

POLITECNICO DI MILANO
Master's degree in Computer Science and Engineering
Department of Electronics, Information and Bioengineering



Machine learning methods for indoor occupancy detection with Co2 multi-sensor data

ANTLab
Advanced Network Technologies LABoratory

Advisor: Prof. Matteo Cesana

Co-advisor: Prof. Alessando Redondi, Edoardo Longo

Master's Degree Thesis of:
Giorgio Giardini,
Matricola 874841

Academic Year 2017-2018

Ringraziamenti

Arrivato al termine di questo percorso universitario, vorrei ringraziare tutte le persone che mi hanno sostenuto, creduto in me, e permesso di portare a termine questo grande obiettivo.

Un ringraziamento particolare va prima di tutto ai professori che mi hanno seguito nell'ultimo anno in questo lavoro di tesi, Matteo Cesana ed Alessandro Redondi, per avermi fatto appassionare al mondo IoT e permesso di lavorare nel loro laboratorio su questo progetto.

Vorrei poi ringraziare tutti i membri dell'ANTLab per avermi supportato e aiutato in tutte le fasi del lavoro di tesi. In particolare Edoardo, che mi ha seguito più da vicino, ed è stato sempre disponibile a ragionare insieme su ogni problema, fornendomi idee e consigli utilissimi per l'ottenimento del risultato finale.

Un altro ringraziamento va poi inevitabilmente a tutte le persone e gli amici che hanno trascorso al mio fianco questi 5 anni universitari, permettendomi di affiancare i miei studi a incredibili viaggi, escursioni estreme in montagna e momenti di svago. In particolare ringrazio Valentina, conosciuta proprio in università e con la quale ho condiviso i più grandi viaggi e le più grandi gioie di questi ultimi anni. Un grazie anche a tutte le amicizie di vecchia data, a chi dai banchi delle scuole superiori mi è ancora vicino, mi ha aiutato a crescere, coltivare le mie passioni e diventare la persona che sono. Sia chi ancora è qui, che chi per un motivo o per l'altro si è dovuto allontanare, resterà per sempre parte fondamentale e integrante della mia vita.

Un grazie anche a tutto il team Dynamis, con una nota particolare ai ragazzi del reparto elettronica, che mi hanno permesso di affiancare agli studi la mia grande passione per la guida e il mondo delle corse.

Infine, è inevitabile ringraziare tutti coloro che mi hanno permesso, non solo da un punto di vista economico, di portare a termine questo percorso universitario. Un grazie di cuore ai miei genitori, Angelo e Roberta, con i quali sono cresciuto e ho condiviso ogni gioia e delusione della mia vita, ricevendo sempre supporto nelle mie scelte e aiuto nei momenti di difficoltà. Un grazie va anche a tutti gli altri ingegneri della famiglia, il nonno Rino e lo zio Andrea, alla mia nonna Giovanna e a chi invece porta avanti il lato più artistico e culturale della famiglia, ovvero mio fratello Filippo e mia zia Bruna. Un pensiero va anche a mio nonno Aldo, che sarebbe stato senza dubbio fiero di vedermi terminare questo percorso.

A tutti voi dedico questo lavoro e questo traguardo.

Abstract

Think about knowing how busy your favourite bar is before getting dressed and leaving home.. Think about a smart house, which automatically controls heating and lighting, and at the same time saves energy when nobody is at home.. Think about a university, in which you always find a place in class because the assignment of the classrooms is based on previous occupations data.. It could certainly be useful.

Internet of Things (IoT) and Machine Learning (ML) are nowadays gaining more and more importance in computer science world. The aim of this project is to use these two disciplines in order to find a robust model able to predict occupancy status of an indoor space. Real-time measurements on the occupancy status of a room can be exploited in many scenarios (HVAC and lighting system control, building energy optimization, allocation and reservation of spaces, etc.). Traditional techniques involve the use of cameras combined with video analysis. To avoid problems of privacy violation in public places it is necessary to use less invasive techniques, such as measurement of the concentration of carbon dioxide (Co2) in the air, that has been proved to be very related to the amount of people who are breathing in an indoor space. An economic network of precise sensors has been designed and developed for data collection and machine learning studies were performed, obtaining excellent prediction results in two different scenarios. The greatest contribution to these results is mainly due to the innovative technique of using more than one Co2 sensor, positioned in strategic places and communicating in a local network, and to a first phase of prediction regarding the opening state of windows. If combined with other techniques, such as other environmental sensors or wireless packets sniffers, we believe that an even more precise estimate can be reached, in order to achieve the higher goal of a smart campus development.

