Classifier

```
polynomial_svm_clf = Pipeline ([
   ("poly_features", PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)),
       ↪ # add column z=x*y -> (x, y, z)
   ("svm_clf", LinearSVC(C=5, loss="hinge"))
   ])
```

```
polynomial_svm_clf.fit(X_train, y_train)
```

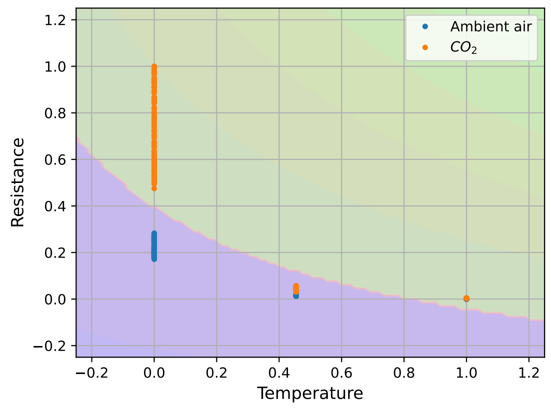


Figure 69: Linear SVM classifier using Polynomial features

The blue function presented in figure 70 corresponds to the mapping function used to transform the old dataset into the new one.

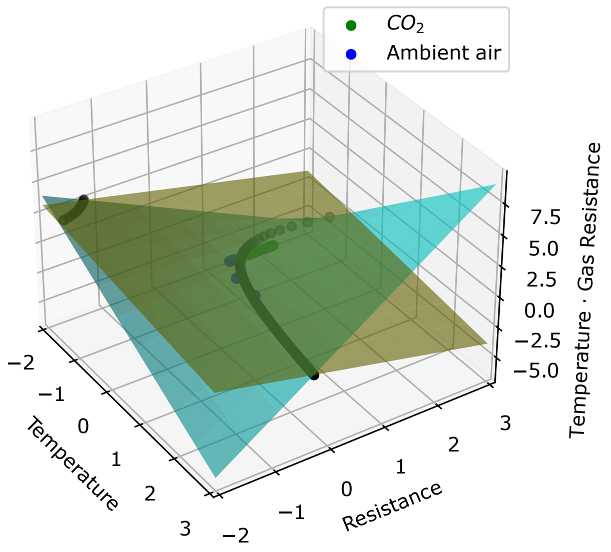


Figure 70: Linear SVM classifier using polynomial features (3D visualization)

As can be seen, the interception between the mapping function and the hyperplane corresponds to the classifier presented in figure 69, responsible for separating the two classes: (CO_2) and Ambient air. One aspect mentioned before was the evaluation of machine learning model performance. In the case of the SVM classifier, the evaluation method chosen was accuracy. The implementation of this metric is represented in code 7, and its result was around **0.67**. The accuracy was calculated in the test set mention earlier (20% of total dataset).

Listing 7: SVM Classifier Accuracy

```
# SVM Classifier Accuracy
y_pred = polynomial_svm_clf.predict(X_test)
print("SVM Classifier Accuracy:", end=" ")
print(accuracy_score(y_test, y_pred))
```

The performance of the resulting model has a very low accuracy due to the scale difference between resistance values for different temperatures (heater steps). An alternative to solve this problem is to remove the temperature as an input parameter of the model, thus making it entirely dependent on the resistance value of the sensor. Figure 71 illustrates resistance values at a temperature of 320°C, where a vertical line can be identified to separate the data into two distinct classes.

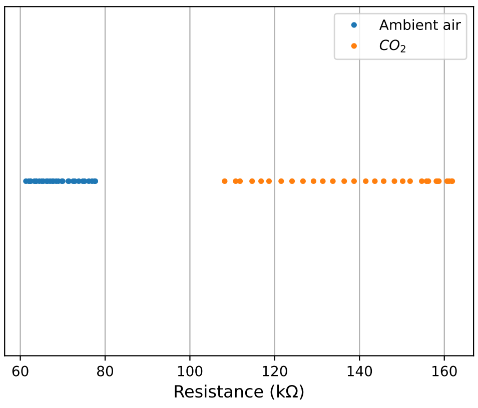


Figure 71: BME688 resistance at 320°C for CO_2 and ambient air

By training a classifier with 80% of the data shown in figure 71 it is possible to train a model with an accuracy equal to 1 that finds a hyperplane (in the case of a 1-dimensional dataset, the hyperplane corresponds to a point) that separates the two gas. Figure 72 shows the SVM classifier model result with one sensor as input.

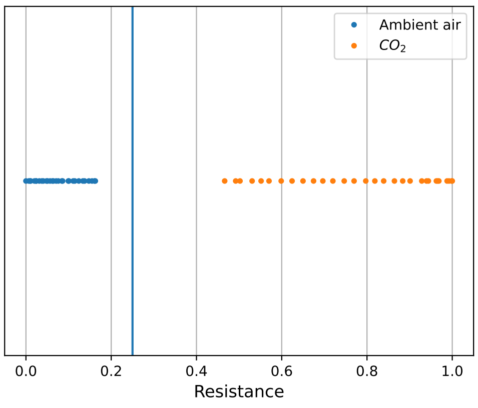


Figure 72: SVM classifier with one sensor resistance as input

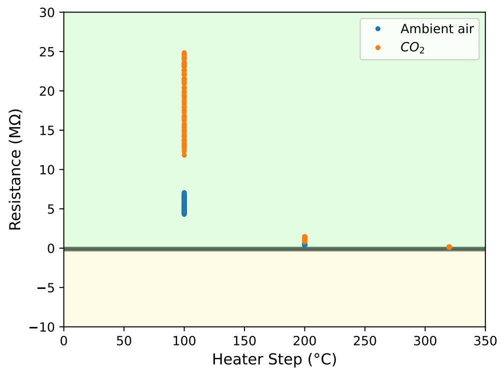
5.7 Decision Tree Classifier

The decision tree classifier implementation will be performed over the same dataset used by the SVM classifier presented in the subsection 5.6. The decision tree training process was done using the DecisionTreeClassifier function provided by the Scikit-learn library, where code 8 shows the training process implementation of the decision tree classifier.

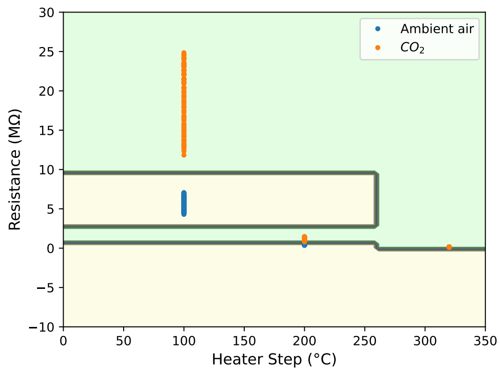
Listing 8: Decison Tree Classifier

```
tree_clf = DecisionTreeClassifier(max_depth=6)
tree_clf.fit(X_train, y_train)
```

In the decision tree classifier implementation, the *max_depth* parameter was set to 6. This choice resulted from comparing several graphs created from training some decision tree models with different values for the *max_depth* parameter. Figure 73 represents two decision tree classifiers that are not the best option to separate the two classes presented in the dataset (CO_2 and ambient air).



