VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

# Using Soft Computing Methods to Identify Operational Technical statuses in Smart Home within IoT

2020

Ismael Abu-jadur García

I hereby declare that this bachelor's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, April 2020

[illegible]

## Abstract

This bachelor project aims to focus on the occupancy monitoring of Smart Home Care (SHC) to optimize functions that SHC provides and optimize energy waste. The primary cause of this project, being focused on this topic, is that, so far, most of the software developed for SHC is based in room occupancy. To know the presence of persons without being invasive, we must use indirect methods to predict it. The project is based on the prediction of the CO2 waveform based on the use of sensors that measure the indoor temperature and the relative humidity. Support Vector Machine (SVM) is employed to predict the presence of a person with high accuracy.

**Keywords**: IoT, Smart Home, prediction of room occupancy, presence of person monitoring

# Contents

## List of symbols and abbreviations

| | | |
|---|---|---|
| SH | – | Smart Homes |
| SHC | – | Smart Home Care |
| IB | – | Intelligent Buildings |
| IoT | – | Internet of Things |
| HVAC | – | Heating, ventilation, air conditioning |
| SVM | – | Support Vector Machine |
| SC | – | Soft Computing |
| HC | – | Hard Computing |
| IM | – | Indirect Methods |
| DM | – | Direct Methods |
| LDA) | – | Linear Discriminant Analysis |
| MNLR | – | Multiple Non-linear Regression |
| DNN) | – | Deep neural network |
| AWS | – | Amazon Web Service |
| MQTT) | – | Message Queue Telemetry Transport |

# List of Figures

## List of Tables

# Code Listings

# 1 Introduction

In this section, the project and the problem to solve will be introduced. Also, the motivation and the actual situation of Smart Homes will be explained.

## 1.1 Definitions

In the following we define and explain the terms used in this project.

### 1.1.1 Smart Homes

Smart Homes (SH) are buildings where technical functions like heating, ventilation, air conditioning (HVAC), and lighting are controlled and automatized [1]. The idea of SH is to allow and let the automatic systems make decisions based on the environment [2], behavior, and occupancy of the house.

### 1.1.2 Intelligent Buildings

The concept of Intelligent Buildings (IB) is hard to define because there is not a single definition of it. In [3], Shengwei Wang described and categorized the different types of IB. He formalized three categories.

- **Performance-based definitions**: "Defines an IB as a building created to give its users the most efficient environment, at the same time, the building utilizes and manages resources efficiently and minimizes the life costs of hardware and facilities." [3]
- **Services- based definitions**: "Defines an IB is a building with the service functions of communication, office automation, and building automation and is convenient for intelligent activities. Services to users are emphasized." [3]
- **System-based definitions**: "States that IB provide building automation, office automation and communication network systems, and an optimal composition integrates the structure, system, service and management, providing the building with high efficiency, comfort, convenience and safety to users." [3]

However, generically, in this project, we will describe IB as buildings that use infrastructure to create more efficient and productive buildings. By the automatization and remote control of multiple tasks as HVAC, lighting, and others, it is allowed to increase the performance of the occupants.

### 1.1.3 Soft Computing and Hard Computing

Soft computing (SC) is a set of methods that allow solving complex real-life problems reducing costs with high consistency and robustness [4]. Usually, it deals with problems that cannot be solved by mathematical and exact methods (Hard Computing) because these methods will take an extremely long time to obtain results.

SC is an alternative to Hard Computing (HC) to solve problems where HC will consume much time to provide suitable solutions [5](17). In SC, the solving of problems is imprecise, tolerant, and uncertain. However, most of the solutions provided can be usable. There different methods to get solutions to problems. Nevertheless, it is crucial to know the “No free lunch” theorem [6], which means that there is no universal method to solve every problem, meaning that each method will fit better for some determined kind of problems. Besides, the selection of the method to get solutions will depend on the problem to solve. Using SC is a good option as there are large quantities of data to process and compared to HC the time to train the models will be shorter.

### 1.1.4 Indirect and direct methods

We define Indirect Methods (IM) as the methods which can predict the occupancy without being invasive. These methods are user friendly, and they do not involve cameras, microphones, or other devices that invade the privacy of the user. The difference between IM and Direct Methods (DM) is straightforward. DM can use all of these invasive devices being easier to predict the occupancy, but for private envirómments, IM are way better.

## 1.2 Project description

The SH market made 51,75 billion Euros in 2018, and it is expected to grow twice as big for 2023. Europe represents 14% of this market and is expected to be 30,9% in 2023. Knowing these statistical data, we can assume that the SH is the future for buildings, making them remarkably important with a new level of technology development [7] [8]. That is to say, the more optimal and improved energy efficiency [9] [10], the more benefits. These benefits are both economic and ecologic, making SH more acceptable with what the customer gets and its reliability.

One of the motivations is that in SH, some ways to detect the occupancy are based on direct methods. These methods do not respect the privacy and intimacy, making people feel uncomfortable [11] [12]. Furthermore, these methods raise way too much the costs and usually are difficult to adapt to future changes [10] [13] [14]. The use of mobile phones for detection is a great choice but assumes that every person carries their mobile phone everywhere [15]. Nevertheless, working with CO2, relative humidity, and indoor temperature sensors allow making SH systems more affordable and less invasive. Another motivation for this project is the care of the elderly or with disabilities [16]. The detection and recognition of the elderly or with disabilities behavior allows them to live independently, as these IoT systems can be remotely controlled and they can detect critical situations before they happen [14] [17].

The detection of occupants is a hard task, and it can be performed in many ways, and this is because human presence changes its environment, and this can be measured with different sensors [2] [11]. Thus, as before was said, these systems are expensive and do not have good scalability. This task can be focused on the detection of occupants based on the prediction of CO2 concentration [18] [19] [20], as it is an excellent way to detect the presence, cheap and not invasive.

The use of statistical models to obtain results in the problem of prediction of human presence will have more accurate results. Methods such as Linear Discriminant Analysis (LDA), Classification and Regression Trees (CART), and Random Forest (RF) have accuracies from 95 to 99% [21]. In the same way, clustering methods (K-means, Self-Organizing Maps), to find patterns in the data to predict the occupancy, will obtain positive results [22]. Even further, the use of models like Multiple Non-linear Regression (MNLR) and Deep neural network (DNN) throws small percentages of error (Between 2,37% and 10,34%) [23].

Our work aims to determine the efficiency of the SVM method, believing that it is innovative and not commonly used in this field [24]. SVM works for classification and regression problems based on the kernel trick, which transforms the data allowing it to find optimal solutions.

# 2 Project development

In this section, we strip the project in different phases, and we summarize all the chapters in order to get a better comprehension of it.

## 2.1 Analysis of requirements

In this phase, the problem will be analyzed and investigated so that the algorithm (SVM) can be developed accordingly to obtain better results. The objective of this phase is to understand and work with the data gathering systems from VSB University for a better understanding of the available data: where they come from, what relationships exist, among others.

## 2.2 Data preprocessing and algorithm design

The algorithm design will be developed out generically, to solve the main problem, and refine the details later, thus being able to obtain optimal results. The data provided will be transformed to be optimal for processing, selecting essential characteristics and attributes, and dropping those that do not provide information.

## 2.3 Coding

The design of the algorithm will be coded by adding all the details that are necessary for its optimal operation. To accomplish the coding of the algorithm, the use of the laptop, Dell XPS 13 9360 will be done.

## 2.4 Execution and verification

Once we finish the codification of the algorithm, it is essential to carry out the tests with the available data and see that works and operates optimally.

## 2.5 Depuration

In the case of finding errors in the execution or miss functionality, we will fix them, and we will repeat the previous step. It would be a cyclical process until the execution is correct and optimal.

## 2.6 Benchmarking and experimentation

With the developed algorithm, we will work with the available data, proposing different environments, and seeing different results for later comparison.

## 2.7 Documentation

This step is part of all the other phases. However, in the end, we will work with the results obtained to develop the final discussion and conclusions of the project.

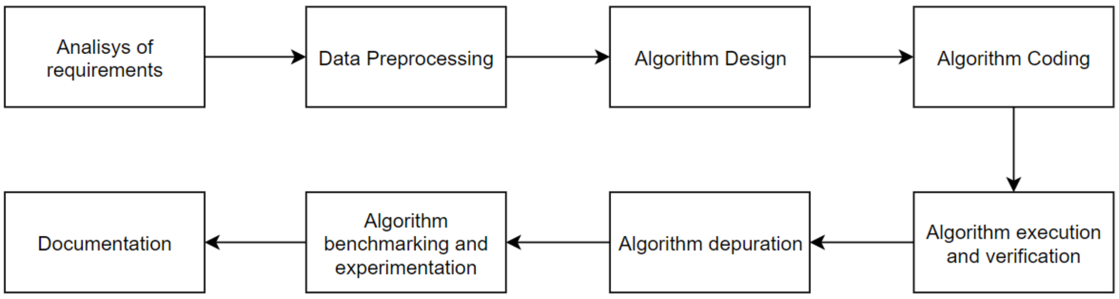


Figure 1: Flow of the different project Phases

## 2.8 Outline

This thesis is organized as follows:

- In **Section 3**, Internet of Things (IoT), Cloud Services for IoT and possible theoretical implementations are explained.
- In **Section 4**, theoretical foundations of SVM (Kernels, hyperplanes, margins) are introduced.
- In **Section 5**, we give a detailed explanation of the implementation, experimentation and the faced challenges. Furthermore, Python and libraries used are explained.
- In **Section 6**, we show the process of experimentation made with different configurations of the parameters and compare the different models obtained.
- In **Section 7**, we resume the project and make some conclusions of it, although we explain future work that could be done.

# 3 Internet of Things

In this section, we explain what the Internet of Things (IoT) is, cloud services that offer interconnection with IoT and possible theoretical implementations for the connection between KNX and these cloud services, or the SVM developed in this project.

## 3.1 Definition of IoT

The definition of IoT could be the grouping and interconnection of devices and objects through a network (either private or the Internet), where all of them could be visible and interact. Regarding the type of items or devices, they could be any, from sensors and mechanical devices to everyday objects such as the microwave, coffee machines, or clothing. Everything could be connected to the Internet and interact without the need for human intervention.

Concepts closely related to IoT can be Smart Cities and Smart Buildings, which were defined in section 1.1. In these concepts, IoT devices are used to improve traffic control, control of water, and heating supplies in a building, management of public transport, etcetera.

As it is possible to see, its scope is vast, and more and more devices are appearing every day that make this technology possible. Before-mentioned technology associated with the IoT allows data to be collected and sent to the network for analysis, or even to perform a prior study and then send it to the network.

Writing about technology and implementations in IoT means writing about solutions proposed by different companies and that those solutions are in constant evolution. There is not only one technology, but many of them must be analyzed to adapt them to the specific solution that we want to develop.

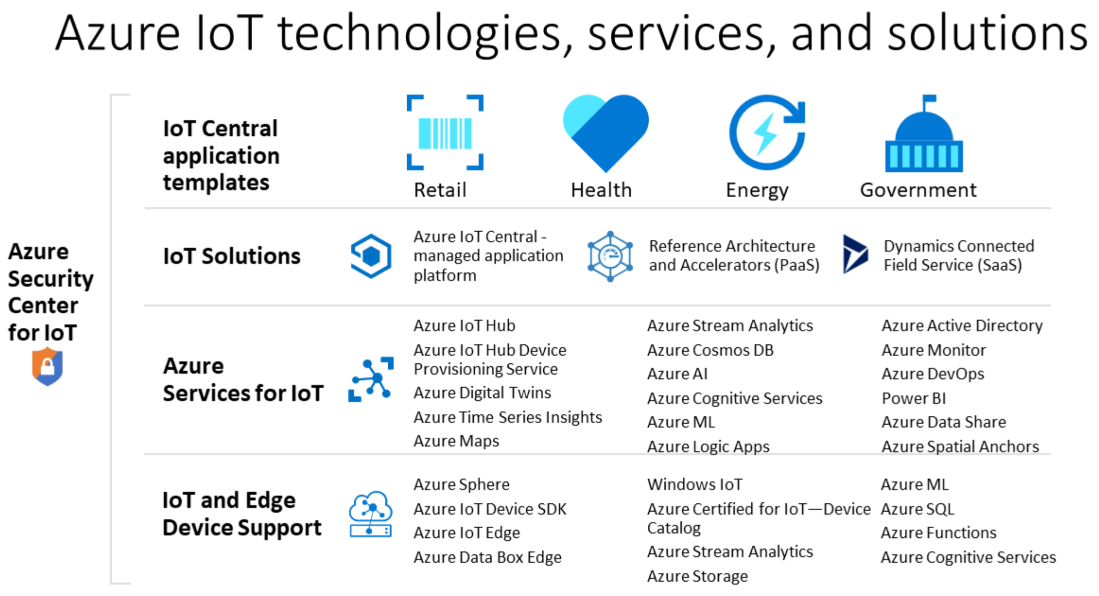
## 3.2 Cloud Services within IoT

The concept of IoT is based on the massive interconnection of devices and objects to the network. This generates an enormous volume of data, so efficient management of the same will allow the development of applications and services that improve the quality of life of society. Due to the large amounts of information generated by the Internet of Things, its relationship with cloud computing arises.

Cloud computing presents exciting characteristics for the Internet of Things concept since it allows ubiquitous access, storage, and management of the data collected by the devices, offering scalability and flexibility. This is a great advantage when developing IoT applications in different areas. In the following subsections, some of these platforms are presented, including the explanation of their most characteristic points.

### 3.2.1 Azure IoT

This platform created by Microsoft offers the users the ability to develop IoT solutions according to their needs, counting on it with various technologies and IoT solutions from Azure. Figure 2 shows a summary and classification of these technologies and solutions. The most critical points of the azure architecture



Azure IoT technologies, services, and solutions

Azure Security Center for IoT

| | | | | |
|---|---|---|---|---|
| IoT Central application templates | Retail | Health | Energy | Government |
| IoT Solutions | Azure IoT Central - managed application platform | Reference Architecture and Accelerators (PaaS) | Dynamics Connected Field Service (SaaS) | |
| Azure Services for IoT | Azure IoT Hub<br>Azure IoT Hub Device Provisioning Service<br>Azure Digital Twins<br>Azure Time Series Insights<br>Azure Maps | Azure Stream Analytics<br>Azure Cosmos DB<br>Azure AI<br>Azure Cognitive Services<br>Azure ML<br>Azure Logic Apps | Azure Active Directory<br>Azure Monitor<br>Azure DevOps<br>Power BI<br>Azure Data Share<br>Azure Spatial Anchors | |
| IoT and Edge Device Support | Azure Sphere<br>Azure IoT Device SDK<br>Azure IoT Edge<br>Azure Data Box Edge | Windows IoT<br>Azure Certified for IoT—Device Catalog<br>Azure Stream Analytics<br>Azure Storage | Azure ML<br>Azure SQL<br>Azure Functions<br>Azure Cognitive Services | |

Figure 2: Azure IoT technologies and solutions [25]

are defined below:

- IoT devices refer to the IoT devices that register and connect to the cloud to send and receive data. IoT Edge devices correspond to devices that are capable of processing data on the same device, which is achieved thanks to Azure IoT Edge.
- Azure Cloud Gateway consists of a cloud gateway for the connection and sending of data from the devices, including their management and control capabilities. This task is done thanks to the Azure IoT Hub technology.
- Stream processing is about analyzing and processing large data streams. One of the tools that take this task is Azure Stream Analytics. On the other hand, Azure Machine Learning allows machine learning after analyzing this data. However, it does not have support for SVM for regression.
- Data transformation consists of the treatment of the data and its modification if necessary. Azure Functions technology is responsible for this process.

### 3.2.2 Amazon Web Services IoT

Amazon Web Services IoT (AWS IoT) is a service whose objective is to be able to obtain, save and interpret data collected from devices connected to the Internet and can additionally create applications meant for users so that they can manage these devices. To accomplish this, AWS establishes two-way communication between sensors, actuators or smart devices, and the AWS cloud. Some of the factors that define AWS IoT are described in the following:

- Infrastructure Services that comprise virtual servers, web-building applications, load balancing, and autoscaling, block, and integrated storage, virtual private networks, direct connections, and DNS services.
- The Platform Services integrate analysis tools, data transmission, machine learning, equipment virtualization, notifications, test environment, mobile development and services for IoT.
- Operational Services include the implementation and management of the source code, templates, management and monitoring tools, resource auditing, firewalls and application services such as notifications, application streaming, workflow queues.