Sommario

Pensa di poter conoscere quanto è affollato il tuo bar preferito prima di vestirti e uscire di casa.. Pensa a una casa intelligente, in grado di controllare automaticamente il riscaldamento e l'illuminazione, risparmiando energia consumata inutilmente quando nessuno è a casa.. Pensa a un'università, nella quale trovi sempre posto a lezione perchè l'assegnamento delle aule si basa su dati storici di occupazione.. Sarebbe certamente utile.

Internet of Things (IoT) e Machine Learning (ML) sono due discipline che al giorno d'oggi stanno guadagnando sempre più importanza nel mondo dell'informatica. L'obiettivo di questo progetto è quello di unire queste due discipline con lo scopo di creare un modello in grado di fornire previsioni robuste riguardo lo stato di occupazione di un ambiente chiuso. Misurazioni in tempo reale riguardo lo stato di occupazione di una stanza possono essere sfruttate in diversi scenari (sistemi HVAC, sistemi di controllo illuminazione, ottimizzazione energetica di edifici, allocazione di spazi, ecc..). Le tecniche tradizionali prevedono l'utilizzo di telecamere e analisi dei video prodotti. Tuttavia, per evitare problemi di violazione della privacy dei presenti in luoghi pubblici, è necessario utilizzare tecniche meno invadenti, come la misurazione della concentrazione di anidride carbonica (Co2) nell'aria, che si è dimostrata essere molto legata alla quantità di persone che stanno respirando nell'ambiente chiuso. Una rete economica di sensori precisi è stata progettata e sviluppata per raccogliere dati, e sono stati fatti studi di machine learning, ottenendo eccellenti risultati di predizione in due diversi scenari. Il maggior contributo a questi risultati è stato principalmente dato dalla tecnica innovativa di utilizzare più di un singolo sensore, posizionati in posti strategici e comunicanti in una rete locale, e da una prima fase di predizione riguardante lo stato di apertura delle finestre. Se combinato con altre tecniche, come l'utilizzo di altri sensori ambientali o sniffer di pacchetti wireless, crediamo che una stima ancora più precisa possa essere raggiunta, con l'obiettivo più grande di sviluppare un campus universitario intelligente.

Contents

1	Introduction	1
1.1	Overview	1
1.2	Project Objective	2
1.3	Thesis Outline	3
2	State of the art	5
2.1	Image-based detection	5
2.2	Detection and radio-based sensors	6
2.3	Sniffing network packets	7
2.4	Environmental sensors	7
2.5	Co2-based prediction	9
2.5.1	Physically modeling human emissions	9
2.5.2	Machine learning methods	11
2.6	Considerations and common problems	12
3	IoT system implementation	15
3.1	Theoretical background	15
3.1.1	Co2 sensor	15
3.1.2	Microcontrollers	16
3.1.3	MQTT protocol	19
3.1.4	Node-Red	20
3.1.5	MySQL database	20
3.2	System implementation	21
3.2.1	Sensor Side	22
3.2.2	Server Side	26
4	Machine learning study	33
4.1	Machine learning overview	33
4.2	Problem definition	35
4.3	Implementation	36
4.3.1	Data read	36
4.3.2	Sensors aggregation	38
4.3.3	Resample dataset	38
4.3.4	Filter series	39
4.3.5	Compute features	40
4.3.6	Time lag	42
4.3.7	Dataset fusion	42

4.3.8	Filter outliers	43
4.3.9	Two step estimate	43
4.3.10	Cross-validation	45
4.3.11	Feature evaluation and selection	46
4.3.12	Model estimation	48
4.3.13	Prediction	53
4.3.14	Evaluation metrics	53
5	Experiments and results	57
5.1	Scenarios	57
5.1.1	AntLAB small	57
5.1.2	AntLAB big	58
5.2	Results	60
5.2.1	Datasets description and pre-process baseline	60
5.2.2	Baseline model	64
5.2.3	Multi-sensor variation	66
5.2.4	Multi-feature variation	67
5.2.5	Model variation	70
5.2.6	Windows	72
5.2.7	Time lag	74
5.2.8	Resample and filter	75
5.2.9	Learning curves	76
5.2.10	Binary case	77
5.2.11	Mixing scenarios	78
6	Conclusions and future works	81
	Bibliography	83