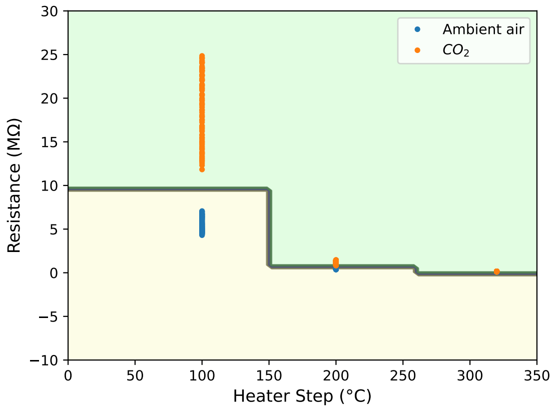
(a) Decision tree with *max_depth* parameter equal to 2



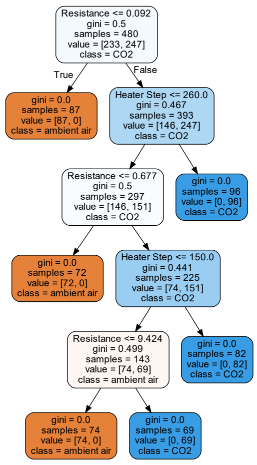
(b) Decision tree with *max_depth* parameter equal to 8

Figure 73: Decision trees with *max_depth* parameter different from 6

Figure 73b, which represents the decision tree classifier with the parameter *max_depth* equal to 8, is overfitting the data, whereas figure 73a is underfitting, leading to the best solution for a classifier the *max_depth* parameter set to 6 (figure 74a). This classifier is characterized by a tree of conditions that is represented in figure 74b. After the training process of the decision tree classifier, the model accuracy calculation was performed using the training dataset (20%). The evaluation result was **1**, indicating that the classifier, whose input parameters are the internal resistance of the sensor and the corresponding operating temperature, can distinguish between a CO_2 sample and an ambient air sample.



(a) Decision tree with *max_depth* parameter equal to 6



(b) Tree of conditions for CO_2 detection

Figure 74: Decision Tree classifier for CO_2 detection with 3 temperatures

Just as done for the SVM classifier, it is also possible to train a decision tree model with only a sensor resistance as input. This approach was applied in the case of the SVM classifier to increase its accuracy. However, in the case of the decision tree, the reason for implementing this technique is to reduce the costs and resources needed to run the model. Figure 75 show the tree of conditions resulted from training a decision tree with 80% of the resistances instances at 320°C.

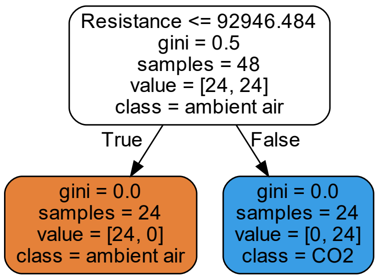


Figure 75: Tree of conditions for CO_2 detection with 1 sensor

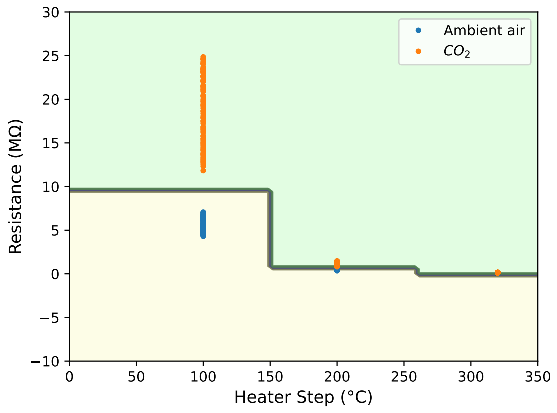
5.8 Random Forest Classifier

Despite the great accuracy results with the SVM and the decision tree classifier, two random forest classifiers were also developed for CO_2 detection. The Random Forest classifiers implementation is based on the bagging method to mitigate the risk of overfitting. Code 9 trains a group of three decision tree classifiers (i.e., random forest classifier) using an ensemble learning approach.

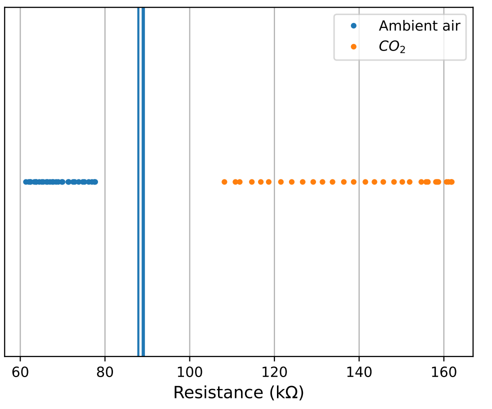
Listing 9: Random Forest Classifier

```
# Bagging
bag_clf = BaggingClassifier (
   DecisionTreeClassifier(max_leaf_nodes=6),
   n_estimators=3, max_samples=1.0, bootstrap=True, n_jobs=-1
)
bag_clf.fit(X_train, y_train)
```

The resulting classifiers are represented in figure 76, and it is possible to see that those models are quite good at separating the ambient air from CO_2.



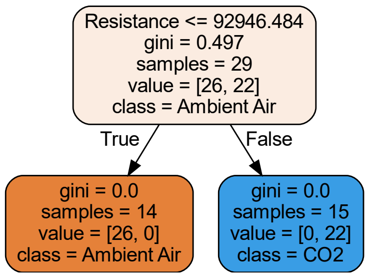
(a) Random forest with 3 temperatures



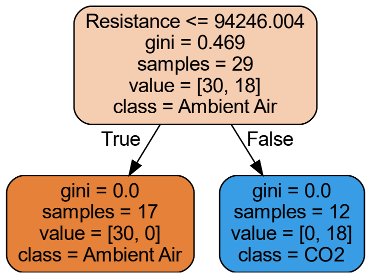
(b) Random forest with 1 sensor

Figure 76: Random classifiers for CO_2 detection

Following the training of the random forest classifiers, the subsequent phase involves assessing their accuracy using the test dataset, which constitutes 20% of the overall dataset. The resulting evaluation yielded a value of **1** for both models, signifying their proficiency in distinguishing between CO_2 and ambient air instances. For the model with the resistance of a sensor as an input parameter, two of the condition trees that make up its random forest are shown in figure 77.



(a) Decision tree n°1 with 1 sensor



(b) Decision tree n°2 with 1 sensor

Figure 77: Two of the three classifiers that make up the random forest with 1 sensor resistance as input

Since the accuracy of a random forest increases with the number of decision trees, all implemented random forest models are composed of three decision trees. This number is not too large since the project aims to study ML models for systems with scarce resources.

5.9 Classifiers Tests and Results

The main difference between a regressor and a classifier is that regression algorithms are used to predict continuous values, such as age, size, etc., and classifiers are used to categorize discrete values, such as True or False, one or zero. As a result, evaluating a classifier is different from evaluating a regressor. This subchapter will show all the performance measures executed in the classifiers developed in this dissertation (SVM, Decision Tree, Random Forest).

5.9.1 Accuracy

Table 4 displays the accuracy scores obtained by the classifiers during evaluation with the test dataset. The results indicate that all classifiers performed similarly, except the SVM classifier that utilized 3 temperatures as input, which displayed inferior performance.