More specific characteristics of AWS are:

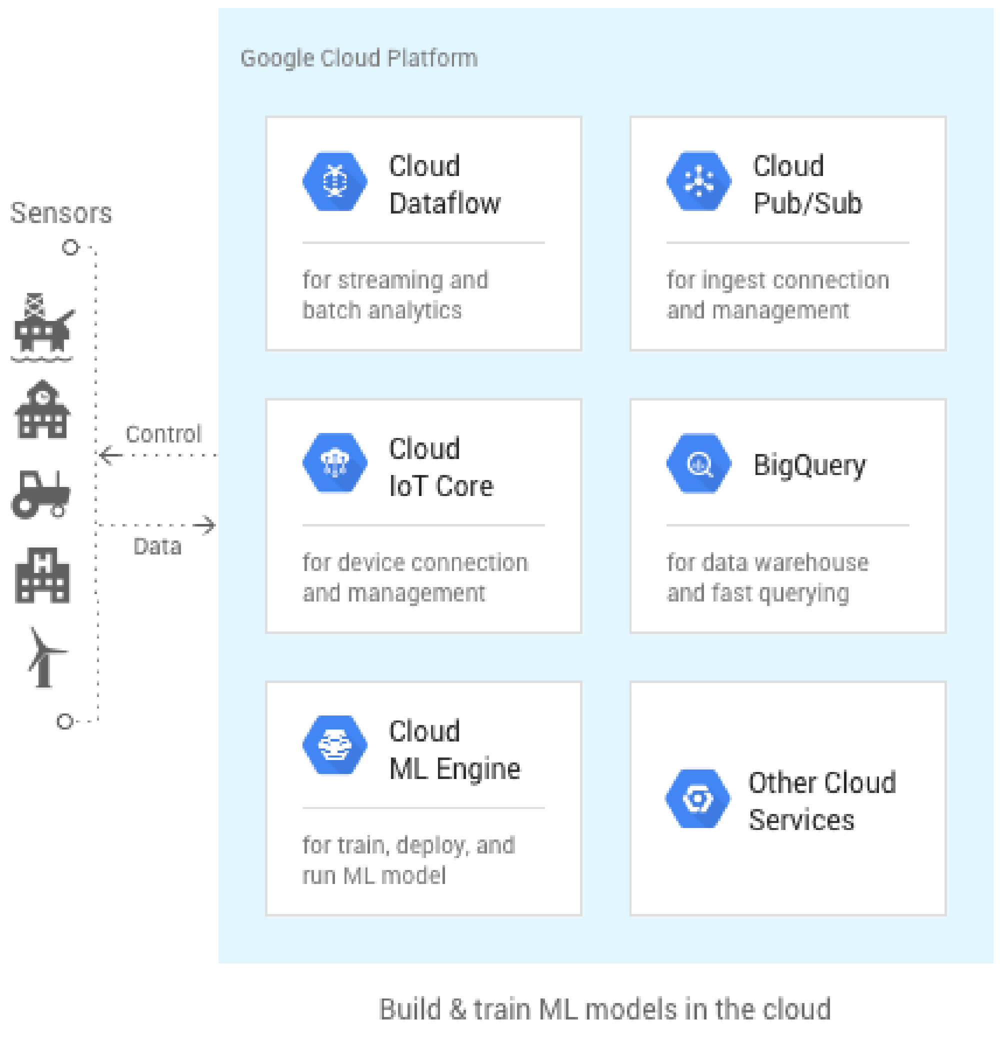
- It has a gateway device that enables secure and effective communication with AWS IoT.
- It has a message agent, which uses MQTT for publication and reception between AWS IoT devices and applications.
- It has a service called Device Shadow with which device status information is published so that applications or other connected devices can use it.
- It proposes a Jobs Service, with which you can designate a set of remote procedures for one or more devices to be executed.

AWS is integrated with other AWS services such as Amazon S3 (scalable AWS cloud storage), Amazon DynamoDB (NoSQL databases), and AWS Lambda (code execution in Amazon EC2 virtual servers) among others.

### 3.2.3 Google Cloud IoT

This platform is made up of a set of useful tools. These tools make possible the device connection as well as the processing, storage, and analysis of data in the cloud. Below are the main services provided by Google Cloud IoT:

- **Cloud IoT Core** is the main service of this platform. Its objective is the connection, management, and control of data from different devices, offering security and simplicity in the process. Combined with other Cloud Cloud IoT services, a complete solution is achieved, including the collection, processing, analysis, and visualization of IoT data.



Google Cloud Platform
Sensors
Control
Data
Cloud Dataflow
for streaming and batch analytics
Cloud Pub/Sub
for ingest connection and management
Cloud IoT Core
for device connection and management
BigQuery
for data warehouse and fast querying
Cloud ML Engine
for train, deploy, and run ML model
Other Cloud Services
Build & train ML models in the cloud

Figure 3: Basic structure of how Google Cloud IoT works [26]

- **Cloud Pub/Sub** is a messaging service that allows the sending and receiving of messages between independent applications, as well as the distribution of data between projects and applications in hybrid environments.
- **BigQuery** tool is defined by being a serverless data warehouse that uses the SQL language.
- **Cloud Dataflow** manages batch and real-time data streaming. It is defined by its reliability and high processing capacity since it does not need a server.
- **Cloud Machine Learning Engine** (Cloud ML Engine) gives preparation and prediction services based on the data analysis that has been carried out. This enables machine learning models to be developed and used in production environments.

### 3.2.4 ThingSpeak

ThingSpeak is an open-source IoT analysis platform that allows the collection of the data measured by sensors, its storage in the cloud, and its visualization and analysis. Besides, it makes it possible to prototype and build IoT systems without having to configure servers or develop web software[27]. In figure 4 is shown how the basic structure of how ThingSpeak works. The main abilities of this platform can be summed as follows:

- It has a simple configuration of the devices that take the data collection and send them to the cloud using IoT protocols.
- It offers an analysis of the stored data. The feature that sets this platform apart from others is that it provides access to MATLAB. This allows the data to be visualized in graphs, diagrams, or meters and facilitates obtaining algorithms, patterns, or predictive models for IoT systems.Among the algorithms offered, we can also find SVM, which allows us to create SVM models oriented to the main problem of this project.
- As development tools, it has ThingSpace Develop, which offers applications to design, verify, and implement IoT solutions and machine to machine technologies, which accelerate the development of the entire process.
- At the service level, it has a unified device monitoring and management tool, scalable, reliable, and customizable by the user.
- It enables setting automatic responses according to the results of the analysis of the data, being able to create alerts and communicate through third party services.

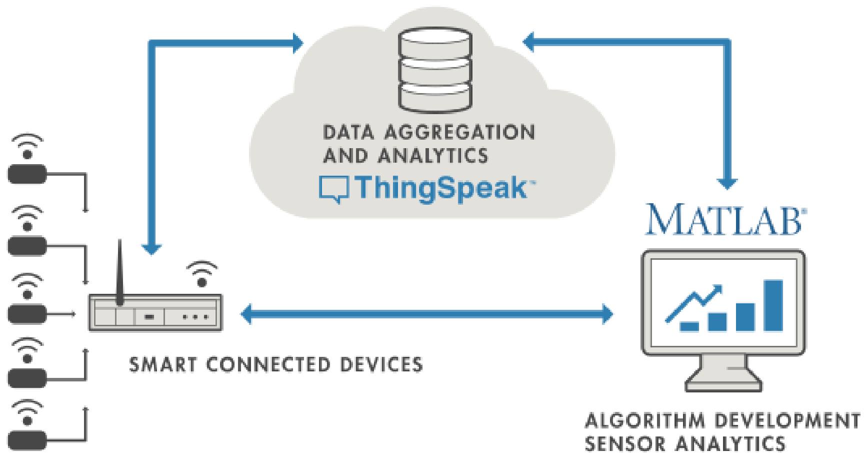


Figure 4: Basic structure of how ThingSpeak works [27]

## 3.3 Theoretical implementation of a interface for IoT services

For this theoretical implementation, we choose a four-layer architecture [28], which will have the following layers:

- The **perception layer** is where the KNX sensors act, capturing the information required for each type of application.
- The **network layer** is responsible for sending the data collected by the KNX sensors from the perception layer to the processing layer.
- The **processing layer** is where the data is stored and analyzed, proceeding from the network layer.
- The **application layer** is responsible for implementing the services of each type of application to users. As mentioned previously, between the kinds of applications in which IoT can be implemented, we can find smart homes, smart cities, etcetera.

### 3.3.1 Helium

Helium is a software that acts as middleware. A middleware solution is a software that acts as an interface between layers. Helium is a high-level solution that allows the integration between the perception layer and application layer in an easy way [29]. We found that there is a Python library that implements all the Helium functions. This library is called helium_client. Because of this, the language Python is chosen for this theoretical implementation.

### 3.3.2 Theoretical implementation

To implement the perception layer and its connection with the other layers, a Raspberry Pi 3 B+ has been chosen, and the KNX sensors would be connected to obtain real data. As operative system we will use Raspbian. In this Raspberry, the Helium library [29], and the python script would be developed to act as a gateway to the cloud services.

To make the hardware connection between the KNX bus devices and the Raspberry, we will use the BAOS 838 [30] hardware interface. This interface has been chosen for its cost since it is the cheapest and the best quality it offers. In figure 5 is shown how the interface works. However, other hardware interfaces could also be used, like the KNX interface IC00R01KNX, or others.

To be able to use this hardware interface and make the connection, we have several software options. However, we are interested in using the Bobaos library [31]. This library is developed in Javascript and implements KNX ObjectServer Protocol for the chosen hardware interface. As prerequisites to be able to use this library, we need to have NodeJS installed.

One of the advantages of using Python as a language to develop this implementation is that it supports the integration of Javascript code in Python scripts.

Once the sensors are connected by hardware and software to the raspberry, we can proceed to dump the data from the sensors to Helium. Through Helium, we can establish the connection between the dumped data with Bobaos library and the cloud services. Any of the services previously explained in section 1 can be used. Helium supports many cloud services, which is a great advantage. However, the most profitable and which could be more useful are Google Cloud IoT and AWS IoT.

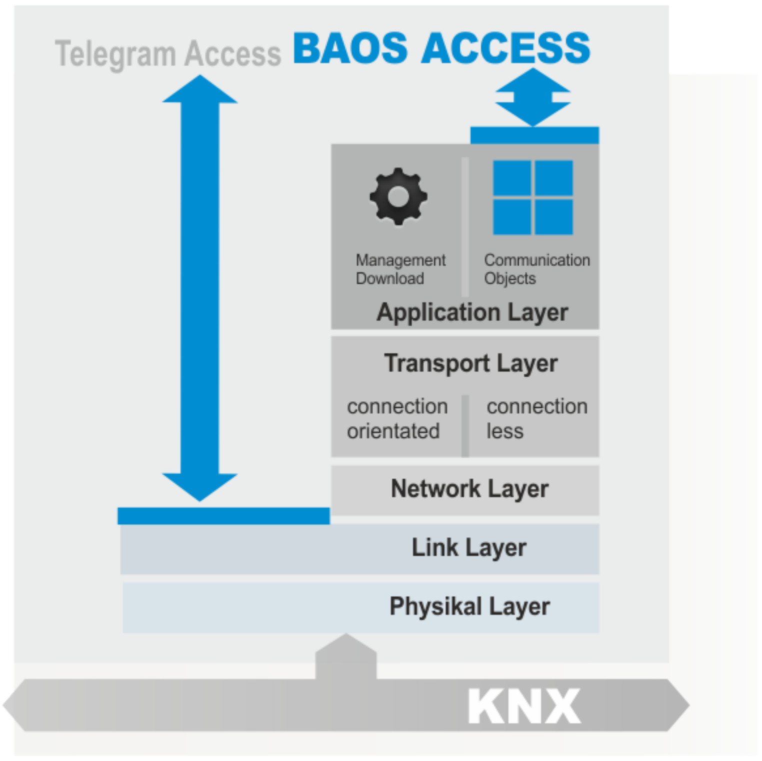


Figure 5: Telegram access for BAOS 838 [30]

Since we are interested in making the connection between our KNX sensors and cloud services secure, we can make use of the channels Helium offers to establish the connection. These channels support Azure IoT, AWS IoT, and Google Cloud IoT. If, for example, we chose Google Cloud IoT as cloud service, we could create the connection through the channels offered by Helium, or also make use of Google's Firebase services to access all the data also safely.

Finally, when the connection is established, we can use cloud services to create artificial intelligence models, analyze the data, even create a notification service for users.

# 4 Support Vector Machine (SVM)

In this section, the theoretical foundations of SVM are explained as all of the parts that form it. Approaches to handle regression and classification with SVM are introduced.

## 4.1 What is SVM?

SVM forms part of Supervised Learning, a part of statistical learning which starts from a series of examples to create a predictor system that tries to predict new values. SVM [32] are learning algorithms that analyze data on classification or regression problems and outliers detection, being one of the most potent methods supervised learning models. SVM methods try to find the optimal line, plane, or hyperplane, which linearly separates the data by maximizing the margin.

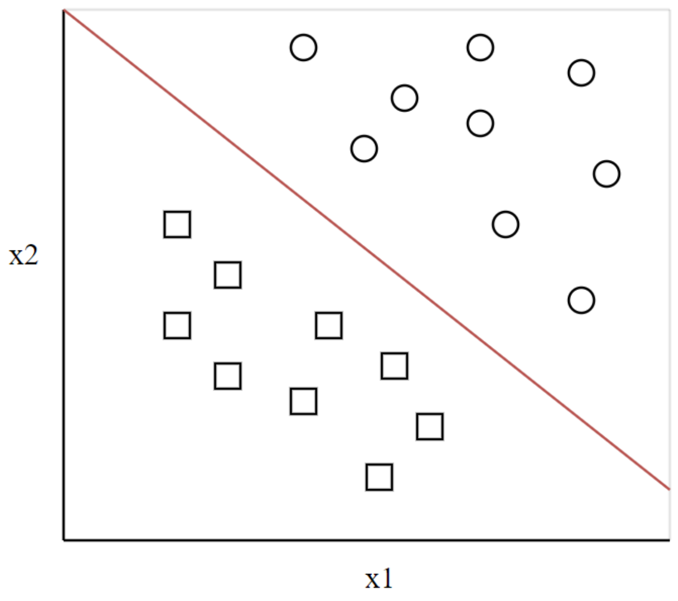


Figure 6: Linearly separable data

As it is possible to see in figure 6, there is a set of two-dimensional points that are linearly separable. SVM works excellent and fast with this kind of data. However, SVM also operates with non-linearly separable data.

## 4.2 Vector length

The length of a vector is also known as the norm of the vector. It is calculated by the euclidean formula, which tells how far are from the origin. Being $v = (v_1, v_2, v_3, ..., v_n)$ his norm is [33]:

$$\|v\| = \sqrt{v_1^2 + v_2^2 + v_3^2 + ... + v_n^2}$$

## 4.3 Direction of a vector

Vector direction ($w$) is calculated as the division of every element of the vector by the norm of itself. Being $v = (v_1, v_2, v_3, ..., v_n)$ his direction is [33]:

$$w = \left( \frac{v_1}{\|v\|}, \frac{v_2}{\|v\|}, \frac{v_3}{\|v\|}, ..., \frac{v_n}{\|v\|} \right)$$

## 4.4 Dot product

The dot product between vectors is a scalar quantity. It tells how two vectors are related.

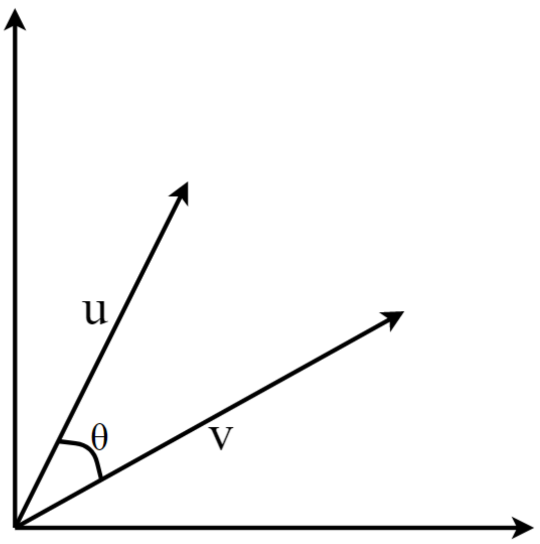


Figure 7: Graphical representation for dot product [33]

Mathematically is defined as [33]:

$$u \cdot v = \|u\| \, \|v\| \, cos(\theta)$$

For a two n-dimensional vector, the dot product is computed [33] as:

$$u \cdot v = \sum_{i=1}^{n} u_i v_i$$

## 4.5 Hyperplane

A hyperplane is a codimension-1 vector subspace from a vector space [33]. In a more informal way to define a hyperplane, we say that it's a line that divides the data into two different sets. In figure 6, we can see how a hyperplane splits the dataset between squares and circles. The equation for the hyperplane [33] is:

$$w \cdot x + b + \varepsilon = 0$$

Where **w** is a vector normal to the hyperplane (**w** is orthogonal to the hyperplane), and **b** is an offset. For any vector **x**, we can compute the next equation $w \cdot x + b + \varepsilon = y$. If $y = 0$, then x is on the hyperplane.