List of Tables

2.1	Multi-sensor occupancy detection research	8
2.2	Carbon dioxide sensor-based occupancy detection research	11
3.1	K30 sensor properties	17
5.1	Scenarios properties	57
5.2	Datasets properties	61
5.3	Baseline models results	66
5.4	Multi-sensors regression models results	67
5.5	AntLAB small occupancy feature evaluation	67
5.6	AntLAB big occupancy feature evaluation	67
5.7	Multi-feature regression models results	68
5.8	Lasso models results	70
5.9	Random Forest Regressor models results	72
5.10	Models results in AntLAB small considering different methods for windows prediction	73
5.11	Models results in AntLAB big considering different methods for windows prediction	73
5.12	Windows prediction evaluation on test data	73
5.13	RMSE evaluation with change of resample frequency	76
5.14	RMSE evaluation with change of filter normalized cutoff frequency	76
5.15	Binary prediction evaluation on test data	78
5.16	Models results in a mixed scenario	79

List of Figures

2.1	Simplified scheme of gas exchange in a room	10
2.2	Theoretical Co2 concentrations during 8-hours sleeping of a person in a 21m ³ room, with different flow rates	11
3.1	schematic of an NDIR sensor	15
3.2	Co2 meter K30 sensor	16
3.3	Arduino Uno board	17
3.4	ESP8266 Wi-Fi module	18
3.5	Raspberry Pi 3	18
3.6	MQTT protocol structure	19
3.7	Node-Red development tool	20
3.8	General hardware schema	21
3.9	Sensor side wireless configuration	22
3.10	Serial connection between Arduino and K30	23
3.11	Esp8266 firmware update mode	24
3.12	Arduino software	24
3.13	Sensor side cable configuration	25
3.14	SmartGate used for ground truth computing	26
3.15	Raspberry Pi block diagram	26
3.16	Netgear WNA3100 Wi-Fi USB adapter	27
3.17	Node-Red application code	28
3.18	Node-Red GUI screenshot	30
3.19	Node-Red mobile GUI screenshot	30
3.20	PhpMyAdmin database screenshot	31
4.1	Real-time prediction scenario	35
4.2	ML process definition	36
4.3	ML process execution flow	37
4.4	Bode plot of a first-order Butterworth filter	39
4.5	Example of second order Butterworth on Co2 data	40
4.6	Example of LBF on a generic acquisition	43
4.7	Two step estimate general schema	44
4.8	K-fold cross-validation	45
4.9	Learning curve computation	46
4.10	Ideal learning curve	47
4.11	Validation procedure schema	48
4.12	Logistic Regression with a single input feature	49
4.13	Example of a generic Decision Tree	50