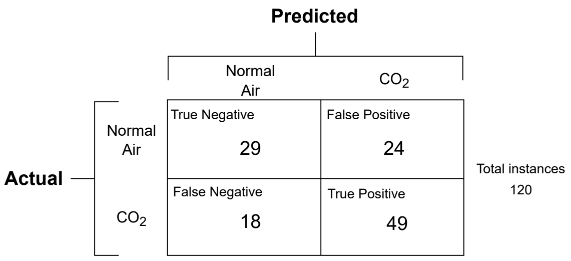
Table 4: Classifier's Accuracy

Classifiers	**Accuracy**
SVM with 3 temperatures	0.67
Decision Tree with 3 temperatures	1
Random Forest with 3 temperatures	1
SVM with 1 sensor	1
Decision Tree with 1 sensor	1
Random Forest with 1 sensor	1

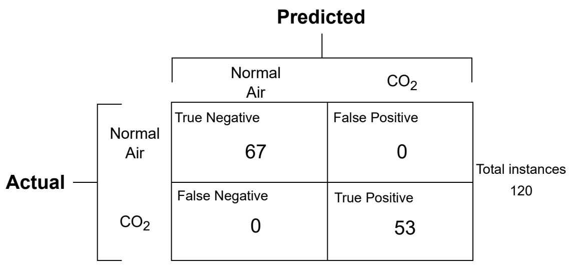
This method of evaluating the performance of a classifier can sometimes mislead the user. For example, in a dataset of photos built by only 5% of photos with dogs, the probability of a classifier trained to indicate that the sample received as input is another animal is 95%. This example shows that accuracy is not advised to some classifiers, especially when they are trained in *skewed datasets* (datasets with some classes much more frequent than others) [15].

5.9.2 Confusion matrix

As was explained in the first chapters, another way to evaluate a classifier is by analyzing the confusion matrix of its model. Figures 78a, 78b and 78a demonstrate the confusion matrices resulted from all classifiers with temperature and resistance as input parameters previously implemented.

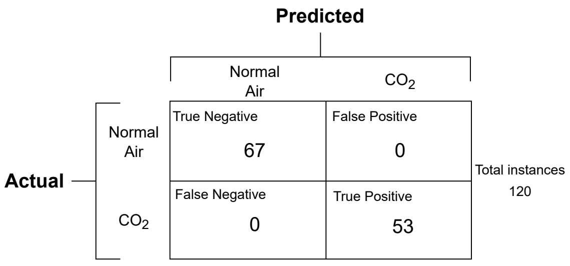


(a) SVM confusion matrix



(b) Decision Tree confusion matrix

Figure 78: Confusion matrices of the classifiers with 3 temperatures as input



(a) Random Forest confusion matrix

Figure 78: Confusion matrices of 3 temperature classifiers

The first row of each confusion matrix represents the non-CO_2 instances (negative class), and the second row shows the CO_2 instances. In the non-CO_2 row, the cell to the left shows the number of points correctly classified as non-CO_2. The cell to the right shows the number of non-CO_2 instances classified as CO_2. The classifier that performs the worst classification of non-CO_2 data is the SVM since it has the higher value of *False Positives* points of all classifiers. This model has a higher value of *False Negative* instances, further reinforcing the idea that it is the model with the lowest performance.

Regarding the classifiers with resistance as the only input parameter, the confusion matrix is the same for all three, indicating a similar performance. The result is represented in figure 79, where it can be seen that the dataset size is considerably smaller than the models incorporating 3 temperatures because this particular dataset comprises only 20% of the instances recorded at a temperature of 320°C.

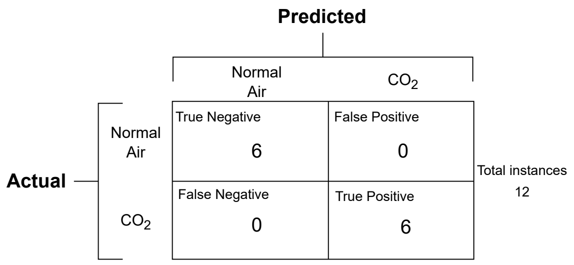


Figure 79: Confusion matrix of the classifiers with 1 sensor as input

5.9.3 Precision and Recall

There are several ways to evaluate a classifier, and two of them are precision and recall. Those metrics calculations are represented in equations 24 and 25, respectively. These equations are also illustrated in the theory chapter.

$$precison = \frac{TP}{TP + FP} \tag{24}$$

$$recall = \frac{TP}{TP + FN} \tag{25}$$

Table 5 shows the results from precision and recall calculations of SVM, Decision Tree, and Random Forest Classifiers. In this table, it is possible to see that the model with the worst performance is the SVM classifier with 3 temperatures.

Table 5: Classifier's Precision/Recall

Classifiers	**Precision**	**Recall**
SVM with 3 temperatures	0.67	0.73
Decision Tree with 3 temperatures	1	1
Random Forest with 3 temperatures	1	1
SVM with 1 sensor	1	1
Decision Tree with 1 sensor	1	1
Random Forest with 1 sensor	1	1

5.9.4 ROC Curve

The receiver operating characteristic (ROC) curve is a frequently used tool for binary classifiers (which have only two classes). The *true positive rate* (recall) and *false positive rate* are plotted on the ROC curve.

The ratio of negative events that are mistakenly labeled as positive is known as the FPR (*false positive rate*). It equates to one less than the *true negative rate* (TNR). As a result, the ROC curve represents *sensitivity* (recall) as a function of one less TNR [15]. Figure 80 shows the ROC curves of all classifiers implemented in the previous chapter.

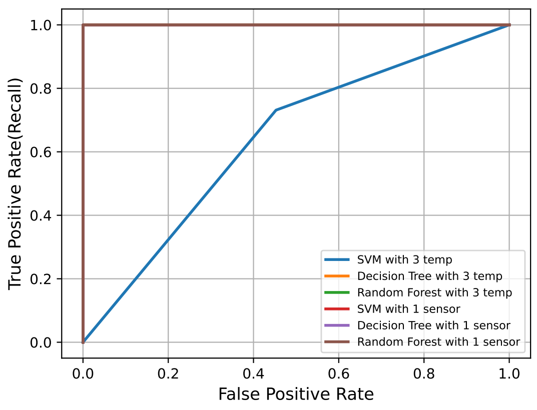


Figure 80: ROC Curves

The *area under the curve* (AUC) can be used to compare classifiers, and as can be seen, the SVM classifier with 3 temperatures performs worst than the others since it has an AUC lower than the others.

5.9.5 Computational Complexity

During the development of an algorithm, it is crucial to always keep in mind the temporal and physical cost it may occupy. In other words, it is necessary to establish a compromise between the space occupied and the execution time of an algorithm or model. The Big O notation is a mathematical notation that describes the limiting behavior of a function when its arguments tend to infinity values [24]. Table 6 shows the computation complexity for each classifier in the Big O notation.

Table 6: Classifier's Computation Complexity

Classifier	**Time Complexity**	**Space Complexity**
SVM	$O(l \cdot d)$	$O(l \cdot d)$
Decision Tree	$O(log(n))$	$O(m)$
Random Forest (Three Classifiers)	$O(k \cdot log(n))$	$O(k \cdot m)$

where l corresponds to the number of support vectors in the SVM classifier, d the dimension of the dataset, n the size of the dataset, m the node's number and k the number of decision trees inside the random forest. This notation is related to the running time of the models and not concerning the training process, since one of the objectives of this dissertation is to study and select ML algorithms for resource-scarce systems. Therefore, looking at the training models, it is possible to collect all the variables (l, d, n, m, and k) necessary to choose which model presents the higher temporal and spatial complexity.