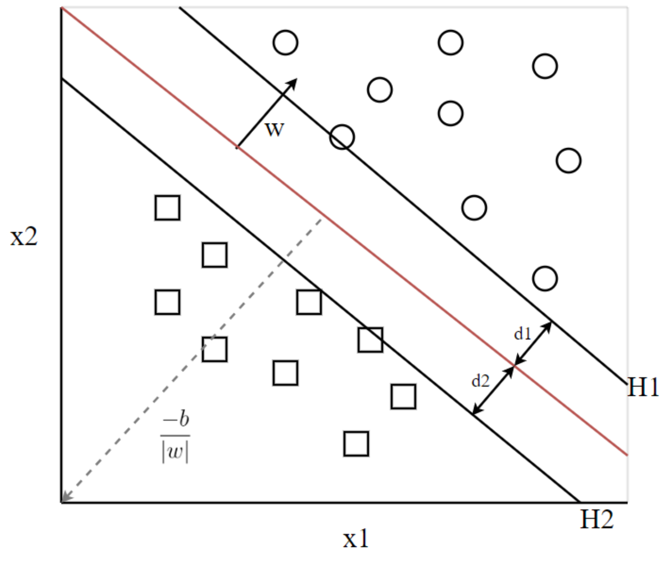


Figure 8: Data linearly separated by a hyperplane $w \cdot x + b + \varepsilon = 0$ [34]

H1 and H2 are the Support Vectors and they are mathematically defined as:

$$H1: \; w \cdot x + b = +1$$
$$H2: \; w \cdot x + b = -1$$

The sum of d1 and d2 is the margin, in there will never be data, this distance is known as epsilon. SVM attempts to find the maximum margin. For SVM, it is vital to choose the optimal hyperplane.

## 4.6 How SVM classifies and regresses?

To know how to classify with SVM, all **x** of the data above or on the hyperplane are classified as +1. If they are under the hyperplane, they are classified as -1. The formal definition of the constraints are [32]:

$$prediction = \begin{cases} y_i - wx_i - b \leq \varepsilon \\ wx_i + b - y_i \leq \varepsilon \end{cases}$$

## 4.7 Optimal Hyperplane

The optimal hyperplane [35] is the one with the most significant margin. In section 4.5, we explained H1 and H2, and they are the hyperplanes that go through the Support Vectors. The distance between H2 and the origin is calculated as $\frac{(-1-b)}{||w||}$ and the distance from H1 to the origin is $\frac{(1-b)}{||w||}$. The margin [32] ($M$) between H1 and H2 is given by:

$$M = \frac{(1-b)}{||w||} - \frac{(-1-b)}{||w||} = \frac{2}{||w||}$$

The margin then is defined by $\frac{1}{||w||}$ because $\frac{2}{||w||}$ is twice the margin. SVM attempts to find the optimal hyperplane, which is the one that maximizes the margin, so the bigger $\frac{1}{||w||}$ it is, the better. This problem can be changed to a minimization problem by trying to find the minimum inverse of the margin [32].

$$max\, \frac{1}{||w||} \;=\; min\, \frac{1}{\frac{1}{||w||}} \;=\; min\, ||w||$$

And this minimization problem is equivalent [32] to:

$$min\ ||w|| \equiv min\, \frac{||w||^2}{2}$$
$$such\, that\ y_i\,(w \cdot x + b) - 1 \geq 0,\ for\, i = 1...l$$

This equivalent problem is known as a convex quadratic optimization problem. This is the main optimization problem of SVM.

## 4.8 Hard and Soft Margin

SVM, working with linearly separable data, can be formulated in various ways, with Hard Margin or with Soft Margin. The difference between them is that Hard Margin SVM does not work with non-linearly separable data and does not accept outliers. Outliers (noise in data) is an obstacle for real-life problems since these, by definition, are not usually linearly separable, and we can find noise in them. For Hard Maring SVM, it is necessary to satisfy all the constraints in $y_i\,(w \cdot x_i + b) \geq 1$, and the outliers make this impossible, making the optimization problem unsolvable.

### 4.8.1 Soft Margin

Soft Margin is a more flexible model. It consists of adding a slack variable $\zeta_i$ to the constraints of the problem. This slack [32] variable is where the sample is located in relation to the hyperplane and the margin.

$$(w \cdot x_i + b) \geq 1 - \zeta_i,\ for\, y_i = +1$$

$$(w \cdot x_i + b) \leq -1 + \zeta_i,\ for\, y_i = -1$$

If we combine both of the equation above, we get the next equation [32]:

$$y_i\,(w \cdot x_i + b) - 1 + \zeta_i \geq 0,\ for\, y_i = +1, -1$$

Where $\zeta_i \geq 0$ for $i = 1...l$.
For a better regularization, $C$ parameter is added. This parameter controls the importance of the slack variable. For $C = 0$, the algorithm does not allow any misclassification. For $C > 0$ means that C samples can be misclassified. The more significant C is, the higher the margin will be. If $C = \infty$, then the model will misclassify most of the samples. In conclusion, Soft Margin allows margin violations. This margin violation means choosing a hyperplane, which will allow some data to stay between the margin area or to be misclassified.

## 4.9 Non linearly separable data

In this project, the problem is a real-life problem, which means more complex data. This data is not linearly separable, so we need to work with a version of SVM, which generalizes better and works with different shapes of the data. If the data is not linearly separable in the input space, then transformations to the data are applied. These transformations map the data from the input space into a higher dimensional feature space. The purpose is that after these transformations to a higher-dimensional feature space, the data is now linearly separable.

As is explained before in section 4.7, having linearly separable data, the aim is to obtain the optimal hyperplane which separates the data between the different classes. This hyperplane will be a hyperplane in the higher-dimensional space. These transformations of the data is named as “Kernel trick”.

### 4.9.1 Kernel Trick

The definition of the Kernel Trick [36] is a function that takes as input vectors that are in the original space and returns the dot product of these vectors in a higher-dimensional feature space. This function is named “kernel function” and has to satisfy Mercer’s constrains [37].
The formal for the Kernel Trick [38] definition is:

$$K\,(x_i, x_j) = (\phi\,(x_i)\,, \phi\,(x_j)) \tag{1}$$

It is possible to find multiple kernel functions [39] that transform the input data into a higher-dimensional feature space. Some of the kernels are:

- Polynomial [38]
$$K\,(x_i, x_j) = (x_i \cdot x_j + a)^p \tag{2}$$

- RBF [38]
$$K\,(x_i, x_j) = e^{-\gamma\left(x_i - x_j\right)^2} \tag{3}$$

- Sigmoid [38]

$$K(x_i, x_j) = tanh(\eta x_i \cdot x_j + \upsilon) \tag{4}$$

### 4.9.2 Parameters

Working with SVM for regression (SVR), there are two parameters to work with, which are C and Epsilon. However, it depends on the selected kernel. In this project, the chosen kernel is RBF because it's a well-studied method, and it works well in practice. This kernel is explained in the previous section, which adds another parameter to estimate. This new parameter is Gamma.

**4.9.2.1 C** This parameter is the error penalty. It punishes the errors made when the model misclassifies instances. If it is a small value means that the punishment done to the mistake made is small. If it's a high value, it will punish with more significant penalties to the errors. When C is large, it can lead to worse generalization models. Ergo, it's crucial to choose a good C value. [40]

**4.9.2.2 Epsilon** For SVR, this parameter will tell how smooth and how good will the model generalize. This is because it affects the number of support vectors selected. [41]

**4.9.2.3 Gamma** This parameter defines how far the influence of a single training example reaches. Low values mean that the area of influence is significant, and larger values indicate that the ara of influence will be small. [40]

# 5 Experimentation

This section covers the details of the implementation and obstacles faced in the process of transforming the theoretical concepts into operative python code.

## 5.1 Hardware

For the development of this project, we used as main computer a DELL XPS 13 9365 for computing tasks, research, and model evaluations. This computer is an ultrabook made for long battery duration, being the CPU an i5-7200U. The problem we faced here is that for tasks of cross-validation, parameter estimation, and model fit, it would take long periods of time. This time could be reduced by using dedicated services like cloud computing, GPU processing, or a computer with better specs.
Still, in order to speed up the project, the tool Google Colaboratory has been used. This tool offers cloud computing an environment similar to Jupyter notebook. The specs that the free plan have are the next ones:

- Tesla K80 GPU
- 1 core 2 threads @ 2.2GHz
- 13GB RAM
- 33GB Free Space
- Max 12 hours of code execution

As specified before, Google Colaboratory offers a free GPU. Still, the library used for the model creation and training is scikit-learn, which does not intend to be used as a deep learning framework.

## 5.2 Python

We selected Python as the language for the development and implementation of this project for the following reasons:

- **Readable and easily maintainable code**: Python allows us to develop clear and concise code, without the need for complications. It is simple and easy to use.
- **Compatible with most systems and platforms**: Python is an interpreted programming language that allows us to execute the same code on multiple platforms without recompiling. Therefore, it is not necessary to recompile the code after making any changes.
- **Libraries**: Standard Python libraries allows us to choose from a wide range of modules according to our needs. Each module also allows us to add functionalities to the project without writing additional code. One of the benefits of Python libraries for data analysis is that it offers access to a wide variety of libraries related to data analysis and data science.

- **Many Frameworks and open source tools**: As an open-source programming language, Python helps us eliminate development costs from this project. It is possible to use several open-source Frameworks, libraries, and tools to reduce development time without increasing the costs.

- **Scalability**: It helps with the scalability of projects providing great flexibility and different ways of addressing various problems. We can find Python in multiple industries, allowing faster application development.

### 5.2.1 Libraries

The Python libraries selected for the realization of this project are the following.

**5.2.1.1 Pandas** Pandas is a Python library that allows us to work with different data formats, thus helping to have better data management, whether to perform analysis of them, preprocessing, or other tasks in which it is required to work with large amounts of data. It is an open-source library, and it is kept updated by its great use among the programmer community in Python.

**5.2.1.2 Numpy** This open-source Python library provides the project facilities to perform mathematical calculations. For example, a great facility to work with vectors, since it incorporates a vector object with which we can operate as if it were a scalar so that this tool will be necessary for the development of SVM.

**5.2.1.3 Matplotlib** We are going to use this open-source library for graphics. These graphics will help the visualization of the results, apart from being very useful when comparing and being more visual, it will facilitate the understanding of the results. This module is similar to the functions offered by MATLAB, being very configurable and easy to use.

**5.2.1.4 Scikit-learn** Scikit-learn is an open-source Python library that facilitates and provides us a large number of supervised and unsupervised learning algorithms. With the use of this library, we would already have incorporated all the previously explained libraries (Pandas, NumPy, Matplotlib) and many others (IPython, SciPy, Sympy) that will not necessarily be used in this project but may be helpful. Some of the models offered by this library are:

- **Clustering**: Methods such as KMeans, among others, to perform clustering tasks.

- **Datasets**: Generation of datasets with certain properties to test algorithms.

- **Feature Extraction**: Define attributes in different types of data such as text or image.

- **Feature Selection**: Allows the identification of attributes to create new models.

- **Cross-validation**: View the performance of the different algorithms developed with data that we have not previously worked.

## 5.3 Data preprocessing

Data preprocessing is the step where we transform and shape the data, so when we are working with, it's easier to understand, and the algorithm can interpret the data better. The steps for the realization of the data preprocessing are:

- Outlier detection
- Data filtering (Noise reduction)
- Data normalization

### 5.3.1 Data

The data we are working with are five files with the measures from three different sensors: CO2 (ppm), relative Humidity(%), and temperature(°C) from the day 21 December 2018 to the day 31 December 2018. The measures made were temperature indoor, relative humidity indoor, and CO2 concentration indoor. Those features were recorded in five different rooms of SHC for a period of one week utilizing five stations using Loxone temperature/ humidity/ CO2 sensors. The places included: "living room, children's room, bedroom, kitchen, and bathroom." The sensor specifications are:

- Indoor CO2 measurement (Within 0 and 2000 ppm, accuracy ±50 ppm)
- Indoor temperature ($T_i$) measurement (Within 0 and 50°C, accuracy at ±0.3°C)
- Indoor relative humidity (rH) (within 0 and 100% RH, accuracy ±3% RH)

### 5.3.2 Data normalization

Data normalization is not always necessary. If all the data is in the same range, we don't need to scale it. However, as we have different types and varieties of data (percentage, degree, and ppm), we need to normalize them. Normalize the data means to scale it between 0 and 1. And we have different methods to achieve this. The technique that we are going to use is known as min-max normalization. The formula is for data normalization is [42]:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5}$$

### 5.3.3 Outlier detection and removal

An outlier can be defined as a data point that is far away from all the other data points. In this problem, we understand outliers as mistakes done in the measurements made by the sensors. To delete these outliers, we decide to use Z-score, which tells how far is a data point from the mean. The formula to calculate the Z-score is [35]:

$$z = \frac{x - \mu}{\sigma} \tag{6}$$

Where $\mu$ represents the mean, and $\sigma$ the standard deviation. After we calculated the Z-score, we need to establish a threshold to delete all of those instances that their Z-score is higher or lower than the threshold are considered outliers. We set a threshold of 3, meaning that every point that is more than three times the standard deviation will be considered as an outlier.

### 5.3.4 Data filtering (Noise reduction)

Savitzky–Golay filter is a noise reduction and smoothing filter that can be applied to the data to the data. By using this filter, it is possible to increase the accuracy of the predictions without distorting the signal. tendency. Applying this filter reduces the noise and deletes most of the outliers from the data signal, allowing important patterns to stand out and the more reliable prediction of the model. As it is shown in figure 9, all the noise from the data is erased. [43]

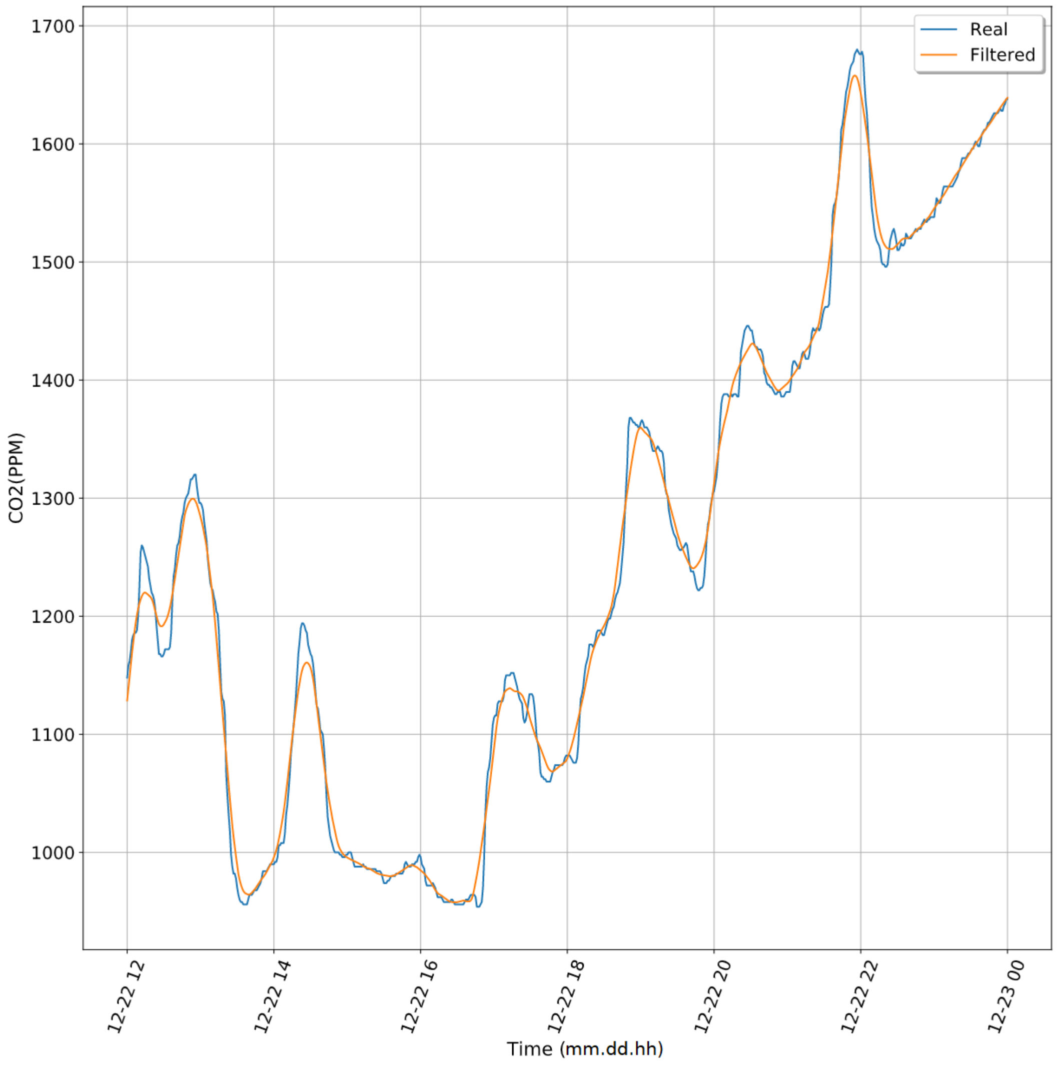


Figure 9: Living room CO2 signal(4 hours) compared with living room CO2 signal with applied Savitzky–Golay filter, with a window length of 21 and polynomial order of 1

### 5.3.5 Metrics for model evaluation

Because we want to be sure that our model is efficient and accurate, we selected three different metrics that are the next ones.