4.14	Example of a generic polynomial regression	52
4.15	Binary confusion matrix	54
5.1	AntLAB small map	58
5.2	Ground truth mobile device and sensor powered from battery . . .	59
5.3	Video camera used and an example of photo taken through it . . .	59
5.4	AntLAB big map	60
5.5	Example of aggregation and in-out feature computation	61
5.6	Frequency response of a second order Butterworth filter with normalized cutoff frequency 0.30	62
5.7	Example of Butterworth filter application	62
5.8	Co2 samples distribution (on the left AntSmall, on the right AntBig)	63
5.9	People samples distribution (on the left AntSmall, on the right AntBig)	63
5.10	Co2-People scatter (on the left AntSmall, on the right AntBig) . . .	63
5.11	Gradient-People scatter (on the left AntSmall, on the right AntBig)	64
5.12	Time-People scatter (on the left AntSmall, on the right AntBig) . .	64
5.13	Baseline discretized model (on the left AntSmall, on the right AntBig)	65
5.14	Baseline model real vs predicted on test data (on the left AntSmall, on the right AntBig)	65
5.15	Baseline model people train, people test, people test predicted temporal series (above AntSmall, under AntBig)	66
5.16	Multi-feature model real vs predicted on test data (on the left AntSmall, on the right AntBig)	68
5.17	Multi-feature model people train, people test, people test predicted temporal series (above AntSmall, under AntBig)	69
5.18	Co2-Gradient regression discretized model (on the left AntSmall, on the right AntBig)	69
5.19	Lasso validation on alpha (on the left AntSmall, on the right AntBig)	70
5.20	Random Forest Regressor validation on max-depth (on the left AntSmall, on the right AntBig)	71
5.21	Co2-Gradient Random Forest Regressor model for AntSmall (on the left max-depth = 3, on the right max-depth = 5)	72
5.22	Windows logistic regression model (on the left AntSmall, on the right AntBig)	74
5.23	Windows classifier tree model (on the left AntSmall, on the right AntBig)	74
5.24	Nrmse vs time-lag with a sample every 2 minutes (on the left AntSmall, on the right AntBig)	75
5.25	Nrmse vs time-lag with a sample every 30 seconds (on the left AntSmall, on the right AntBig)	75
5.26	RMSE train and test learning curve for optimal models (on the left AntSmall, on the right AntBig)	76
5.27	Co2-BinaryState scatter (on the left AntSmall, on the right AntBig)	77
5.28	Time-BinaryState scatter (on the left AntSmall, on the right AntBig)	77
5.29	Gradient-BinaryState scatter (on the left AntSmall, on the right AntBig)	78

5.30	Confusion matrix for binary prediction (on the left AntSmall, on the right AntBig)	78
5.31	Mixed scenario real vs predicted on test data (on the left AntSmall, on the right AntBig)	79

Chapter 1

Introduction

1.1 Overview

The *Internet of Things (IoT)* is a new paradigm that has become very popular in the last few years. It provides the existence of smart electronic objects, which are capable to communicate their information to other objects or central units, through a wireless internet network. Smart objects can be sensors, computers or embedded systems with the capability to connect and be part of the same network. IoT application are both in everyday problems and activities (such as healthcare, transportation, houseworks, etc.), with the aim of making our lives more comfortable, and in industrial contexts (such as manufacturing, agriculture, energy management, etc.), with the aim of automating or simply monitoring some fundamental processes. Some studies show that the number of objects connected to a network nowadays is growing strongly: if in 2017 they were “only” 8.4 billions, it is expected that 30 billions will be reached by 2020 [34].

Another important area of modern computer science is *Machine Learning (ML)*. It is a subset of artificial intelligence which studies algorithms by which a machine can learn and create a mathematical model simply by observing a large amount of collected data. This is commonly used in cases in which someone wants to predict the status of some variables or make decisions, without knowing anything about the laws that controls a physical phenomena, or the criteria with which to make the decisions themselves. Machine learning is becoming more popular every day and its fields of application are becoming even larger. They range from computer vision, to weather forecasts, to predictions of simple variables, such as the number of people in a room. In 2019 it became almost mandatory for a large company to have a department dedicated only to this specific practice.

The most spontaneous union of these two disciplines, *Internet of Things* and *Machine Learning*, is divided into the following steps:

- An IoT wireless sensor network is used to collect a large amount of observations about one or more interesting variables and all data are stored in a central device.
- Data are than processed by a machine learning algorithm which crates a model for the prediction of one or more of these variables, based on the value of the others.

- The model is implemented in the central unit, which is reprogrammed for predicting and displaying the interested status, based on real time values coming from sensors.

This project shows a specific application of this process, focusing in detail on the development of the first two points.