The number of support vectors in the SVM classifier with 3 temperatures is 261 ($l = 261$), and the dimension is equal to 2 since the model has two inputs (Heater Step and Resistance). The SVM classifier with 1 sensor has 6 support vectors (dataset with 1D dimension). The other variables have these values: $n = 600$ for 2 inputs classifiers, $n = 60$ for classifiers with a 2D dataset, $m = 2$ for the decision trees with 1 sensor as input, and 6 for each decision tree with 3 temperatures as input, and finally $k = 3$ (random forest with 3 decision trees). The resulted calculation is presented in the table 7.

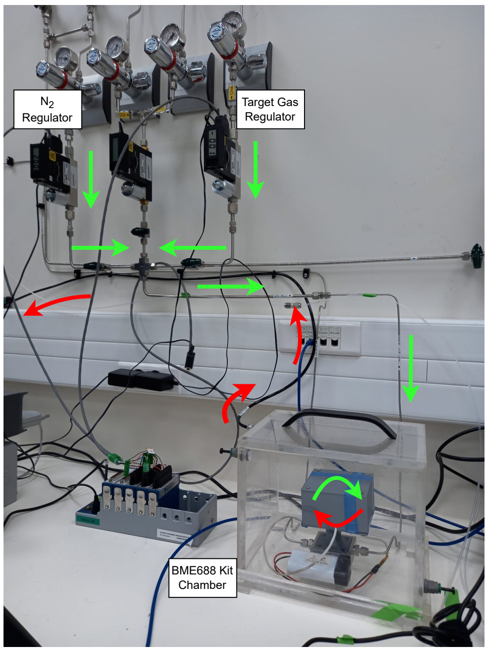
Table 7: Classifier's Computation Complexity result

Classifier	**Time Complexity**	**Space Complexity**
SVM with 3 temperatures	$O(261 \cdot 2 = 522)$	$O(261 \cdot 2 = 522)$
Decision Tree with 3 temperatures	$O(log(600) = 2.83)$	$O(6)$
Random Forest with 3 temperatures	$O(3 \cdot log(600) = 8.49)$	$O(3 \cdot 6 = 18)$
SVM with 1 sensor	$O(6 \cdot 1 = 6)$	$O(6 \cdot 1 = 6)$
Decision Tree with 1 sensor	$O(log(60) = 1.77)$	$O(1)$
Random Forest with 1 sensor	$O(3 \cdot log(60) = 5.33)$	$O(3 \cdot 1 = 3)$

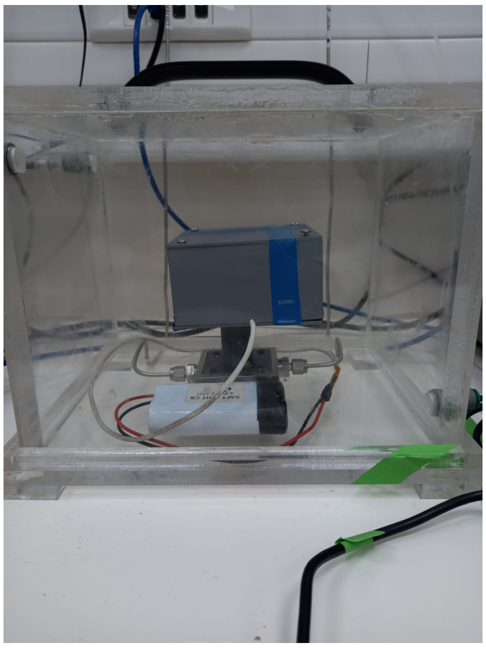
The classifier with a time and space complexity lower is the decision tree model (1 input), unlike the SVM (2 inputs), which has the most expensive cost.

5.10 Dataset for the Regressors

Instead of using classifiers to detect carbon dioxide, the following chapters will train and implement regressors to predict the amount of methane (CH_4) in a given environment. This selection of different gases (datasets) for both the classifiers and regressors was determined by the objective of this dissertation to conduct a comprehensive study of the BME688 sensor's performance across several gas types. To the regressors train, it was collected a dataset with different concentrations of methane using the eight BME688 sensors that come with the BME688 development kit. The setup used to measure several levels of CH_4 concentrations is represented in figures 81a and 81b.



(a) Setup used to collect the regressor's dataset



(b) Controlled environment to release the gas

Figure 81: Setup for the regressor's dataset

Figure 81b shows the controlled environment where it was released several concentrations of CH_4, and where the BME688 development kit was placed. The CH_4 gas concentration released into the controlled environment was managed by two flow regulators presented in the upper part of figure 81a. During this collection process, a control gas was used as the reference point for all readings. The selected gas for this purpose was N_2, owing to its prevalence as the primary component of the Earth's atmosphere, and its low reactivity. One of the flow regulators was deployed to monitor the concentration of N_2, while the other regulated the concentration level of the target gas (CH_4). The range of concentrations chosen were as follows: 0 ppm (0 % LEL), 4100 ppm (9.36 % LEL), 8200 ppm (18.64 % LEL), 12300 ppm (27.96 % LEL), 16400 ppm (37.28 % LEL), 20500 ppm (46.6 % LEL), 24600 ppm (55.92 % LEL), 28700 ppm (65.24 % LEL), 32800 ppm (74.56 % LEL), 36900 ppm (83.88 % LEL), and 41000 ppm (93.2 % LEL) (LEL refers to Lower Explosive Limit). These values were decided because the maximum value of CH_4 allowed to release into the controlled environment is 41000 ppm. Each concentration reading took about 7 minutes, resulting in a total of 77 minutes to collect the dataset needed for the regressors.

Following CH_4 dataset collection, a thorough study was conducted to assess its behaviour with the BME688 sensor. Figure 4 displays the relationship between concentration and internal resistance for a randomly selected sensor (sensor number 5) from the BME688 development kit, observed during heater step 9 (320 °C) at minute 5:53 (it allows the gas under study to settle inside the chamber). The figure 82 demonstrates the non-linear behavior of the sensor and the non-uniqueness of the internal resistance/-concentration function. Consequently, it is challenging to accurately determine the CH_4 concentration based on individual resistance values.

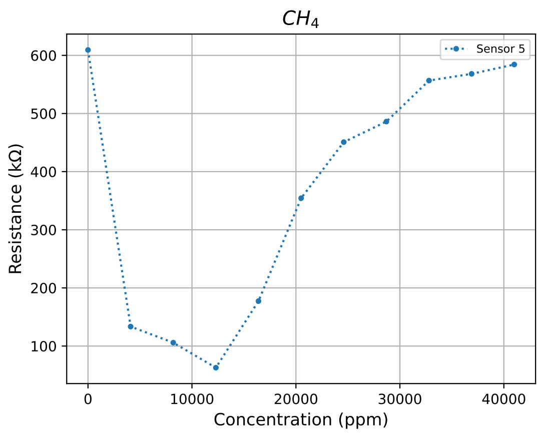


Figure 82: Concentration/Resistance relation of sensor 5 at heater step 9

A potential approach to avoid the challenge of identifying the concentration of a gas sample based on identical resistance values is to aggregate measurements obtained at different temperatures from a single sensor. Figure 83 displays the results obtained from the last three heater steps for each temperature point in the heater profile. Notably, the three graphs never intersect, indicating that the internal temperature variation of the sensor only accelerates the response time of the sensor, and does not impact the accuracy of concentration measurements. A logarithmic scale was adopted to facilitate visualization. This phenomenon was also visible in the classifier dataset, since the resistance scale between temperatures was too different.