**5.3.5.1 Mean Squared Error** Mean Squared Error (MSE) is one of the most used and known metrics for regression problems. It measures the average of the squared difference between the predicted values and the actual values. Because it is the square of the difference, all the errors, even the small ones, have significant penalties. In conclusion, the lower the score obtained with this metric, the better. The formula to calculate MSE is [44]:

$$MSE = \frac{1}{n}\sum (y_i - \check{y}_i)^2 \tag{7}$$

**5.3.5.2 Root Mean Squared Error** RMSE is a quadratic score that measures the average of the error. It's the square root of the MSE, which has been explained in the previous section 5.3.5.1. This metric is handy in terms of knowing if there are outliers that might be messing our model.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum (y_i - \check{y}_i)^2} \tag{8}$$

**5.3.5.3 Mean Absolute Error** Mean Absolute Error (MAE) is powerfully robust with outliers and missing data because it does not penalize as hard as MSE. For our problem, it's useful because we are not working with data that has a big group of outliers and missing data. The formula to calculate MAE is [44]:

$$MAE = \frac{1}{n}\sum |y_i - \check{y}_i| \tag{9}$$

**5.3.5.4 R Squared** R squared (R2), also known as the coefficient of determination, evaluates the performance of our model by comparing our current model with a constant baseline and specifies how much better is our model. It always will be less than 1. The range of the score is from $-\infty$ to 1, the closest to 1, the better it will be. The formula to calculate R2 is [44]:

$$R2 = 1 - \frac{\sum (y_i - \check{y}_i)^2}{\sum (y_i - \bar{y}_i)^2} \tag{10}$$

## 5.4 General ensemble vs. One model per room

Ensembling is a technique that combines multiple weak models to get a robust and optimal predictor model. Ensemble methods help to minimize factors like noise, improving the stability and performance of the model. There are many different methods to create ensemble models. For our project, two types of ensembles were used: Bagging and Voting regressor. Both are offered in the library of Scikit-learn. Despite that, we also worked on generating a different model per room, having better accuracy than the ensembles tried.

### 5.4.1 Bootstrap Aggregation ensemble (Bagging)

As the name explains, Bagging is composed of two different parts: Bootstrapping and Aggregation, to form one ensemble model.

- **Bootstrapping**: This is a sampling method. Consists of the creation of different random subsamples using replacement. After the selection of these subsamples of the dataset, the learning algorithm that is SVM will proceed to predict.
- **Aggregation**: After the selection of the subsamples, and the multiple model creations, the aggregation method is used to aggregate all the models and form a more efficient predictor.

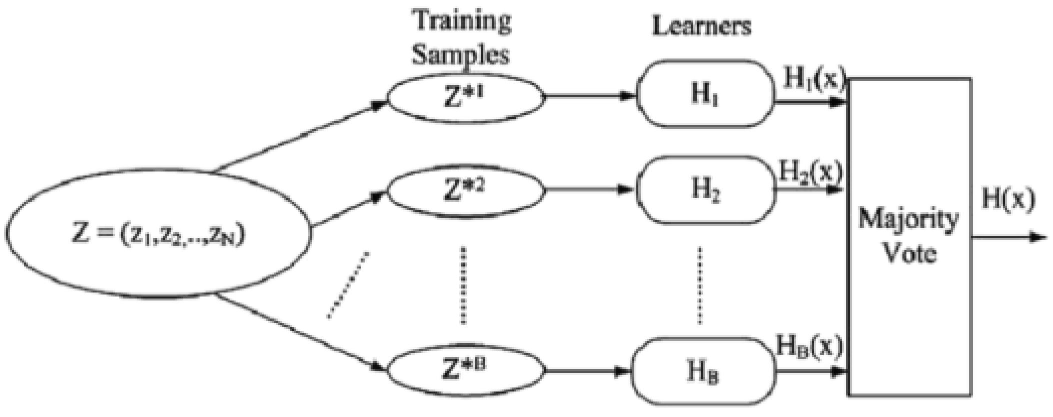


Figure 10: Bagging approach to regression. Weak learners are trained with subsamples of the data. Then the final regression prediction is made with some averaging method. [45]

Bagging can offer multiple advantages as a combination of multiple weak models to produce a strong predictor, helping to reduce the variance, noise, and overfitting. But it also has disadvantages. It has a really high computational cost, and it can be easily underfitted. This ensemble approach is applied to our project by preprocessing the data in a different way.

As explained in section 5.3.1, the data provided consists of 5 files, one for each room (Bedroom, Living room, Kitchen, Bathroom, Kids room). These files are merged into one file in order to train the ensemble bagging model. Each subsample selected can contain measures from different files.

### 5.4.2 Voting ensemble

The voting ensemble is an easy way to create an ensemble model. The idea of voting doesn't differ much from the bagging idea. Multiple heterogeneous weak models are created using subsamples from the dataset. Then the final model makes predictions based on the average of the predictions made by these weak models. The formula to calculate the averga value is [35]:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{11}$$

This approach differs from Bagging in our project because of the preprocessing of the data, in each ensemble model, the ways to prepare the data to train the weak learners are different. In this ensemble approach, the data is not merged into one file, but a different model is trained for each file. These individual models are merged using the average method to predict the regression.

### 5.4.3 One model per room

Although approaching this problem through the model ensemble is a good option, besides the creation of different models per room, it is also an excellent choice to get higher accuracy in the predictions. The data from the rooms differ much from one to another. For example, the values of temperature in the kitchen vary too much from the living room. This situation happens due to the activities of cooking made in the kitchen. The same problem occurs with the measured relative humidity in the bathroom and the other rooms.

Furthermore, the creation of a python script to create a predictor model is faster, more efficient, and the models made will have higher accuracy because the data to process and fit is significantly lower. This fact allows that more simple processors could process and predict all of this data. For example, products like Raspberry Pi or Arduino can be used for this task.

## 5.5 Code

This section finally shows the Support Vector Machine code. This is one of the main objectives of the thesis.

### 5.5.1 Code flow

The direction and how the code is executed is shown in figure 11. This code flow is applied for all the approaches to our project. These approaches are explained in section 5.4.

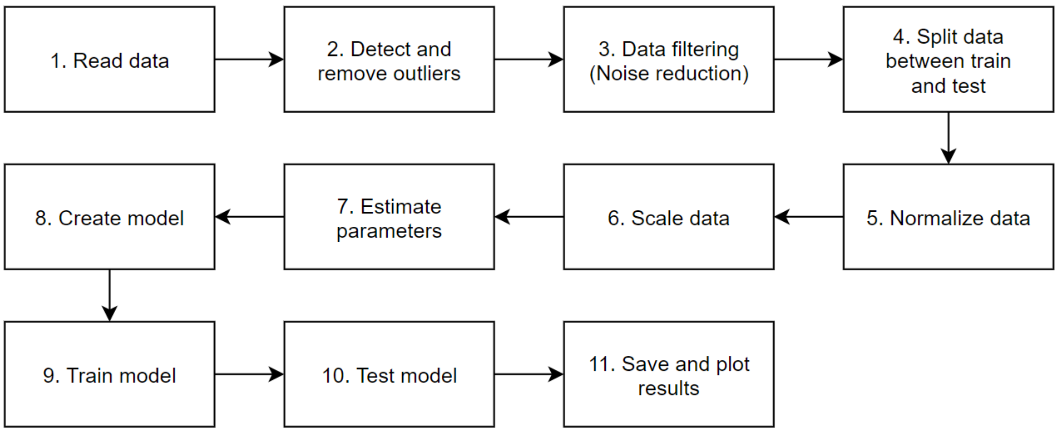

1. Read data
2. Detect and remove outliers
3. Data filtering (Noise reduction)
4. Split data between train and test
5. Normalize data
6. Scale data
7. Estimate parameters
8. Create model
9. Train model
10. Test model
11. Save and plot results

Figure 11: Flowchart of the steps followed in code

### 5.5.2 Code steps

1. **Read data**: Open the csv files and transform the data that they contain.
2. **Detect and remove outliers**.
3. **Data filtering (Noise reduction)**: Apply Savitzky–Golay filter.
4. **Split between train and test**.
5. **Normalize data**: Scale all the data into a range from 0 to 1.
6. **Scale data**: Apply a standard scalation to every attribute has the same importance in the algorithm.
7. **Estimate parameters**: Calculate the optimal parameters for the model of each file.
8. **Create model**: With the estimated parameters create the optimal model.
9. **Train model**.
10. **Test model**.
11. **Save and plot results**.

### 5.5.3 Read data

As the first step in our code, we have to read files so we can preprocess them later. For that purpose, we use the library Pandas. This library has the function read_csv, which opens a CSV file and loads it as a dataframe. Working with dataframes it is easy and efficient. While the file is loaded into the dataframe, every data is transformed to its type: Float to float, int to int, string to string, etcetera. However, for the attribute date, we have to apply a different process. We use the function contained in the package pandas called to_datetime. The code for this part has the same process for every script, though it varies in some details. On line 23, 4 and 9 from files ensemble_voting.py, ensemble_bagging.py and svr_splitted_rooms.py is shown the code for this task.

### 5.5.4 Detect and delete outliers

The outliers are removed from the data with the function zscore that Scipy has in their package stats. This function calculates the Z-score for each instance, allowing the posterior removal of those instances that their Z-score is higher than three, considered outliers. In figure 12, is shown the visualization of the scattered data to find possible outliers, that are marked with a black rectangle.

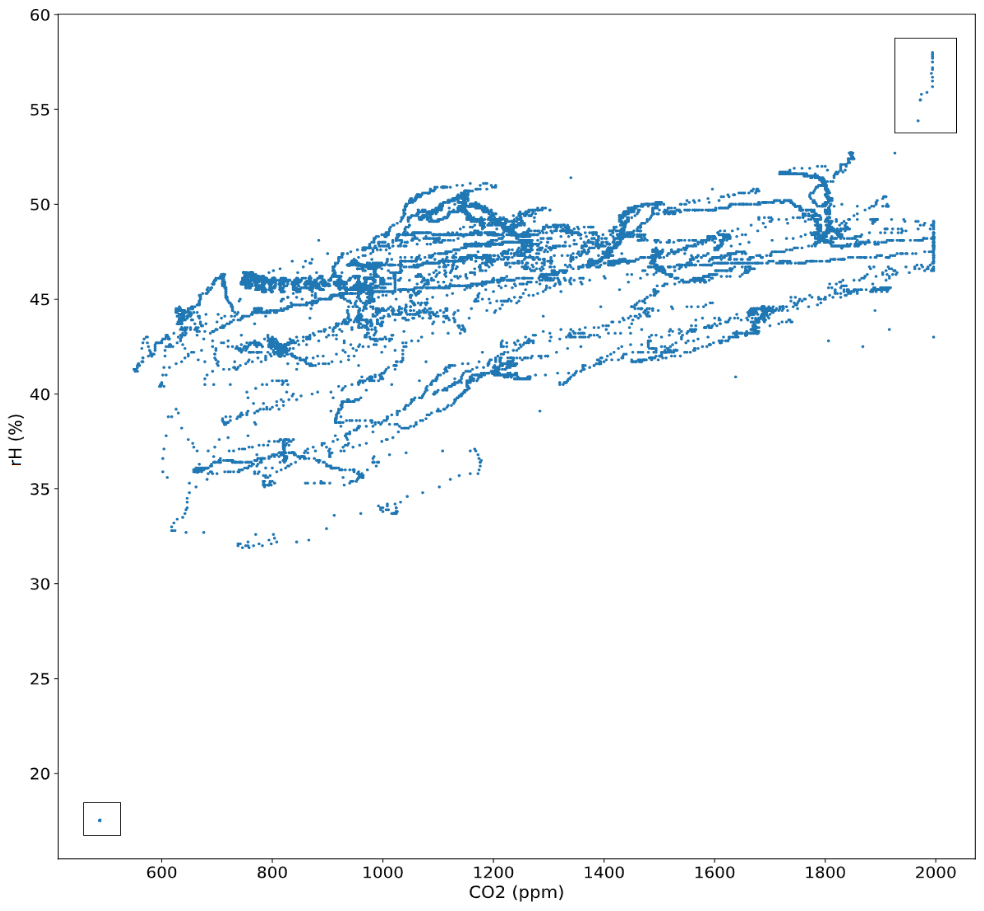


Figure 12: Outlier visualization from Living Room file: CO2 (ppm) vs rH(%)

After the detected outliers are removed, the scattered the data looks like shown in figure 13. As it is possible to see all the outliers have been removed.

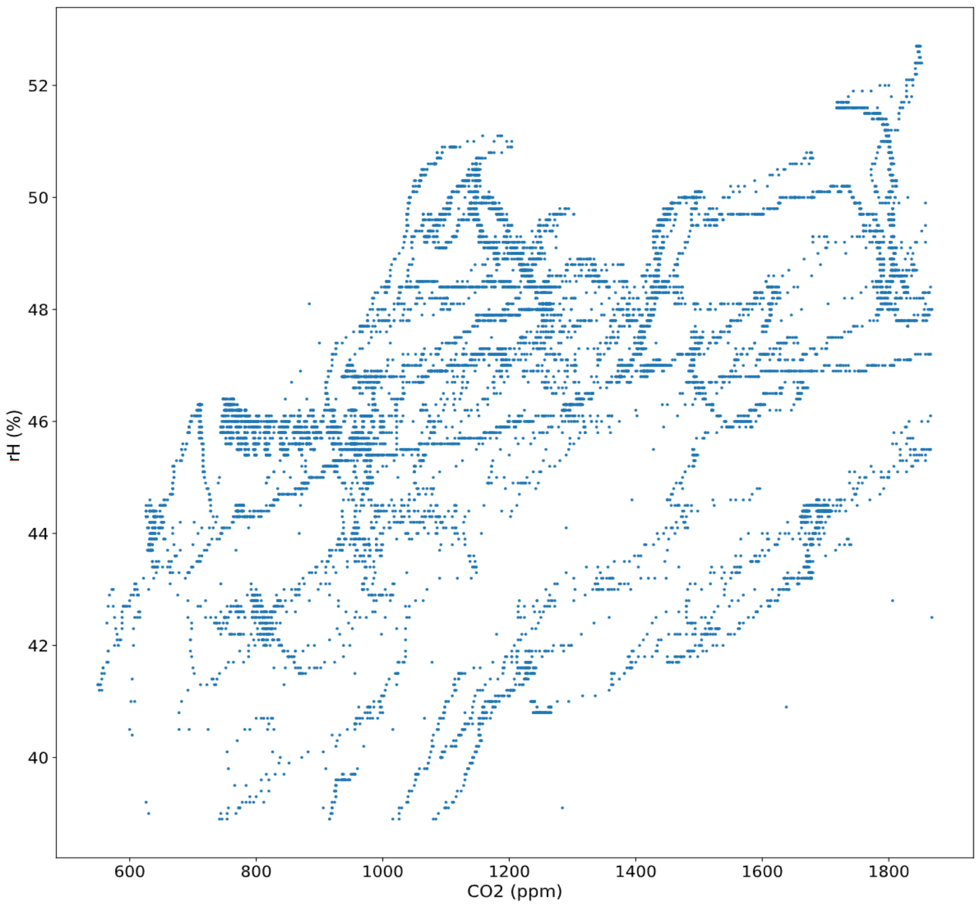


Figure 13: Scattered data after the removal of the outliers from Living Room file: CO2 (ppm) vs rH(%)

### 5.5.5 Data filtering (Noise reduction)

Scipy has the package Signal, which has the function savgol_filter. This function applies a Savitzky–Golay filter to the data, with the selected window length and polynomial order. We used a window length of 21 and a polynomial order of 1 to reduce all the possible noise that the data signal could have but without distorting the original signal. Additionally, this helps to remove possible outliers that the zscore function could not. The result of the applying of this filter is shown in figures 9 and 14.

### 5.5.6 Normalize data

As it is specified in section 5.3.2, the applied formula for normalization is applied to the whole dataframe in only one line. As said in section 5.5.3, working with dataframes is useful and practical. On line 57, 41 and 40 from files ensemble_voting.py, ensemble_bagging.py and svr_splitted_rooms.py is shown the code for this task.

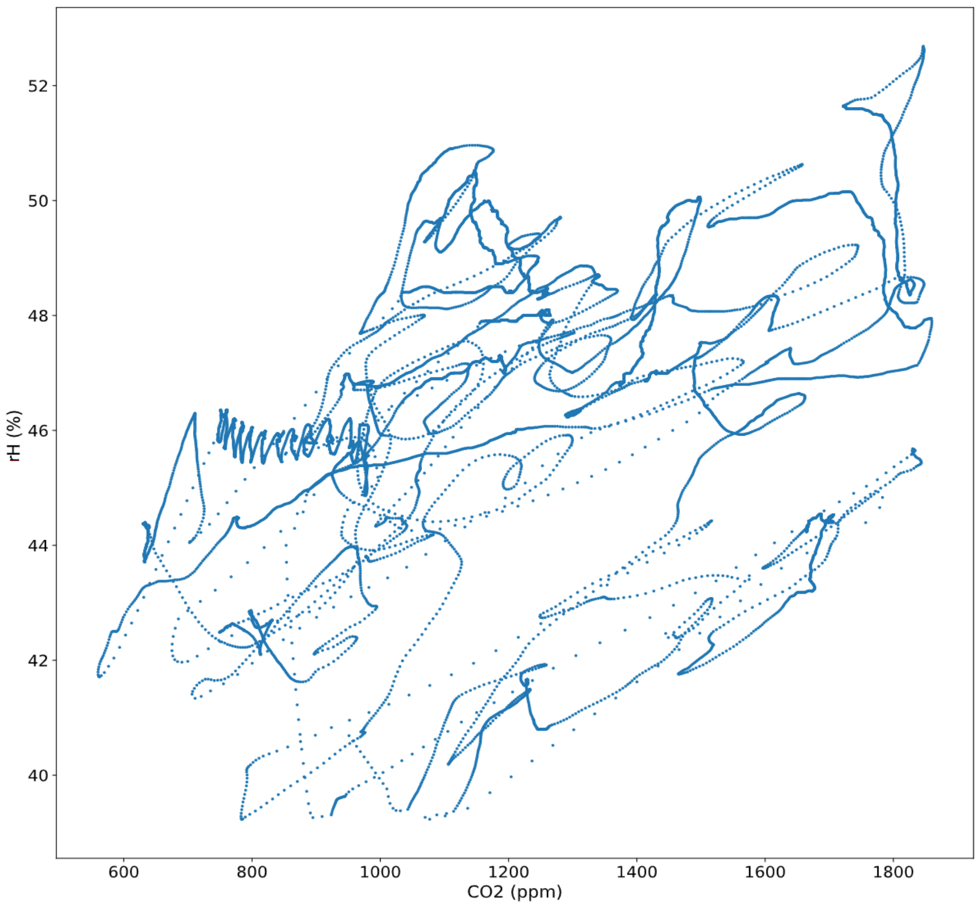


Figure 14: Scattered data after the filter Savitzky–Golay is applied to the data from Living Room file: CO2 (ppm) vs rH(%)

### 5.5.7 Scale data

Since we want all the data we are working with, to have the same range, we scale the data. The object StandardScaler that Scikit-learn offers scales the data to have the same weight. On lines 108, 48 and 46 from files ensemble_voting.py, ensemble_bagging.py and svr_splitted_rooms.py is shown the code for this task.

### 5.5.8 Split between train and test

Working with the same data to train and test is not right because we might overfit the model. We need new data to test it, so we split it into two sets: Train set and test set.

We selected two different test and train sizes of the dataset to see how the model works in varied circumstances. We set one day (1440 rows) to train and test in the first instance, and five days (7200 rows) to train and five days to test. Because we are working in two different ways to create the ensemble models, two ways to test the ensemble models are made:

1. Two tests for each file: For each file, we will have two test sets, one with a duration of one day and another with a duration of five days.

2. Two global tests: We create two tests from the merged file with random samples with the duration of one day and five days.

To split the data, we use the function train_test_split, which will allow us to split between train and test easily. Just with setting the number of instances we want for train and test it will return two objects with train and test subsets with the specified size. On lines 37, 21 and 25 from files ensemble_voting.py, ensemble_bagging.py and svr_splitted_rooms.py is shown the code for this task.

### 5.5.9 Parameter estimation and model training

Because these parameters are problem dependent, so exploration to find the optimal ones, must be done. With the function, RandomizedSearchCV, which uses cross-validation, allows the user to enter a parameter grid, and it will randomly select subsets of parameters to test. It will return the best model with the optimal parameters. In the article "A Practical Guide to Support Vector Classification" [46], it says that "We found that trying exponentially growing sequences of C and $\gamma$ is a practical method to identify good parameters.", this is why we use the logspace function that the package NumPy has. On lines 84 and 57 from files ensemble_voting.py and svr_splitted_rooms.py is shown the code for this task.

Because of the different ways to process the data, and how the functions from Scikit-learn works, we do two different ways to estimate the parameters. For the bagging ensemble approach, the parameters to estimate are not from SVR. Instead, we have to estimate the parameters: max_samples (%), max_features (%), and if bootstrap is applied or not. On line 62, the code for this task in file ensemble_bagging.py is shown.

Parameter estimation to achieve good results is a real big problem, and this is because it is a problem-dependent. For each problem, there are different optimal parameters. This task after data preprocessing is one of the most significant in the process of data analysis and future prediction.

**5.5.9.1 C** For the estimation of the parameter C, an exponential range from 1 to 512 has been selected. However, the C selection is critical because it will influence how good the model will predict. We tried to cover an extensive range for this penalty factor in getting the best possible value.

**5.5.9.2 Epsilon** The values of Epsilon to get good prediction goes on a range from 0 to 1. We decided to use the values: 0.01, 0.05, 0.1, 0.125, 0.2, 0.5, 0.7 to cover most of the range. The best value obtained in most of the trained models is 0.125.

**5.5.9.3 Gamma** For the estimation of the parameter gamma, an exponential range from 1 to 512 has been selected. As we used for the evaluation of the parameter C. The fact that we had a vast range to estimate C and gamma made the process of parameter estimation slow, taking up a long time to determine it.

### 5.5.10 Predict

With the model created and the parameter estimation finished, we have to predict the CO2 level from test sets. For this job, the object SVR from Scikit-learn has the function predict, which with the test set, will predict all the CO2 values. On lines 121, 65 and 61 from files ensemble_voting.py, ensemble_bagging.py and svr_splitted_rooms.py is shown the code for this task.

# 6 Discussion

In this section, the experimentation and discussion of the results obtained from the multiple python scripts coded for this project are presented. Some troubles faced, and some improvements discovered are also shown.

## 6.1 Applying shuffle

Once we visualized the data, we noticed that how the data is distributed is not regular and sometimes is day dependent. Because of this, training with specific days does not have sense, and it will lead to bad scores because of the overfitting, so to avoid this situation, we shuffle the data. Obtained results with and without shuffle are shown in table 1 and table 3.

| r2 | mae | mse | rmse | train size | test size | file |
|---|---|---|---|---|---|---|
| -0.8687 | 0.2974 | 0.1314 | 0.3625 | 1440 | 1440 | bathroom |
| -1.0935 | 0.2493 | 0.0926 | 0.3043 | 7200 | 7200 | bathroom |
| -1.1107 | 0.348 | 0.1785 | 0.4225 | 1440 | 1440 | bedroom |
| -0.26 | 0.2001 | 0.0542 | 0.2328 | 7200 | 7200 | bedroom |
| -1.6436 | 0.3328 | 0.1434 | 0.3787 | 1440 | 1440 | kidsroom |
| -0.1456 | 0.1269 | 0.0483 | 0.2197 | 7200 | 7200 | kidsroom |
| -0.9397 | 0.2517 | 0.0921 | 0.3035 | 1440 | 1440 | kitchen |
| -0.392 | 0.2174 | 0.0621 | 0.2493 | 7200 | 7200 | kitchen |
| -1.1625 | 0.3545 | 0.1549 | 0.3936 | 1440 | 1440 | livingroom |
| -0.3731 | 0.1323 | 0.0326 | 0.1806 | 7200 | 7200 | livingroom |

Table 1: SVR results with one model per room and non shuffled data. Used two test and train sets per file (1440 instances that equal one day, and 7200 instances that equals five days)

When we shuffle the data, SVR has an improvement in the predictive performance and a higher quality of the model. This is because when we shuffle the data, we avoid some patterns while we split the data, having various data in the train and test set. As it is possible to observe, the improvement of the R2 score is enormous, almost 200%.

Some of the results obtained with the R2 metric are negative and even less than -1. This is due to the implementation made by the Scikit-learn library. Depending on the implementation, the possible range to obtain results is from 1 to $-\infty$. Where in that range, the closer the results are to 1, the better the model, meaning that predicted values are pretty similar to the true values. If the values are close to 0, the model won't be able to explain anything of the test set.

## 6.2 Applying Savitzky–Golay filter

Applying Savitzky–Golay filter was a great improvement in our problem to predict CO2 levels. This is because of all the noise that was on the dataset. With this filtering, all the noise is erased, smoothed, and we get the signal without distorting it. This filtered signal is shown in figure 9, and the scatter of the filtered data is shown in figure 14.

| r2 | mae | mse | rmse | train size | test size | file |
|---|---|---|---|---|---|---|
| 0.5619 | 0.1169 | 0.0295 | 0.1716 | 1440 | 1440 | bathroom |
| 0.7119 | 0.0726 | 0.0189 | 0.1376 | 7200 | 7200 | bathroom |
| 0.7381 | 0.0739 | 0.0164 | 0.1282 | 1440 | 1440 | bedroom |
| 0.8119 | 0.0529 | 0.0113 | 0.1065 | 7200 | 7200 | bedroom |
| 0.8284 | 0.046 | 0.0087 | 0.0932 | 1440 | 1440 | kidsroom |
| 0.8747 | 0.0309 | 0.0065 | 0.0807 | 7200 | 7200 | kidsroom |
| 0.5417 | 0.1211 | 0.0262 | 0.1618 | 1440 | 1440 | kitchen |
| 0.6095 | 0.0968 | 0.0221 | 0.1486 | 7200 | 7200 | kitchen |
| 0.7593 | 0.0614 | 0.0122 | 0.1104 | 1440 | 1440 | livingroom |
| 0.8298 | 0.0419 | 0.0089 | 0.0942 | 7200 | 7200 | livingroom |

Table 2: SVR results trained and tested with one day, and five days of the data (1440 instances, and 7200 instances). The data was not filtered with the Savitzky–Golay filter.

There is a large difference from the scores obtained without shuffling and filtering the data. The obtained results after both tasks finished are shown in table 3.

| r2 | mae | mse | rmse | train size | test size | file |
|---|---|---|---|---|---|---|
| 0.8386 | 0.0449 | 0.0102 | 0.1009 | 1440 | 1440 | bathroom |
| 0.8319 | 0.0489 | 0.0108 | 0.1039 | 7200 | 7200 | bathroom |
| 0.8691 | 0.0497 | 0.0089 | 0.0943 | 1440 | 1440 | bedroom |
| 0.7251 | 0.0852 | 0.0186 | 0.1362 | 7200 | 7200 | bedroom |
| 0.9412 | 0.0153 | 0.004 | 0.0632 | 1440 | 1440 | kidsroom |
| 0.917 | 0.0256 | 0.0058 | 0.076 | 7200 | 7200 | kidsroom |
| 0.7344 | 0.0642 | 0.0161 | 0.127 | 1440 | 1440 | kitchen |
| 0.7322 | 0.0682 | 0.016 | 0.1263 | 7200 | 7200 | kitchen |
| 0.7949 | 0.0571 | 0.0122 | 0.1103 | 1440 | 1440 | livingroom |
| 0.847 | 0.0422 | 0.0089 | 0.0943 | 7200 | 7200 | livingroom |

Table 3: SVR results with one model per room, shuffled data and filtered with Savitzky–Golay filter. Used two test and train sets per file (1440 instances that equal one day, and 7200 instances that equals five days).

## 6.3 Implementations

We compare three implementations:

- Bagging ensemble: Using the bagging ensemble to train and test, explained in section 5.4.1.
- Voting ensemble: Using the voting ensemble to train and test, explained in section 5.4.2.
- One model per room: Using different models, one per room, to train and test, explained in section 5.4.3.

### 6.3.1 Bagging ensemble vs. Voting ensemble

Since we are working in two different ways to train and test the models (Merging the data of each file and working with it separately), the tables shown are different.

Operating with the voting ensemble made things easier because we can select which room we want to test. This helps set the algorithm in the situation of predicting CO2 values. But, having all the data merged is messy, and we do not know from which room the data selected to train and test comes. That is one of the main reasons why the results from the bagging ensemble are lower than the results from the voting ensemble.

| r2 | mae | mse | rmse | train size | test size | file |
|---|---|---|---|---|---|---|
| 0.6163 | 0.1094 | 0.0285 | 0.1689 | 1440 | 1440 | bathroom |
| 0.5874 | 0.1172 | 0.0313 | 0.1768 | 7200 | 7200 | bathroom |
| 0.7656 | 0.0796 | 0.017 | 0.1304 | 1440 | 1440 | bedroom |
| 0.7552 | 0.0843 | 0.0176 | 0.1328 | 7200 | 7200 | bedroom |
| 0.7713 | 0.0549 | 0.0142 | 0.1193 | 1440 | 1440 | kidsroom |
| 0.7651 | 0.0607 | 0.015 | 0.1226 | 7200 | 7200 | kidsroom |
| 0.4786 | 0.1233 | 0.033 | 0.1816 | 1440 | 1440 | kitchen |
| 0.4859 | 0.1265 | 0.0319 | 0.1787 | 7200 | 7200 | kitchen |
| 0.7415 | 0.0681 | 0.0153 | 0.1239 | 1440 | 1440 | livingroom |
| 0.7256 | 0.0732 | 0.016 | 0.1263 | 7200 | 7200 | livingroom |

Table 4: Voting ensemble results. The results from this ensemble have an average R2 score of 0.66

The results obtained from the voting ensemble are shown in table 4. Although the results from this ensemble are low, this happens because there are two rooms in which their data has more unpredictable patterns. These rooms are the kitchen and the bathroom.

| r2 | mae | mse | rmse | train size | test size | file |
|---|---|---|---|---|---|---|
| 0.3652 | 0.1519 | 0.0363 | 0.1906 | 1440 | 1440 | merged data |
| 0.3786 | 0.1501 | 0.0354 | 0.1882 | 7200 | 7200 | merged data |

Table 5: Bagging ensemble results. The results from this ensemble have an average R2 score of 0.36

Values predicted for both algorithms are shown in figures 15 and 20. As it is possible to see in both figures, the metrics used to evaluate the models are represented in the predicted values. Since the better the results of the metrics, the closer the predicted values are to the true values.

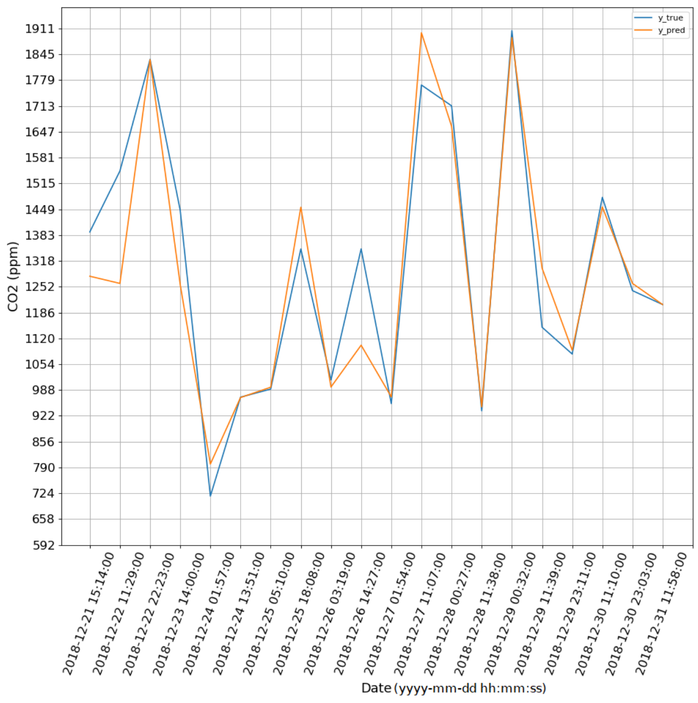


Figure 15: Predictions made by the best voting ensemble model. The data is from the bedroom. Train and test size: 1440 instances (one day). R2 score: 0.76, MAE: 0.07, MSE: 0.01, RMSE: 0.13

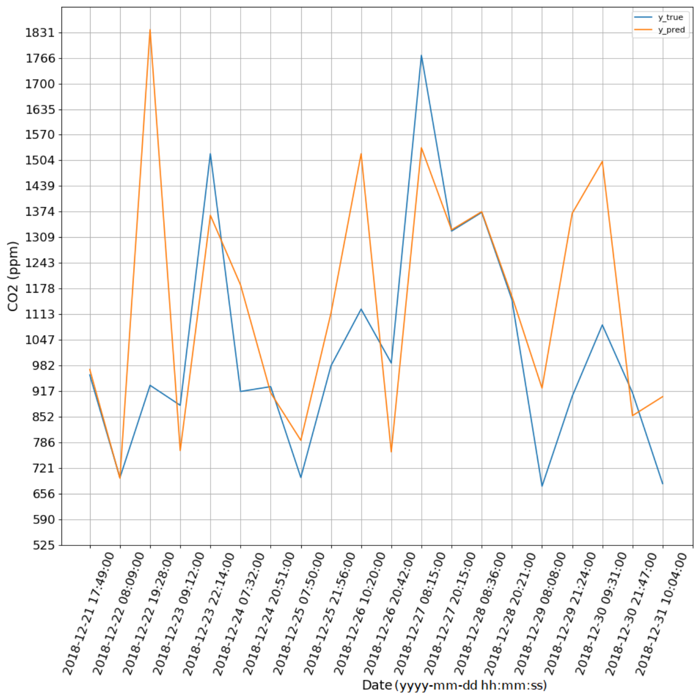


Figure 16: Predictions made by the best voting ensemble model. The data is from the kitchen. Train and test size: 1440 instances (one day). R2 score: 0.47, MAE: 0.12, MSE: 0.03, RMSE: 0.18