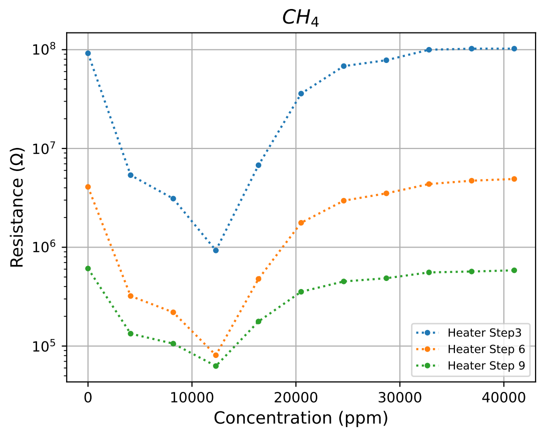


Figure 83: Concentration/Resistance relation of sensor 5 at heater step 3, 6 and 9

An alternative approach to addressing the issue of ambiguity is to combine data collected at a specific temperature (320°C, which has the highest sensitivity) from multiple BME688 sensors. Figure 84 illustrates the data acquired from the six BME688 sensors (Two of them had a much longer working time, which led to different behaviors) in the development kit at heater step 9 at minute 5:53, where it is possible to verify some crossing points. This characteristic can be used to address the ambiguity problem by using multiple sensors to determine a unique CH_4 concentration value based on the combination of different sensors' resistance.

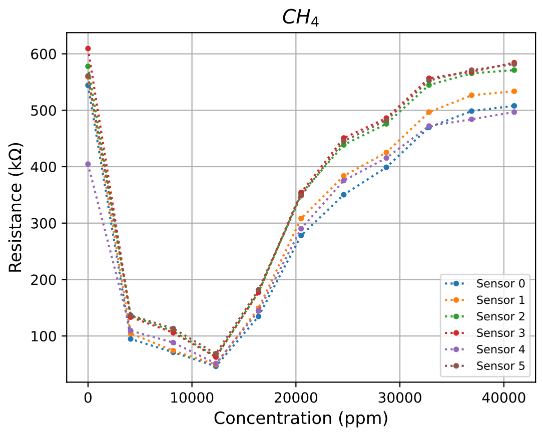


Figure 84: Concentration/Resistance relation of 6 sensors at heater step 9

5.10.1 Training and Test set

The training and test dataset creation for the regressors was done the same way as for the classifiers. The test set corresponds to 20% of the original dataset (sum of all concentrations), and the training set is the remaining 80%. The test set will help the evaluation process of the model trained with the training set. This division is frequent in machine learning, as it helps in the process of developing a model [17].

5.11 Artificial Neural Network (ANN)

This subsection will show several approaches to predict the CH_4 concentration in a specific environment, using ANNs as the prediction model. The model that will serve as the basis for the various ANNs is represented in figure 85, where it is possible to observe that the model input parameters are the internal resistances of 6 BME688 sensors. Still, within this subsection, several models will also be developed with different numbers of hidden layers, always having as input parameters 6 BME688 sensors. The selection of neural networks as an algorithm for CH_4 detection was based on the fact that, in general, they have a reasonable execution time and complexity level, which makes their implementation relatively easy in resource-constrained systems. To ensure the performance of the artificial neural networks (ANNs), a normalization procedure was applied to the training dataset. This ensured that all input values fell within the range of 0 and 1, thereby mitigating any issues related to scale sensitivity. Furthermore, an important factor that was not depicted in figures 85 is that all neurons within the ANN were activated using the rectified linear unit (ReLU) function.

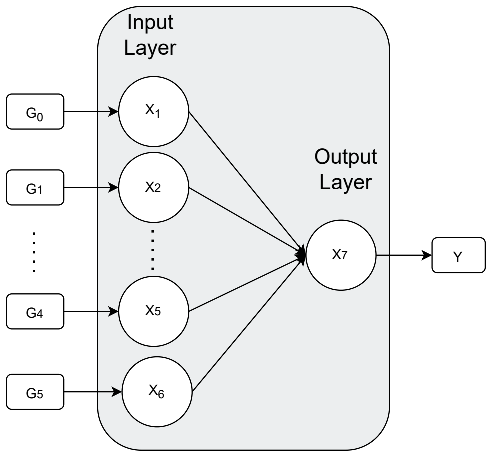
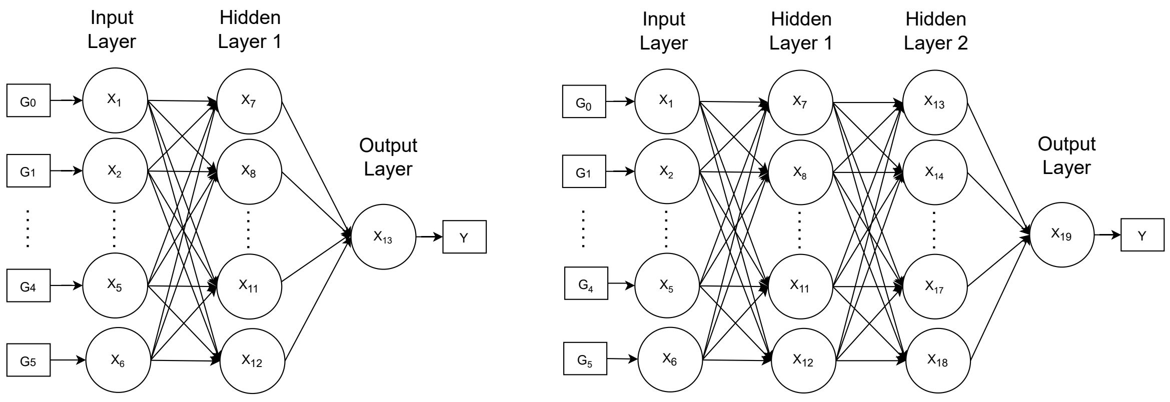


Figure 85: ANN with 6 sensors

Different types of ANNs were used in this approach, such as ANN without any hidden layer, with 1 hidden layer, 2 hidden layers, 3 hidden layers, and 4 hidden layers. Some ANNs used to predict the concentration of CH_4 are illustrated in figures 86a and 86b. The implementation of several ANNs was intended to compare the performance of various machine learning models in the CH_4 concentration prediction process.



(a) ANN with 1 hidden layers (b) ANN with 2 hidden layers

Figure 86: Different types of ANNs to predict CH_4

To implement all these algorithms a deep learning API written in Python, and running on the top of the machine learning platform TensorFlow, was used (Keras). This high-level API provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity. The code 10 demonstrates the model with 4 hidden layer implementation, responsible for detecting various concentration levels of CH_4. In this implementation, it is possible to verify the presence of 4 hidden layers, and the activation functions used (ReLU function) to calculate the output of each neuron in the ANN. This code was developed to promote the implementation of the two neural network models illustrated in figure 86 since it is only necessary changing the number of hidden layers in the code to obtain different models. The code also shows the compile method used to configure the model for the training process and the fit method used to train the model for a fixed number of epochs (iterations on a dataset). All these methods belong to the Keras API.