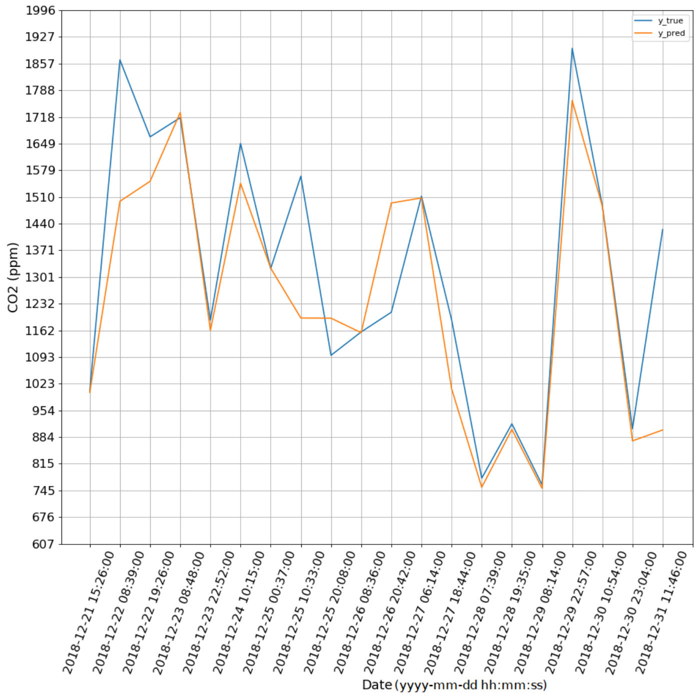


Figure 17: Predictions made by the best voting ensemble model. The data is from the bathroom. Train and test size: 1440 instances (one day). R2 score: 0.61, MAE: 0.1, MSE: 0.02, RMSE: 0.16

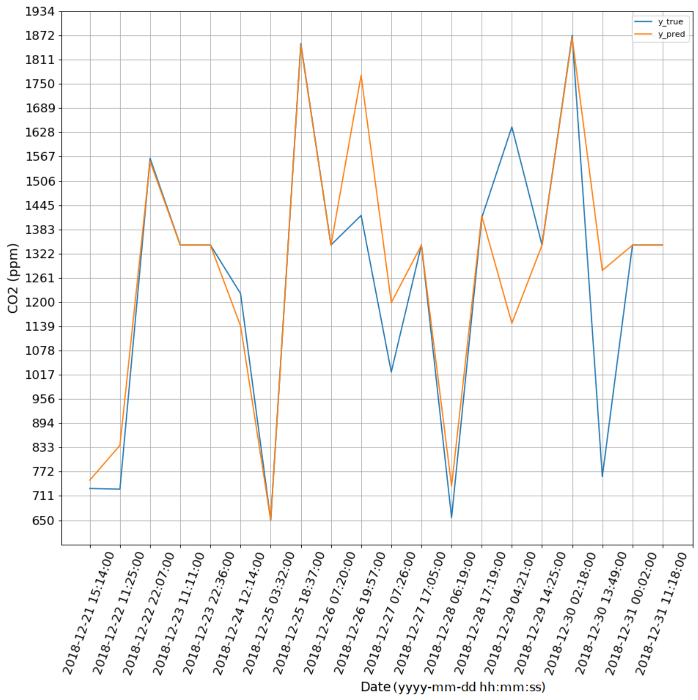


Figure 18: Predictions made by the best voting ensemble model. The data is from the kidsroom. Train and test size: 1440 instances (one day). R2 score: 0.77, MAE: 0.05, MSE: 0.01, RMSE: 0.11

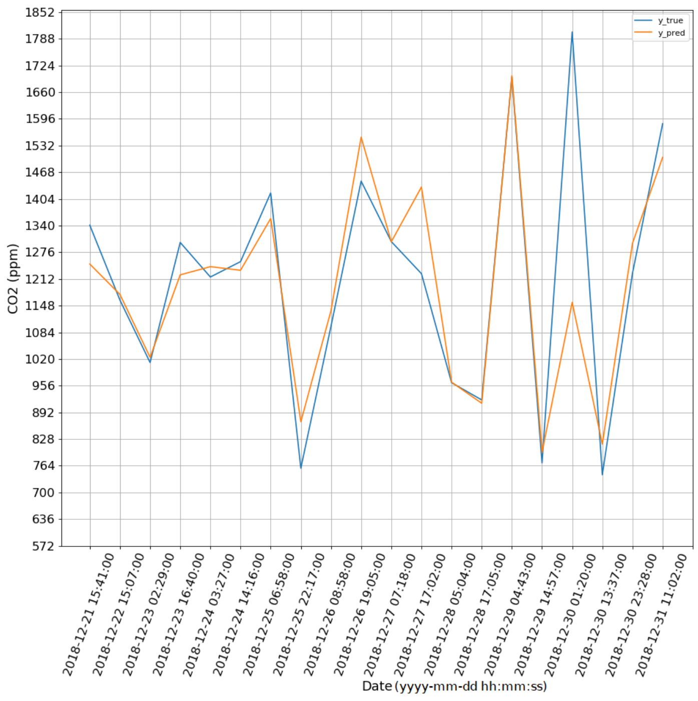


Figure 19: Predictions made by the best voting ensemble model. The data is from the living room. Train and test size: 1440 instances (one day). R2 score: 0.74, MAE: 0.06, MSE: 0.01, RMSE: 0.12

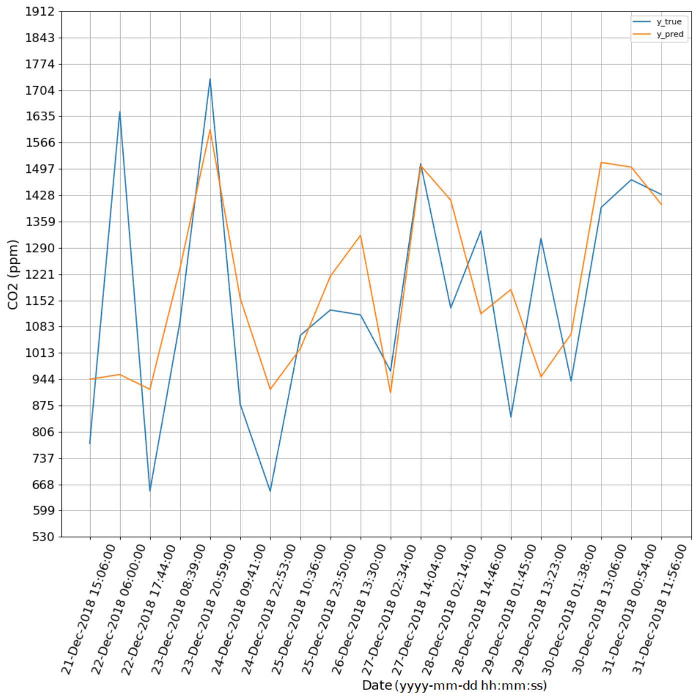


Figure 20: Predictions made by the best bagging ensemble model. The data is random instances from different rooms. Train and test size: 1440 instances (one day). R2 score: 0.37, MAE: 0.15, MSE: 0.03, RMSE: 0.18

### 6.3.2 Working without kitchen and bathroom rooms

Comparing results achieved by the ensemble working with and without the kitchen and bathroom, we found a performance improvement of 18% in the mean of the R2 score. Both rooms have elements (Cooking stove, shower, etcetera), which make drastic changes in the levels of temperature and relative humidity measures made by the sensors. The same situation happens with the bagging ensemble.

| r2 | mae | mse | rmse | train size | test size | file |
|---|---|---|---|---|---|---|
| 0.8222 | 0.0612 | 0.0129 | 0.1136 | 1440 | 1440 | bedroom |
| 0.8343 | 0.0691 | 0.012 | 0.1093 | 7200 | 7200 | bedroom |
| 0.8193 | 0.0429 | 0.0113 | 0.1061 | 1440 | 1440 | kidsroom |
| 0.8274 | 0.0516 | 0.011 | 0.1051 | 7200 | 7200 | kidsroom |
| 0.7872 | 0.058 | 0.0126 | 0.1124 | 1440 | 1440 | livingroom |
| 0.7859 | 0.0679 | 0.0125 | 0.1116 | 7200 | 7200 | livingroom |

Table 6: Voting ensemble results working without bathroom and kitchen rooms. The data is from the bedroom. The results from this ensemble have an average R2 score of 0.80

It is easy to see that the voting ensemble seems more efficient than the bagging ensemble in any situation presented. However, even if the voting ensemble works better, maybe with another data processing to create the bagging ensemble, the performance could improve. Even so, the results obtained by the bagging ensemble have an improvement of 10%, which is not remarkable, but it is still an improvement.

| r2 | mae | mse | rmse | train size | test size | file |
|---|---|---|---|---|---|---|
| 0.4754 | 0.1368 | 0.0314 | 0.1772 | 1440 | 1440 | merged data |
| 0.4577 | 0.1388 | 0.0324 | 0.18 | 7200 | 7200 | merged data |

Table 7: Bagging ensemble results working without bathroom and kitchen rooms. The data is random instances from different rooms. The results from this ensemble have an average R2 score of 0.46

Nevertheless, working without the bathroom and kitchen rooms made the metrics MSE and RMSE higher than working with them. This means that these models have a worse fitting to the data. However, the higher R2 score indicates that most of the data used as the test can be explained, about 83% of the data can be explained.

The problem of removing both rooms means that we are not making a complete ensemble model, so the option of creating a model for each room seems to be a better option from the beginning.

Another option would be to group rooms by the similarity of use, elements, or dimensions. In this way, it would not be necessary to create a model per room. Perhaps with a model for the living room, bedroom, and children's room set, and two other different models for the kitchen and bathroom could be enough to reduce the number of models created.

However, this option has not been implemented as it has already been implicitly developed. We say it has already been developed because by removing the bathroom and kitchen to create this ensembles, we have already seen an improvement. Also, in section 6.3.3, we have seen the results of the individual models for the kitchen and bathroom.

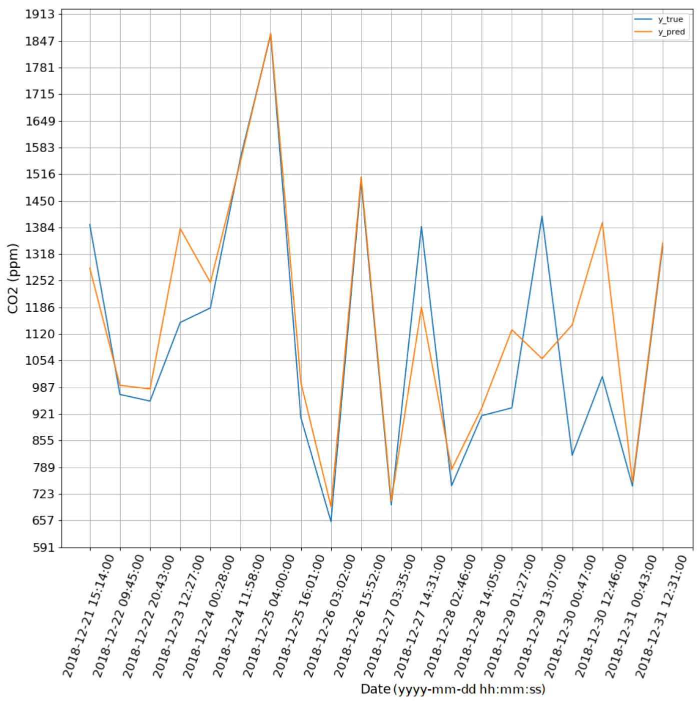


Figure 21: Predictions made by the best voting ensemble model working without bathroom and kitchen rooms. The data is from the bedroom. Train and test size: 7200 instances (one day). R2 score: 0.83, MAE: 0.06, MSE: 0.01, RMSE: 0.1

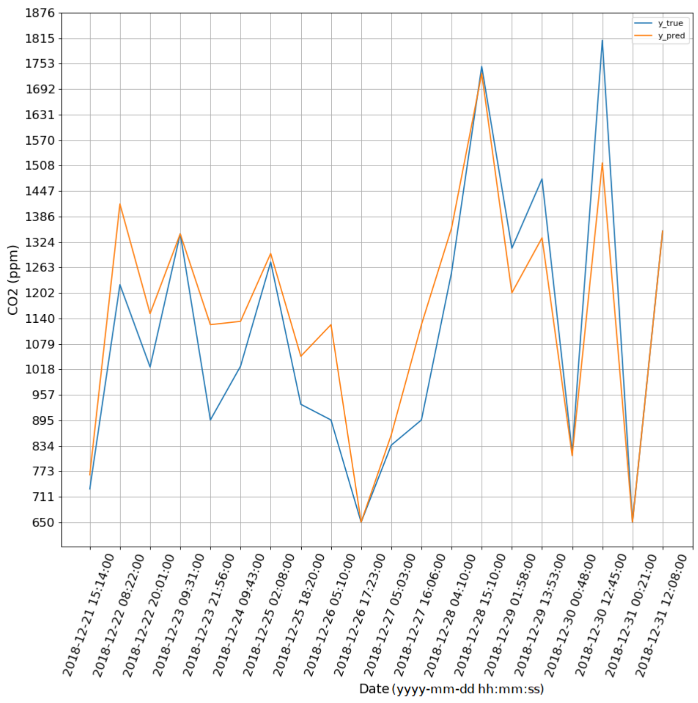


Figure 22: Predictions made by the best voting ensemble model working without bathroom and kitchen rooms. The data is from the kidsroom. Train and test size: 7200 instances (one day). R2 score: 0.82, MAE: 0.05, MSE: 0.01, RMSE: 0.1

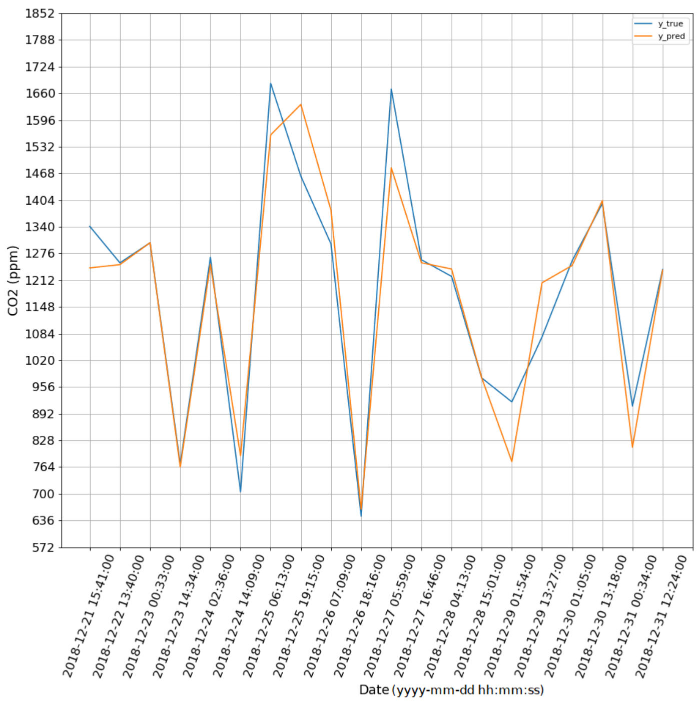


Figure 23: Predictions made by the best voting ensemble model working without bathroom and kitchen rooms. The data is from the living room. Train and test size: 7200 instances (one day). R2 score: 0.78, MAE: 0.06, MSE: 0.01, RMSE: 0.1

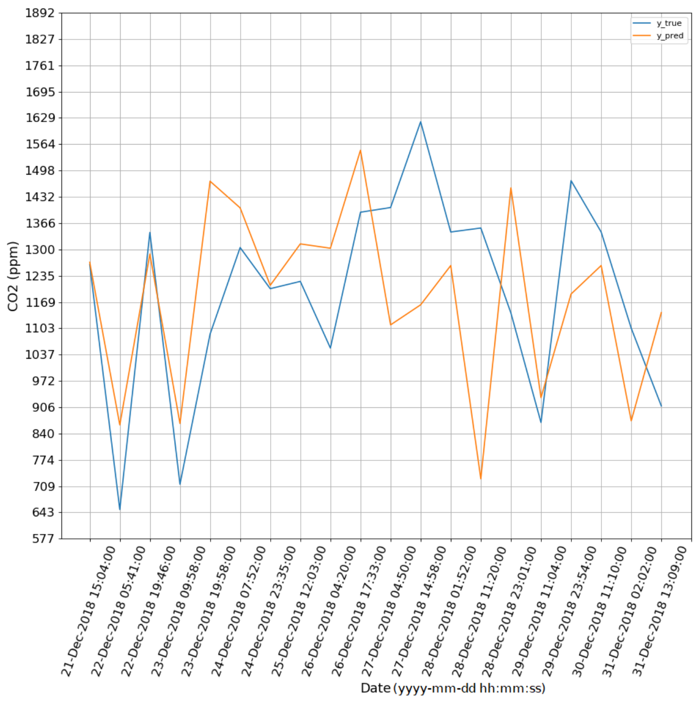


Figure 24: Predictions made by the best bagging ensemble model working without bathroom and kitchen rooms. Train and test size: 1440 instances (one day). R2 score: 0.47, MAE: 0.13, MSE: 0.03, RMSE: 0.18

### 6.3.3 One model per room

Although the ensembles are more robust and stable models, it seems that to work with this problem, the most efficient way is to work with each room individually, creating a model for each one of them. The results of working with a model per room are shown in table 3. These results represent a fairly higher performance improvement, up to 11% improvement in predicting CO2 levels. This happens because apart from the fact that there are elements in each room that alter the measurements taken by the sensors, each room has different sizes and quite different usage patterns. This increases the disparity between rooms data, which means that it is more challenging to create a model that is capable of predicting CO2 values for all rooms.