Listing 10: ANN with 4 hidden layers

```
class ANN(tf.keras.Model):

   def __init__(self):
      super().__init__()
      self.inp = tf.keras.layers.Dense(units=6, input_shape=[6], use_bias=True, activation="
          ↪ relu") # input layer
      self.h1 = tf.keras.layers.Dense(units=6, input_shape=[6], use_bias=True, activation="
          ↪ relu") # 1 hidden Layer
      self.h2 = tf.keras.layers.Dense(units=6, input_shape=[6], use_bias=True, activation="
          ↪ relu") # 2 hidden Layer
      self.h3 = tf.keras.layers.Dense(units=6, input_shape=[6], use_bias=True, activation="
          ↪ relu") # 3 hidden Layer
      self.h4 = tf.keras.layers.Dense(units=6, input_shape=[6], use_bias=True, activation="
          ↪ relu") # 4 hidden Layer

   def call(self, inputs):
      x = self.inp(inputs)
      x = self.h1(x)
      x = self.h2(x)
      x = self.h3(x)
      x = self.h4(x)
      return self.out(x)

model = ANN()
model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(X_train, y_train, epochs=1000)
```

The layers used in this implementation were the dense layers by Keras API, characterized by being deeply connected neural network layers. The neurons in this type of layer receive the output of every neuron of the preceding layer. This type of layers are commonly used in the artificial neural network world.

After training the various ANN models, it is necessary to evaluate their performance, allowing the selection of which model is the best for predicting the concentration of CH_4. The models' performance was calculated using the RMSE and AIC algorithms. Their implementation is represented in code 11. The parameter K in the AIC implementation will be different for each ANN, since this parameter will take as a value the number of parameters of the model. In code 11, K is equal to 217, which is the number of parameters (weights + bias) that the ANN has with 4 hidden layers.

Listing 11: RMSE and AIC Implementation

```
y_pred = model(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE : ", rmse)

# AIC
K = 217
L = rmse
AIC = -2*np.log(L) + 2*K
print("AIC : ", AIC)
```

It is important to note that the training and test datasets correspond to the time range between 2 and 7 minutes (enough time for the sensor to react to different CH_4 concentrations). During the training process of the machine learning models (ANN) with the CH_4 dataset, the final two concentrations (41000 ppm and 36900 ppm) were excluded. This was because the ANOVA analysis revealed sensor saturation for these concentrations, as evidenced by a p-value greater than 0.76 for the last three concentrations.

After training several ANNs, graphs were generated to depict the relationship between actual and predicted values. Figures 87 and 88 illustrate this relationship for ANNs with 1 and 4 hidden layers, respectively, alongside a linear regression line that indicates the model's trend (a slope of 1 denotes an ideal model).

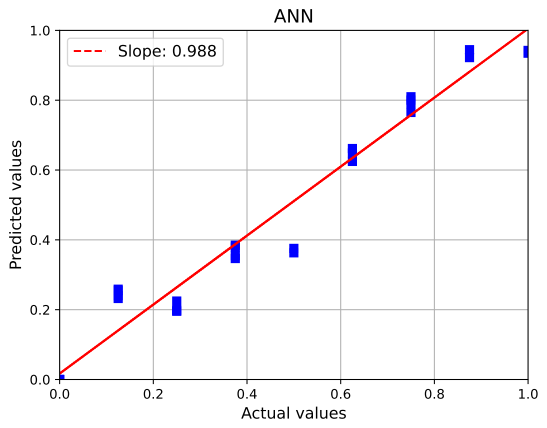


Figure 87: ANN with 6 sensors and 1 hidden layer

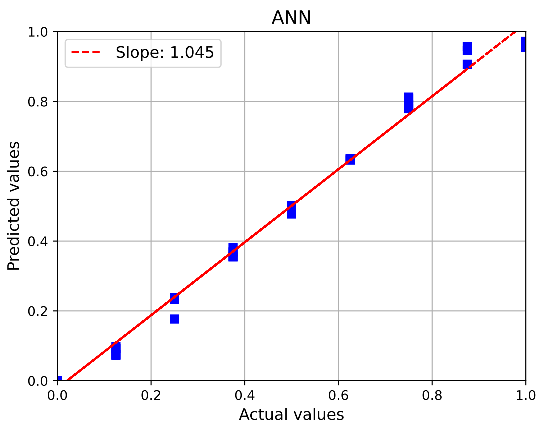


Figure 88: ANN with 6 sensors and 4 hidden layers

5.12 Generalized Additive Models (GAM)

Python Generalized Additive Models (pyGam) is a Python library for fitting and analyzing generalized additive models (GAMs). It is designed to make it easy to specify and fit GAMs in Python, providing several tools to study the model's results. This library provides several tools for evaluating the fit of a GAM, including functions for computing residuals, predicted values, and computing models performance metrics such as mean squared error and R^2. These tools are supplied by the summary method in the class GAM. The code 12 is responsible for the implementation of the GAM model in the CH_4 dataset collected previously. The input space in this implementation is the same used in figure 85, which is characterized by having all the input sensor's values.

Listing 12: GAM Implementation

```python
from BME688_Dataset import X_train, X_test, y_train, y_test, X_valid, y_valid
from pygam import LinearGAM, l, s

gam = LinearGAM(s(0) + s(1) + s(2) + s(3) + s(4) + s(5)).gridsearch(X_train, y_train)

print(gam.summary())
```

As can be seen, this implementation uses the *LinearGAM* class, which is a customized GAM class version with the distribution and link parameters set to normal and identity, respectively (see more in [28]). The distribution parameter was set to normal because the data collected to train this model presented a normal distribution. The functional form of the *LinearGAM* class is built by 6 spline terms, each for a unique feature (6 sensors resistance). Figure 89 shows the relation between actual and predicted values for the GAM model.

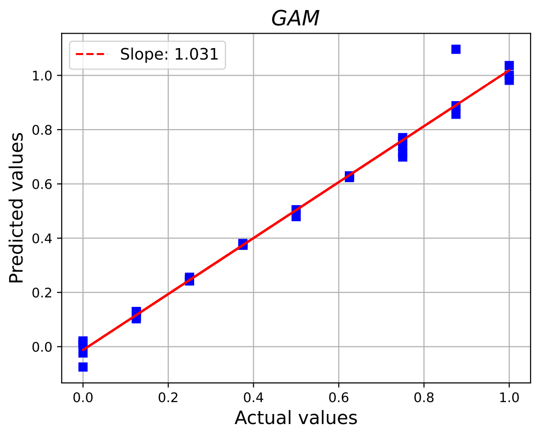


Figure 89: GAM with 6 sensors

5.13 TensorFlow Lattice

As mentioned earlier in the theory chapter, TensorFlow provides tools to train Lattice-based machine learning models. One of those tools is the lattice layer available in [31]. The code 13 shows the training process implemented in the CH_4 dataset used in previous subchapters with TensorFlow Lattice layers.

Listing 13: TensorFlow Lattice Implementation

```
lattice_sizes = [2, 2, 2, 2, 2, 2]

class TFL(tf.keras.Model):

   def __init__(self):
      super().__init__()
      self.out = tfl.layers.Lattice(lattice_sizes=lattice_sizes, monotonicities=["none", "none
         ↪ ", "none", "none", "none", "none"], output_min=0.0, output_max=1.0)

   def call(self, inputs):
      return self.out(inputs)

model = TFL()

model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(X_train, y_train, epochs=1000)
```

In this implementation, it is possible to see that the model is built with one layer, characterized by a null monotonicity for all the input sensor's values. The monotonicity specifies if the output (concentration level) should increase/decrease for the inputs (figure 83 show that relation between concentration and resistance is non-monotonic). The resulted lattice-based ML model has the shape of a 6-dimensional hypercube, whose orthographic projection is shown in figure 90.

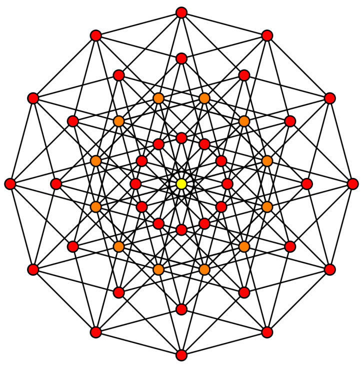


Figure 90: Six-dimensional hypercube with orthographic projection (based on [1])

This model has 64 parameters since it is built with ten features (The timestamp, the eight sensor's resistance, and the heater step). The number 64 results from the operation 2^6. Figure 91 shows the relation between actual and predicted values for the lattice model.

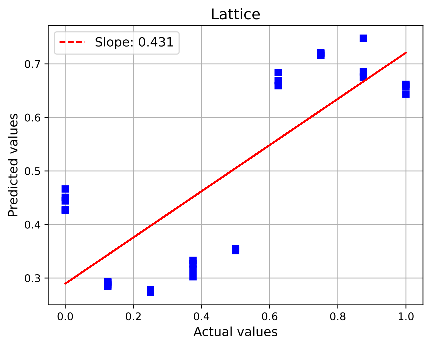


Figure 91: Lattice with 6 sensors

5.14 Regressors Tests and Results

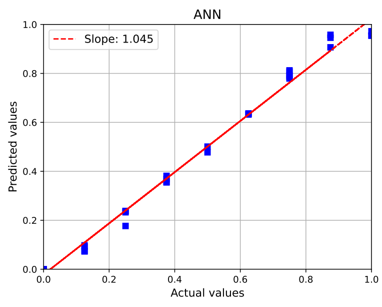
After training all the regressors mentioned earlier, it is necessary to perform some performance tests on these, to be able to select the model that best fits the user's needs. Therefore, the next section will describe all test procedures performed in the developed models and their respective results. As mentioned before, after the dataset was collected, it was split into two subsets: The train set and the test set. The last one was used to evaluate the model performance. Since the classifiers and regressors were trained on different types of datasets (For classifiers, ambient air and CO_2 samples were taken, while for regressors, CH_4 samples in different concentration levels were taken), it does not make sense to execute the performance comparison between classifiers and regressors.

After performing performance measurements and comparisons on the classifiers, it is necessary to evaluate the regressors. Since classifiers and regressors behave differently, the methods used for the regressors will be the RMSE and the AIC. Table 8 shows the RMSE and AIC scores, performed in the test set, of all the regressors.

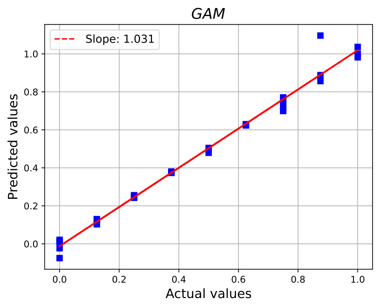
Table 8: Regressor's performance results

Regressors	**RMSE**	**AIC**
ANNs		
Zero hidden layers	0.189	101.320
One hidden layer	0.093	186.734
Two hidden layers	0.078	271.082
Three hidden layers	0.050	355.967
Four hidden layers	0.038	440.496
GAM	0.043	184952.834
Lattice-based Model	0.220	129.509

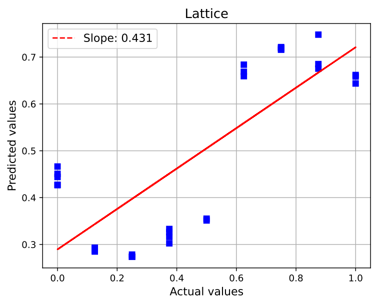
In addition to analyzing the results obtained from calculating the RMSE and AIC, it is necessary to also study the real/predicted value relationship graphs, in order to select a model that is better capable of detecting CH_4. Figure 92 displays the results of the last three regressors from table 8.



(a) ANN with 4 hidden layers



(b) GAM



(c) Lattice

Figure 92: Different types of regressors to predict CH_4

Chapter 6

Conclusion

Machine learning (ML) models have become more widely used to increase the precision and effectiveness of gas detection systems. Gas detection systems are constantly used in industry to monitor the levels of gases potentially harmful to the atmosphere and humans.

One commonly used ML model in the gas detection field is the artificial neural network (ANN). These ANNs can be trained with data collected from gas sensors to recognize patterns in some specific gases. Once trained, these models can be used to analyze some systems. Another popular ML model used in gas detection systems is the support vector machine (SVM). This technique is a supervised learning algorithm that can be used for classification and regression. This model can also be trained with the help of gas sensor data to identify specific gases. Other techniques can be used to improve the accuracy and efficiency of gas detection systems, such as Random Forests, Decision Trees, or Lattice-based models. All these models were implemented and study in this dissertation.

Throughout this dissertation, all these algorithms were trained using readout data from the BME688 gas sensor. The choice of this sensor was, according to BOSCH Sensortec, developed to perform reading functions with low power consumption. This power consumption is one of the main objectives in the gas detection field, since gas systems must be able to operate with little or no maintenance (i.e., they must have a very significant lifetime). This goal should be considered, always keeping in mind the quality and accuracy factor that the gas detection system has in monitoring the surrounding gases. This dissertation has always sought to analyze the performance of various ML algorithms trained on low-power gas sensors.

One of the challenges in this dissertation was to understand the BME688 gas sensor to be able to analyze its readings and thus train algorithms capable of detecting the presence of certain gases. The first approach of this dissertation was to develop a classifier that could distinguish two types of gases, followed by the development and study of regressors responsible for predicting the concentration level of a specific gas. In the end, each model was evaluated regarding its performance, allowing the selection of the model which best suited for gas detection with this type of sensor. A crucial feature that allows the collection of distinct responses from very similar gases was combining the response of several BME688 sensors. In other words, the presence of several BME688 sensors allows the creation of a unique pattern that characterizes a particular gas. Another important aspect to take away from all the work conducted is the contribution of a heating profile in accelerating the response of the BME688 sensor. After studying the BME688 sensor, the classifiers and regressors were trained to select the best model for gas detection. Regarding the classifiers, according to the performance tests, the models with the best results were the decision tree and random forest, while the regressors it was the ANN with 4 hidden layers.

6.1 Future Work

Within the scope of this dissertation, one of the future works that would be important to perform corresponds to the exploration of more classifiers than those implemented. This work would allow studying and selecting with more certainty the classifier that performs best when it comes to gas detection. The same procedure should also be performed concerning the regressors. In this case, an interesting idea could be to study a model that consists of the junction between a neural network with lattice-based models, in which lattices would replace neurons in a neural network. The tests performed should be improved to explore and study the runtime and the spatial cost of the models. One way to accomplish this would be to implement the various models on various processors, followed by a detailed analysis of energy consumption and time. Another aspect that would also be important to evaluate is another approach to the choice of sensor used for the development of the models (exploring other sensor operating technologies that could provide reduced power consumption or even better quality data collection).

References

[1] *6-cube - Wikipedia*. url: https://en.wikipedia.org/wiki/6-cube (visited on 10/08/2022).