These tests are focused on five rooms with a range of ten days of measurements made by the sensors. However, with more data, the models could become more accurate and robust. Even ensemble models could get better results.

In spite of everything, the results seen by the different approaches to the problem are quite good considering that from 0.7 of the R2 score, a model can be viewed as a good model. So we could consider that of the three approaches developed, two of them are quite good. And between these two, the best results have been seen in the approximation of one model per room.

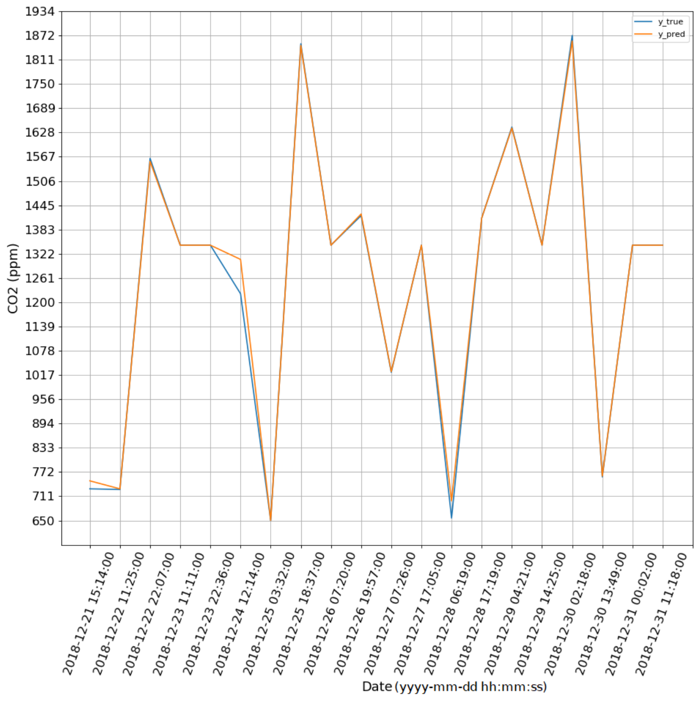


Figure 25: Predictions made by creating one model per room. The data is from the kidsroom. Train and test size: 1440 instances (one day). R2 score: 0.94, MAE: 0.01, MSE: 0.003, RMSE: 0.0599

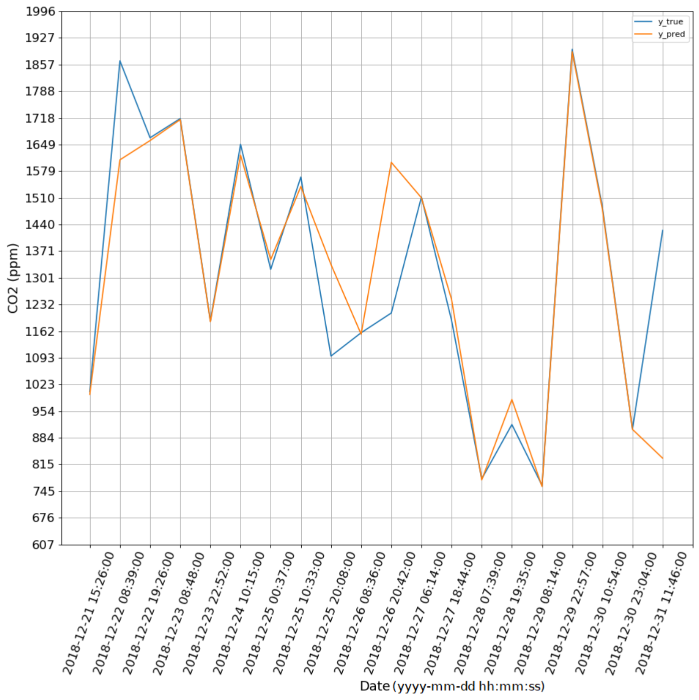


Figure 26: Predictions made by creating one model per room. The data is from the bahtroom. Train and test size: 1440 instances (one day). R2 score: 0.83, MAE: 0.04, MSE: 0.01, RMSE: 0.1

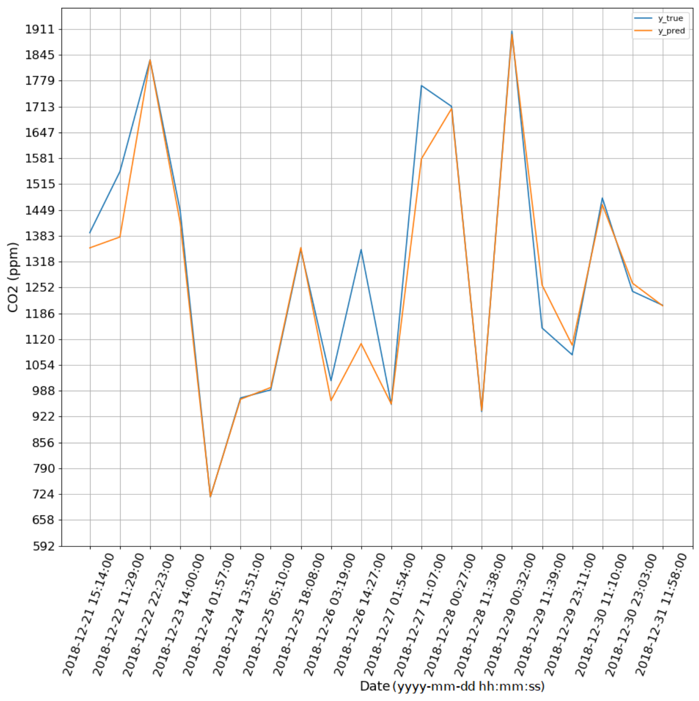


Figure 27: Predictions made by creating one model per room. The data is from the bedroom. Train and test size: 1440 instances (one day). R2 score: 0.86, MAE: 0.04, MSE: 0.008, RMSE: 0.09

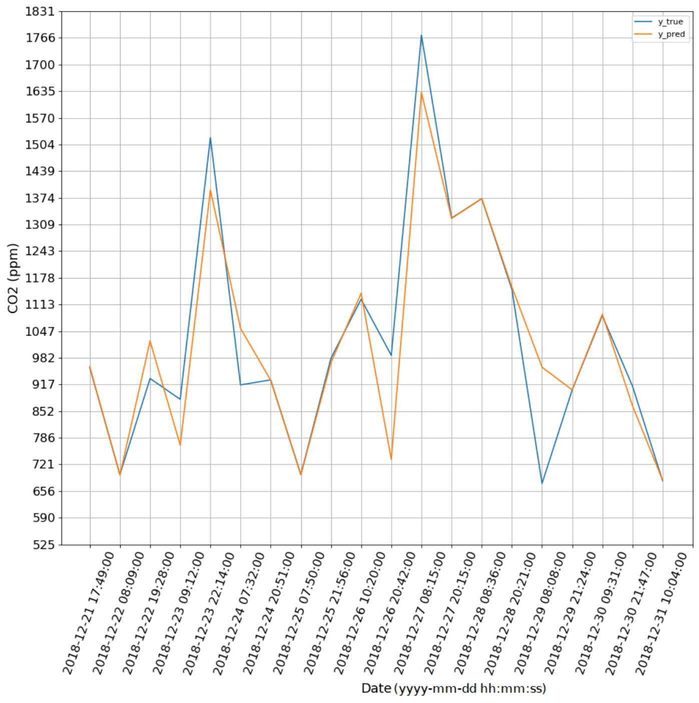


Figure 28: Predictions made by creating one model per room. The data is from the kitchen. Train and test size: 1440 instances (one day). R2 score: 0.73, MAE: 0.06, MSE: 0.01, RMSE: 0.12

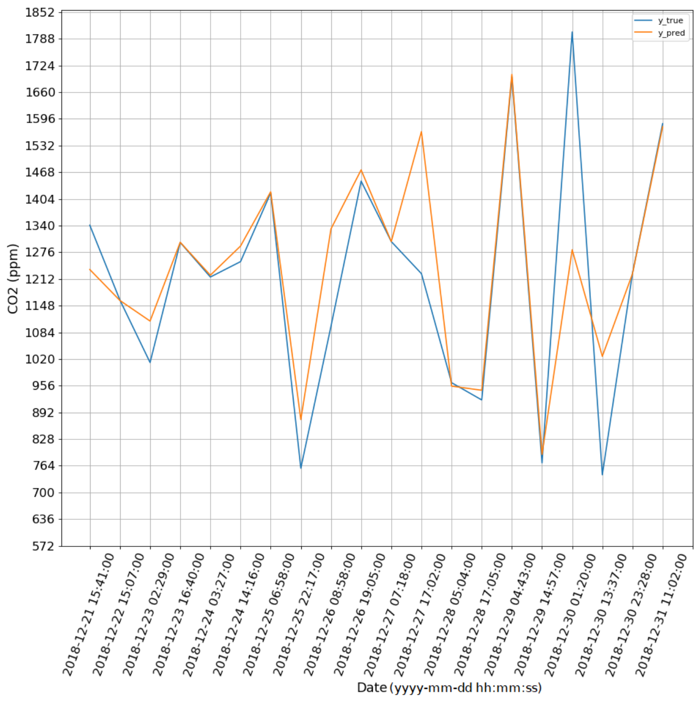


Figure 29: Predictions made by creating one model per room. The data is from the kitchen. Train and test size: 1440 instances (one day). R2 score: 0.79, MAE: 0.05, MSE: 0.08, RMSE: 0.09

In figure 25, is shown how an R2 score of 0.93 is almost a success in the predictions of patterns used for the test. However, if we look at table 3, when we train with five days, and we predict five days, the results are generally lower, this happens since the model is overfitted.

# 7 Conclusions

## 7.1 Analysis of the work done

In this thesis, the support vector machine for the regression algorithm has been analyzed for the identification of operational technical statuses in a smart home. This algorithm has been viewed from a theoretical and practical implementation point of view.

Data preprocessing has been performed, reducing dimensions for faster computing speed and obtaining better results in the algorithm. Noise reduction, data scale, and normalization have been applied.

Thanks to the different approaches to the problem, it has been possible to compare and see which ones are better for this problem. Two of these implementations got nice results, although the voting ensemble got lower results than creating one model per room. For each implementation the parameters C, Gamma, and Epsilon have been estimated.

Different cloud services have additionally been analyzed and investigated, and a theoretical implementation of a hardware and software interface for the connection between KNX sensors and IoT cloud services has been proposed.

## 7.2 Future development

The Python scripts have been completed and are currently working. One of the possible future developments could be the implementation of the theoretical explained implementation of the connection between the KNX sensors and the IoT cloud services, using the libraries explained and the hardware proposed.

Another possible future development could be the optimization of the hardware recourse used, moreover, the parallel and distributed implementation of the scripts to use multiple CPU cores to speed up the process of training and testing.

Furthermore, the use of data with a longer time range could be beneficial for model training and testing.

## References

1. LASHKARI, Behzad; CHEN, Yuxiang; MUSILEK, Petr. *Energy Management for Smart Homes-State of the Art*. SWITZERLAND: MDPI, 2019.
2. PENG, Yuzhen; RYSANEK, Adam; NAGY, Zoltán; SCHLÜTER, Arno. Using machine learning techniques for occupancy-prediction-based cooling control in office buildings. *Applied Energy*. 2018, vol. 211, pp. 1343–1358. ISSN 0306-2619. Available from DOI: `https://doi.org/10.1016/j.apenergy.2017.12.002`.
3. WANG, Shengwei. Intelligent buildings and building automation. *Intelligent Buildings and Building Automation*. 2009-01, pp. 1–248. Available from DOI: `10.4324/9780203890813`.
4. ZADEH, LOTFI A. Soft Computing and Fuzzy Logic. In: *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems*, pp. 796–804. Available from DOI: `10.1142/9789814261302_0042`.
5. PRATIHAR, D.K. *Soft Computing*. S.l. Alpha Science International, Ltd, 2007.
6. WOLPER, David H.; MACREADY, William G. *No Free Lunch Theorems for Optimization*. San Jose: IBM Almaden Research Center, 1996.
7. BLUMTRITT, Christoph. *Smart Home Report 2019*. London: Statista, 2019.
8. VADILLO, Laura. The Role of Smart Homes in Intelligent Homecare and Healthcare Environments. In: *book auth.] Health Sciences Elsevier. Ambient Assisted Living and Enhanced Living Environments*. Woburn, United States: Butterworth-Heinemann, 2016.
9. CLARK, Jordan D.; LESS, Brennan D.; DUTTON, Spencer M.; WALKER, Lain S.; SHERMAN, Max H. (eds.). *Efficacy of occupancy-based smart ventilation control strategies in energy-efficient homes in the United States*. California: BUILDING and ENVIRONMENT, 2019.
10. BELTRAN, Alex; CERPA, Alberto E. Optimal HVAC Building Control with Occupancy Prediction. In: *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. New York: Association for Computing Machinery, 2014.
11. A machine learning approach for indirect human presence detection using IOT devices. In: *2016 Eleventh International Conference on Digital Information Management (ICDIM*. 2016.
12. RAZAVI, Rouzbeh; GHARIPOUR, Amin; FLEURY, Martin; AKPAN. Occupancy detection of residential buildings using smart meter data: A. *Ikpe Justice. s.l. : Elsevier*. 2019, vol. 183, pp. 0378–7788.
13. SHIH, Huang-Chia. A robust occupancy detection and tracking algorithm for the automatic monitoring and commissioning of a building. *Energy and Buildings*. 2014.
14. HARROU, F.; ZERROUKI, N.; SUN, Y.; HOUACINE, A. An Integrated Vision-Based Approach for Efficient Human Fall Detection in a Home Environment. *IEEE Access*. 2019, vol. 7, pp. 114966–114974. ISSN 2169-3536. Available from DOI: `10.1109/ACCESS.2019.2936320`.

15. BARBOUR, E.; DAVILA, C.C.; GUPTA, S.; REINHART, C.; KAUR, J.; GONZÁLEZ (eds.). *Planning for sustainable cities by estimating building occupancy with mobile phones*. Nature Publishing Group, 2019.

16. FAHAD, Labiba Gillani; KHAN, Asifullah; RAJARAJAN, Muttukrishnan sl (eds.). *Activity recognition in smart homes with self verification of assignments*. Neurocomputing, 2015.

17. *Monitoring Activities of Daily Living in Smart Homes: Understanding human behavior*. 2, s.l. IEEE, 2016.

18. CALÌ, Davide; MATTHES, Peter; HUCHTEMANN, Kristian; STREBLOW, Rita; MÜLLER, Dirk. CO2 based occupancy detection algorithm: Experimental analysis and validation for office and residential buildings. *Building and Environment*. 2015, vol. 86, pp. 39–49. ISSN 0360-1323. Available from DOI: `https://doi.org/10.1016/j.buildenv.2014.12.011`.

19. ARIEF-ANG, Irvan; HAMILTON, Margaret; SALIM, Flora. A Scalable Room Occupancy Prediction with Transferable Time Series Decomposition of CO2 Sensor Data. *ACM Transactions on Sensor Networks*. 2018-11, vol. 14. Available from DOI: `10.1145/3217214`.

20. BRENNAN, Colin; TAYLOR, Graham; SPACHOS, P.sl (eds.). *Designing Learned CO2-based Occupancy Estimation in Smart Buildings*. IET Wireless Sensor Systems, 2018.

21. CANDANEDO, Luis M.; FELDHEIM, Véronique sl (eds.). *Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models*. Energy and Buildings, 2016.

22. Vázquez and W. Clustering methods for occupancy prediction in smart home control. In: *2011 IEEE International Symposium on Industrial Electronics*. 2011.

23. ANAND, P.; CHEONG, D.; SEKHAR, C.; SANTAMOURIS, M.; KONDEPUDI, S.sl (eds.). *Energy saving estimation for plug and lighting load using occupancy analysis*. Elsevier, 2019.

24. WANG, Wei; CHEN, Jiayu; HONG, Tianzhen (eds.). *Occupancy prediction through machine learning and data fusion of environmental sensing and Wi-Fi sensing in buildings*. 2018.

25. 2020. Available also from: `https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-services-and-technologies`.

26. 2020. Available also from: `https://cloud.google.com/solutions/iot`.

27. 2020. Available also from: `https://thingspeak.com/pages/commercial_learn_more`.

28. BURHAN, Muhammad; REHMAN, Rana Asif; KIM, Byung-Seo; KHAN, Bilal. IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey. *Sensors*. 2018-08, vol. 18. Available from DOI: `10.3390/s18092796`.

29. 2020. Available also from: `https://www.helium.com/`.

30. KREUTZ, Florian. *KNX BAOS Module 838 kBerry*. 2020. Available also from: `https://www.weinzierl.de/index.php/en/all-knx/knx-module-en/knx-baos-module-838-en`.

31. 2020. Available also from: `https://github.com/bobaos/bobaos`.

32. CORTES, Corinna; VAPNIK, Vladimir. Support-Vector Networks. *Mach. Learn.* 1995-09, vol. 20, no. 3, pp. 273–297. ISSN 0885-6125. Available from DOI: `10.1023/A:1022627411411`.

33. HEFFERON, Jim. *LINEAR ALGEBRA*. Colchester, Vermont: Mathematics and Statistics Department Saint Michael's College, 2017.