[2] S. Asiri. *Machine Learning Classifiers. What is classification? | by Sidath Asiri | Towards Data Science*. url: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623 (visited on 10/11/2022).

[3] G. Brain. *TensorFlow basics | TensorFlow Core*. url: https://www.tensorflow.org/guide/basics (visited on 10/09/2022).

[4] Components101. *What is a Gas Sensor? Construction, Types & Working of Gas Sensors*. 2021. url: https://components101.com/articles/introduction-to-gas-sensors-types-working-andapplications (visited on 06/06/2022).

[5] M. Developers. *Matplotlib — Visualization with Python*. url: https://matplotlib.org/ (visited on 08/09/2022).

[6] N. Developers. *NumPy documentation — NumPy v1.23 Manual*. url: https://numpy.org/doc/stable/ (visited on 10/09/2022).

[7] P. Developers. *pandas - Python Data Analysis Library*. url: https://pandas.pydata.org/ (visited on 08/09/2022).

[8] S.-L. Developers. *Getting Started — scikit-learn 1.1.2 documentation*. url: https://scikit-learn.org/stable/getting_started.html (visited on 10/09/2022).

[9] S. Developers. *Welcome to Spyder's Documentation — Spyder 5 documentation*. url: https://docs.spyder-ide.org/current/index.html (visited on 10/09/2022).

[10] E. Eichhammer. *Qt Plotting Widget QCustomPlot - Introduction*. 2022. url: https://www.qcustomplot.com/index.php/introduction (visited on 10/06/2022).

[11] Espressif. *SPI Master Driver - ESP32 - — ESP-IDF Programming Guide latest documentation*. 2022. url: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/spi_master.html (visited on 10/09/2022).

[12] S. Feng, F. Farha, Q. Li, Y. Wan, Y. Xu, T. Zhang, and H. Ning. “Review on Smart Gas Sensing Technology.” In: *Sensors 2019, Vol. 19, Page 3760* 19 (17 Aug. 2019), p. 3760. issn: 1424-8220. doi: `10.3390/S19173760`. url: `https://www.mdpi.com/1424-8220/19/17/3760/htmhttps://www.mdpi.com/1424-8220/19/17/3760`.

[13] A. F. Gad and F. E. Jarmouni. *Introduction to Deep Learning and Neural Networks with Python*. 2021, pp. i–ii. doi: `10.1016/b978-0-323-90933-4.09993-9`.

[14] J. B. Gomes, J. J. Rodrigues, R. A. Rabêlo, N. Kumar, and S. Kozlov. “IoT-Enabled Gas Sensors: Technologies, Applications, and Opportunities.” In: *Journal of Sensor and Actuator Networks 2019, Vol. 8, Page 57* 8 (4 Dec. 2019), p. 57. issn: 2224-2708. doi: `10.3390/JSAN8040057`. url: `https://www.mdpi.com/2224-2708/8/4/57`.

[15] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2017. url: `http://oreilly.com/safari`.

[16] *HYDROGEN | CAMEO Chemicals | NOAA*. url: `https://cameochemicals.noaa.gov/chemical/8729` (visited on 04/05/2022).

[17] IBM. *What are Neural Networks? | IBM*. url: `https://www.ibm.com/cloud/learn/neural-networks` (visited on 10/06/2022).

[18] javapoint. *Artificial Neural Network Tutorial - Javatpoint*. url: `https://www.javatpoint.com/artificial-neural-network` (visited on 11/05/2022).

[19] L. Klibanov and P. Bold. *Preliminary analysis of Bosch BME688 4-in-1 environmental sensor with AI*. 2021. url: `https://www.linkedin.com/pulse/preliminary-analysis-bosch-bme688-4-in-1-sensor-ai-lev-klibanov` (visited on 03/06/2022).

[20] R. Kwiatkowski. *Gradient Descent Algorithm — a deep dive | by Robert Kwiatkowski | Towards Data Science*. 2021. url: `https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21` (visited on 11/06/2022).

[21] K. LARSEN. *GAM: The Predictive Modeling Silver Bullet | Stitch Fix Technology – Multithreaded*. 2015. url: `https://multithreaded.stitchfix.com/blog/2015/07/30/gam/` (visited on 11/06/2022).

[22] *METHANE | CAMEO Chemicals | NOAA*. url: `https://cameochemicals.noaa.gov/chemical/8823` (visited on 04/05/2022).

[23] Microsoft. *Documentation for Visual Studio Code*. url: `https://code.visualstudio.com/docs` (visited on 10/09/2022).

[24] Prashant. *Computational Complexity of ML algorithms | by Prashant | Analytics Vidhya | Medium*. url: https://medium.com/analytics-vidhya/computational-complexity-of-ml-algorithms-1bdc88af1c7a (visited on 11/10/2022).

[25] R. Pupale. *Support Vector Machines(SVM) — An Overview | by Rushikesh Pupale | Towards Data Science*. 2018. url: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989 (visited on 06/12/2022).

[26] B. Sensortec. *BME688 Datasheet*. 2022. url: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme688-ds000.pdf (visited on 03/06/2022).

[27] B. Sensortec. *GitHub - BoschSensortec/BME68x-Sensor-API: Common Sensor API for the BME680 and BME688 sensors*. 2022. url: https://github.com/BoschSensortec/BME68x-Sensor-API (visited on 10/06/2022).

[28] D. Servén. *A Tour of pyGAM — pyGAM documentation*. 2018. url: https://pygam.readthedocs.io/en/latest/notebooks/tour_of_pygam.html (visited on 10/09/2022).

[29] TechBlog. *Artificial Neural Network | Types | Feed Forward | Feedback | Structure | Perceptron | Machine Learning | Applications - Tech Blog*. 2021. url: https://msatechnosoft.in/blog/artificial-neural-network-types-feed-forward-feedback-structure-perceptron-machine-learning-applications/ (visited on 06/05/2022).

[30] TensorFlow. *TensorFlow Lattice (TFL)*. url: https://www.tensorflow.org/lattice/overview (visited on 10/05/2022).

[31] TensorFlow. *tfl.layers.Lattice | TensorFlow Lattice*. url: https://www.tensorflow.org/lattice/api_docs/python/tfl/layers/Lattice (visited on 03/09/2022).

[32] I. F. USA. *Operating principle Catalytic-type gas sensor*. 2018. url: https://www.figarosensor.com/technicalinfo/principle/catalytic-type.html (visited on 06/06/2022).

[33] I. F. USA. *Operating principle Electrochemical-type gas sensor*. 2018. url: https://www.figarosensor.com/technicalinfo/principle/electrochemical-type.html (visited on 03/06/2022).

[34] I. F. USA. *Operating principle MOS-type gas sensor*. 2018. url: https://www.figarosensor.com/technicalinfo/principle/mos-type.html (visited on 06/06/2022).

[35] D. Wilimitis. *The Kernel Trick in Support Vector Classification | by Drew Wilimitis | Towards Data Science*. 2018. url: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f (visited on 10/07/2022).

[36] S. Yale. *Linear Regression*. url: `http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm` (visited on 10/06/2022).

[37] A. Zajic. *Introduction to AIC — Akaike Information Criterion | by Alexandre Zajic | Towards Data Science*. 2019. url: `https://towardsdatascience.com/introduction-to-aic-akaike-information-criterion-9c9ba1c96ced` (visited on 09/05/2022).