34. PARRELLA, Francesco. Online Support Vector Regression. In: 2007.

35. HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. Available also from: `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`.

36. HUH, Myung-Hoe. Kernel-Trick Regression and Classification. *Communications for Statistical Applications and Methods*. 2015-03, vol. 22, pp. 201–207. Available from DOI: `10.5351/CSAM.2015.22.2.201`.

37. MINH, Ha Quang; NIYOGI, Partha; YAO, Yuan. Mercer's theorem, feature maps, and smoothing. In: *International Conference on Computational Learning Theory*. 2006, pp. 154–168.

38. HOFMANN, Thomas; SCHÖLKOPF, Bernhard; SMOLA, Alexander J. Kernel methods in machine learning. *The Annals of Statistics*. 2008-07, vol. 36, no. 3, pp. 1171–1220. ISSN 0090-5364. Available from DOI: `10.1214/009053607000000677`.

39. GENTON, Marc G. Classes of Kernels for Machine Learning: A Statistics Perspective. *J. Mach. Learn. Res.* 2002-03, vol. 2, pp. 299–312. ISSN 1532-4435.

40. ALPAYDIN, Ethem. Introduction to Machine Learning. In: 2004-01, vol. 56.

41. CELIKYILMAZ, Asli; TRKSEN, I. Burhan. *Modeling Uncertainty with Fuzzy Logic: With Recent Theory and Applications*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN 3540899235.

42. HAN, Jiawei; KAMBER, Micheline; PEI, Jian. *Data mining concepts and techniques, third edition*. Waltham, Mass.: Morgan Kaufmann Publishers, 2012. ISBN 0123814790.

43. SAVITZKY, Abraham.; GOLAY, M. J. E. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*. 1964, vol. 36, no. 8, pp. 1627–1639. Available from DOI: `10.1021/ac60214a047`.

44. HARRISON, M. *Machine Learning Pocket Reference: Working with Structured Data in Python*. O'Reilly Media, 2019. ISBN 9781492047513.

45. ZHANG, Cha; MA, Yunqian. *Ensemble Machine Learning: Methods and Applications*. Springer Publishing Company, Incorporated, 2012. ISBN 1441993258.

46. HSU, Chih-Wei; CHANG, Chih-Chung; LIN, Chih-Jen. *A Practical Guide to Support Vector Classification*. 2003. Available also from: `http://www.csie.ntu.edu.tw/~cjlin/papers.html`. Technical report. Department of Computer Science, National Taiwan University.

# A Code implementation

## A.1 Ensemble voting

```
data_path = 'data/'
onlyfiles = [f for f in listdir(data_path) if isfile(join(data_path, f))]

output = []

test_size = [1440, 7200]

for i in test_size:

    dict_test_X = {}
    dict_test_y = {}
    dict_test_dates = {}

    list_train_X = []
    list_train_y = []

    estimators = []

    for file in onlyfiles:

        #Read file and change date type
        filename = os.path.splitext(file)[0]
        dataset = pd.read_csv('data/'+filename+'.csv', delimiter=";", parse_dates=True)
        dataset['Time'] = pd.to_datetime(dataset.Time)
        print("Working with file: ", filename)

        #Remove outliers
        dataset = wrapper.remove_outliers(dataset)

        # Filter dataset
        dataset = wrapper.filter_data(dataset, 21, 1)
        #Get min and max
        min_CO2=dataset['CO2'].min()
        max_CO2=dataset['CO2'].max()

        #Split between train and test
        data_train, data_test = train_test_split(dataset, train_size=i, test_size=i, random_state=55, shuffle =
            True)

        #Split between X and y
        X_train = data_train.drop(['CO2', 'Time'], axis=1).copy()
        y_train = data_train['CO2']

        X_test = data_test.drop(['CO2', 'Time'], axis=1).copy()
    y_test = data_test [['CO2', 'Time']]

    date_test = y_test ['Time']

    y_test = y_test ['CO2'].to_numpy()

    #Normalize data
    min_H = X_train['H'].min()
    max_H = X_train['H'].max()

    min_T = X_train['T'].min()
    max_T = X_train['T'].max()

    X_train['H'] = (X_train['H']−min_H)/(max_H−min_H)
    X_train['T'] = (X_train['T']−min_T)/(max_T−min_T)

    X_test['H'] = (X_test['H']−min_H)/(max_H−min_H)
    X_test['T'] = (X_test['T']−min_T)/(max_T−min_T)

    # Save the train data in order to train the ensemble later
    list_train_X .append(X_train)
    list_train_y .append( y_train )

    # Save the test data in order to test the ensemble later
    dict_test_X [filename] = X_test
    dict_test_y [filename] = y_test
    dict_test_dates [filename] = date_test

    #Scale data
    sc_X = StandardScaler ()
    X_train = sc_X. fit_transform (X_train)
    X_test = sc_X.transform(X_test)

    # Select tuning parameter range
    gamma_range = np.logspace(0.0001,9,base=2, num=40)
    C_range = np.logspace (0.0001,9, base=2, num=40)
    epsilon_range = [0.01, 0.05, 0.1, 0.125, 0.2, 0.5, 0.7, 0.9]
    tuned_parameters = [{'kernel': ['rbf'], 'gamma': gamma_range, 'C': C_range, 'epsilon': epsilon_range }]

    #Regressor parameter estimation
    regressor = RandomizedSearchCV(SVR(), tuned_parameters, scoring='r2', cv=3, verbose=1, n_iter=10)
    regressor . fit (X_train, y_train )

    estimators .append([filename, regressor . best_estimator_ ])

    #Predict test values
    y_true, y_pred = y_test, regressor . best_estimator_ . predict (X_test)

```

```
        #Normalize outputs to get normalized scores
        y_true_norm = [ ((i-min_CO2)/(max_CO2-min_CO2)) for i in y_true ]
        y_pred_norm = [ ((i-min_CO2)/(max_CO2-min_CO2)) for i in y_pred ]

        #Calculate the score
        results = regression_results (y_true_norm, y_pred_norm)
        results['train_size'] = i
        results['test_size'] = i
        results['file'] = filename
        print(pd.DataFrame(results, index=[0]))

    #Concat the train data in order to use for training the ensemble
    ens_X_train = pd.concat( list_train_X )
    ens_y_train = pd.concat( list_train_y )

    #Scale train data
    sc_X = StandardScaler()
    ens_X_train = sc_X.fit_transform(ens_X_train)

    # Create ensemble model and fit train data in it
    print('Fitting data ...')
    ensemble_model = VotingRegressor( estimators=estimators )
    ensemble_model.fit(ens_X_train, ens_y_train)

    #Test data from the different rooms
    for X_index, y_index, date_index in zip( dict_test_X, dict_test_y, dict_test_dates ):
        X_test = sc_X.transform( dict_test_X[X_index])
        y_test = dict_test_y[y_index]
        date_test = dict_test_dates[date_index]
        y_true, y_pred = y_test, ensemble_model.predict(X_test)

        #Normalize outputs to get normalized scores
        y_true_norm = [ ((i-min_CO2)/(max_CO2-min_CO2)) for i in y_true ]
        y_pred_norm = [ ((i-min_CO2)/(max_CO2-min_CO2)) for i in y_pred ]

        results = wrapper.regression_results (y_true_norm, y_pred_norm)
        results['train_size'] = i
        results['test_size'] = i
        results['file'] = X_index
        output.append( results )
        wrapper.plot_accuracy(y_true, y_pred, int(y_true.size/30), i, i, results['r2'], X_index, sorted(
            date_test))

results_df = pd.DataFrame.from_dict(output)
print( results_df )

results_df.to_csv('predictions/ensemble_voting.csv', sep=';')
```

Code Listing 1: Python script for the creation of an ensemble using voting ensemble approach explained in section 5.4

## A.2 Ensemble bagging

```

#Read file
filename = 'merged'
dataset = pd.read_csv('merged_data/'+filename+'.csv', delimiter=";", parse_dates=True)
dataset = dataset.drop(['id'], axis=1)

dataset = wrapper.remove_outliers(dataset)

#Get min and max
min_CO2 = dataset['CO2'].min()
max_CO2 = dataset['CO2'].max()

# Filter dataset data
dataset = wrapper.filter_data(dataset, 21, 1)

output = []
tests = [1440, 1440*5]
regressor = []
for i in range(len(tests)):
    # Split between train and test
    data_train, data_test = train_test_split(dataset, train_size=tests[i], test_size=tests[i], random_state=55,
        shuffle = True)

    # Split between X and y
    X_train = data_train.drop(['CO2', 'Time'], axis=1).copy()
    y_train = data_train['CO2']

    X_test = data_test.drop(['CO2', 'Time'], axis=1).copy()
    y_test = data_test[['CO2', 'Time']]

    date_test = y_test['Time']

    y_test = y_test['CO2'].to_numpy()

    #Normalize data
    min_H = X_train['H'].min()
    max_H = X_train['H'].max()

    min_T = X_train['T'].min()
    max_T = X_train['T'].max()

    X_train['H'] = (X_train['H']-min_H)/(max_H-min_H)
    X_train['T'] = (X_train['T']-min_T)/(max_T-min_T)

    X_test['H'] = (X_test['H']-min_H)/(max_H-min_H)
    X_test['T'] = (X_test['T']-min_T)/(max_T-min_T)
```

```python

    #Scale data
    sc_X = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    X_test = sc_X.transform(X_test)

    #If first iteration , create the model and estimate parameters
    if i ==0:
        rng = check_random_state(0)
        grid = ParameterGrid({"max_samples": [0.5, 1.0],
                              "max_features": [0.5, 1.0],
                              "bootstrap": [True, False]})
        for params in grid:
            regressor = BaggingRegressor(base_estimator=SVR(),
                              verbose=10,
                              random_state=rng,
                              **params).fit(X_train, y_train)

    #Predict the test data with the best values found
    y_true, y_pred = y_test, regressor.predict(X_test)

    #Normalize outputs to get normalized scores
    y_true_norm = [ ((i-min_CO2)/(max_CO2-min_CO2)) for i in y_true]
    y_pred_norm = [ ((i-min_CO2)/(max_CO2-min_CO2)) for i in y_pred]

    #Save the results for each test
    results = wrapper.regression_results(y_true_norm, y_pred_norm)
    results['train_size'] = tests[i]
    results['test_size'] = tests[i]
    results['file'] = 'merged'
    output.append(results)
    wrapper.plot_accuracy(y_true, y_pred, int(y_pred.size/30), tests[i], tests[i], results['r2'], 'merged',
        sorted(date_test))

results_df =pd.DataFrame.from_dict(output)
print(results_df)
results_df.to_csv('predictions/ensemble_bagging.csv', sep=';')
```

Code Listing 2: Python script for the creation of an ensemble using bagging ensemble approach explained in section 5.4

## A.3 One model per room

```
data_path = 'data/'
onlyfiles = [f for f in listdir(data_path) if isfile(join(data_path, f))]

output = []
for file in onlyfiles:

    #Read file and change date type
    filename = os.path.splitext(file)[0]
    dataset = pd.read_csv('data/'+filename+'.csv', delimiter=";", parse_dates=True)
    dataset['Time'] = pd.to_datetime(dataset.Time)

    dataset = wrapper.remove_outliers(dataset)

    # Filter dataset
    dataset = wrapper.filter_data(dataset, 21, 1)

    test_size = [1440, 1440*5]
    for i in test_size:

        #Split between X and y
        X = dataset.drop(['CO2', 'Time'], axis=1)
        y = dataset[['CO2', 'Time']]

        #Split between train and test
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=i, test_size=i, random_state=55,
            shuffle = True)
        X_train = X_train.copy()
        X_test = X_test.copy()
        date_test = y_test['Time']

        y_test = y_test['CO2'].to_numpy()
        y_train = y_train['CO2'].to_numpy()

        #Scale data
        min_H = X_train['H'].min()
        max_H = X_train['H'].max()

        min_T = X_train['T'].min()
        max_T = X_train['T'].max()

        X_train['H'] = (X_train['H']-min_H)/(max_H-min_H)
        X_train['T'] = (X_train['T']-min_T)/(max_T-min_T)

        X_test['H'] = (X_test['H']-min_H)/(max_H-min_H)
        X_test['T'] = (X_test['T']-min_T)/(max_T-min_T)

```

```
            sc_X = StandardScaler()
            X_train = sc_X.fit_transform(X_train)
            X_test = sc_X.transform(X_test)

            # Select tuning parameter range
            gamma_range = np.logspace(0.0001,9,base=2, num=40)
            C_range = np.logspace(0.0001,9, base=2, num=40)
            epsilon_range = [0.01, 0.05, 0.1, 0.125, 0.2, 0.5, 0.7, 0.9]
            tuned_parameters = [{'kernel': ['rbf'], 'gamma': gamma_range, 'C': C_range, 'epsilon': epsilon_range }]

            #Regressor parameter estimation
            regressor = RandomizedSearchCV(SVR(), tuned_parameters, scoring='r2', cv=3, verbose=1, n_iter=10)
            regressor.fit(X_train, y_train)

            #Predict test values
            y_true, y_pred = y_test, regressor.predict(X_test)

            #Calculate the score
            results = wrapper.regression_results(y_true, y_pred)
            results['train_size'] = i
            results['test_size'] = i
            results['file'] = filename
            output.append(results)
            wrapper.plot_accuracy(y_true, y_pred, int(y_true.size/50), i, i, results['r2'], filename, sorted(
                date_test))

results_df = pd.DataFrame.from_dict(output)
print(results_df)
results_df.to_csv('predictions/results_shuffle.csv', sep=';')
```

Code Listing 3: Python script for the creation of an ensemble using one model per room approach explained in section 5.4

## A.4 Wrapper

```
def plot_accuracy(y_true, y_pred, xax_, t_s, train_size , score, filename, folder, dates):
    """Function that plots the predicted and true values on the y ax, and the date in the x ax.
    Also saves the true and predicted in a csv file .

    Arguments:
        y_true { list } -- list with true values of CO2
        y_pred { list } -- list with predicted values of CO2
        xax_ { int } -- step of the y ax
        t_s { float } -- test size
         train_size { float } -- train size
        score { float } -- R2 score obtained
        filename { string } -- name of the file which has been tested
        folder { string } -- name of the folder where the plot is going to be saved
        dates { list } -- list of date to print in the x ax
    """
    xax=xax_
    fig=plt.figure(figsize=(25, 12), dpi= 80, facecolor='w', edgecolor='k')
    plt.plot(y_true[0::xax], label='y_true')
    plt.plot(y_pred[0::xax], label='y_pred')
    min_y_bound = min(min(y_true), min(y_true))
    max_y_bound = max(max(y_true), max(y_true))
    step = abs(max_y_bound-min_y_bound)/30

    plt.yticks(np.arange(min_y_bound, max_y_bound+step, step), fontsize=15)
    plt.xticks(np.arange(0, y_true[0::xax].size+1,1), dates[0::xax], rotation=70, fontsize=15)
    plt.xlabel('Date', fontsize=16)
    plt.ylabel('CO2 (ppm)', fontsize=16)
    plt.legend()
    plt.grid()
    plt.tight_layout()

    plt.savefig("graphs/"+folder+"/"+filename+"_score{score:.3f}_test{test_s:d}_train{train:d}.png".format(
        test_s=t_s, score = score, train=train_size))
    fname = "predictions/"+folder+"/"+filename+"_score{score:.3f}_test{test_s:d}_train{train:d}.csv".format(
        test_s=t_s, score = score, train=train_size )
    y_true = np.ravel(y_true)
    y_pred = np.ravel(y_pred)
    df = pd.DataFrame({"y_true" : y_true, "y_pred" : y_pred})
    df.to_csv(fname, index=False, sep=";")

def regression_results (y_true, y_pred):
    """Function that calculates the different scores used to compare the models.

    Arguments:
        y_true { list } -- list with true values of CO2
        y_pred { list } -- list with predicted values of CO2

    Returns:
        dictionary -- dictionary with all the calculated scores
    """
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    metrics_dic = {'r2':round(r2,4), 'mae':round(mean_absolute_error,4), 'mse':round(mse,4), 'rmse':round(np.
        sqrt(mse),4)}
    return metrics_dic

def filter_data(data, wl, po):
    """Function that applies Savitzky Golay filter to the data

    Arguments:
        data {dataframe} -- Dataframe with all the data
        wl {int} -- Windows size to apply the filter
        po {int} -- Polynomial order to apply the filter

    Returns:
        dataframe -- Dataframe with all the data filtered
    """
     dataset_filtered = data.copy()
     dataset_filtered['CO2']= savgol_filter(dataset_filtered['CO2'], wl, po)
     dataset_filtered['H']= savgol_filter(dataset_filtered['H'], wl, po)
     dataset_filtered['T']= savgol_filter(dataset_filtered['T'], wl, po)
    return dataset_filtered

def remove_outliers(dataset, threshold):
    """Function that detect the outliers from the dataset and removes them

    Arguments:
        dataset {dataframe} -- Dataframe with all the data
        threshold {float} -- Threshold to consider a data point as outlier

    Returns:
        dataframe -- dataframe without outliers
    """
    z = np.abs(stats.zscore(dataset.iloc[:,1:4]))
    return dataset[(z < threshold).all(axis=1)]
```

Code Listing 4: Python script with the common functions for all the scripts coded