1.2 Project Objective

Knowing the occupation status of an indoor space can be nowadays a crucial information. It can help both a single person, for example in reducing the loss of time per day, or a large company, for example in saving energy spent unnecessarily. Here some examples. Let's suppose you are a student in an exam session. During the session the study halls are full of people and there is the risk of not finding a place to sit down. Campus is big enough and you want to be sure to choose a room with some free places before leaving home. It would be very useful for you to know how many seats are available simply by consulting a web page or writing to a chatbot. This could often save you a lot of time. Let's now suppose that you are in charge of scheduling heating, ventilation and air conditioning (HVAC) systems of university buildings. You know that this systems account for approximately half of the energy consumed in buildings in developed countries, and about 20% of total consumption in the USA [31]. It is therefore essential to design and operate HVAC systems in an energy-efficient manner to meet low-energy targets and save unnecessary consumption [9, 25, 35]. Traditional system are based on fixed scheduling and correction through real-time temperature sensing. Suppose now that a lesson was unexpectedly deleted from the traditional schedule. It would be very useful to know that no one is in the classroom and automatically turn off air-conditioning system even if it is very hot. The same applies to lighting control [23] A third area in which people counting can be useful is the study of spaces and rooms utilization. Collecting data about room occupancy at a given time, a building manager can analyse which rooms are under-utilized or over-utilized, adjusting room allocation and utilization accordingly. Other possible application are for security purposes in big environments, like for example accelerating safe evacuation [36].

The objective of this project is therefore to find a robust method for indoor occupancy detection, without using invasive methods that violate the privacy of those present. In particular, have been used data about Co2 concentration in the air, proved to be very related to the number of people in the room [6, 26, 10, 18, 26]. Moreover, Co2 sensors are often installed in modern Building Management System (BMS) for other purposes, and can be exploited and integrated into a predictive system. An IoT multisensor system has been implemented for data collection, and machine learning techniques have been used, in order to make it possible to arrive at this estimate without knowing the physical model of people's Co2 emissions. Results have been evaluated and discussed.

This study finds its place within the smart campus project, a program with the main purpose of improve students' experience at the university campus, providing them with some IT facilities as, for example, information about the occupation

state of a classroom. Smart campus project also aim to automate some fundamental processes in university buildings, such as heating or air conditioning, things that can be facilitated by the knowledge of the exact real-time occupation of classrooms and common areas [33, 30, 4].

1.3 Thesis Outline

This section summarizes the layout and contents of the thesis chapters.

Chapter 2 analyzes in detail Co2 people emission and analyzes the feasibility of using it to count people in closed environments. Then it describes the state of the art for indoor and outdoor occupancy detection methods. Related works are classified in term of IoT technologies, used prediction features, machine learning methods and obtained results.

Chapter 3 describe the hardware part of the project, starting from some theoretical considerations and ending with a detailed description of this specific implementation of the IoT architecture. It talks specifically of: how a carbon dioxide sensor works, how hardware platforms (such as Arduino, RaspberryPi, Esp8266) can be exploited in IoT applications, communication protocols such as MQTT, visual programming with Node-RED and management of a MySQL database. Motivations regarding the choice of all technologies used, and schemes about our specific implementation, are also reported.

Chapter 4 describe the prediction part of the project and its evaluation methods. Also in this case the chapter describes at first some machine learning techniques from a generic point of view, and than describes in detail how data have been pre-processed and used to get a consistent model. This chapter also talk about model validation and testing methods.

Chapter 5 is the part related to the experiments actually performed with the methods described above. Two different experiment scenarios are described, together with the related methodology of ground truth calculation. For each scenario performance are evaluated, and followed by some graphs that helps in results explanation.

Chapter 6 is the conclusive part of the project. Our initial goals and their achievement are discussed, leaving then space for some considerations about possible future works.

Chapter 2

State of the art

There are a large number of methods used in literature for people counting purposes, both in indoor and outdoor environment. This problem is quite complex due to many variables that must be considered, like users privacy, level of accuracy required and system cost. This chapter gives an overview for all these methods and provide a wide view on the state of the art for the subject of the entire work. This analysis is than taken as common base for the project, starting from the choice of the method for results evaluation, also providing a good initial feasibility analysis. Several systems have been developed and proposed in literature for this purpose, mainly divided into two categories:

- Image-based
- Data-based

Data-based category can be further divided into these main categories:

- Detection and radio-based sensors
- Sniffing network packets
- Environmental sensors

The chapter continues with a more specific analysis regarding the use of Co2 and ends giving an idea of what are the most commonly found problems.

2.1 Image-based detection

All methods that make use of video cameras technology belong to this category. Cameras are used like sensors, able to capture information from frames through image analysis techniques. The starting point of every human detection system is the Viola-Jones one, with the Haar feature based approach [37]. This technique involves the search for Haar-like features in the image and the subsequent processing of these features in a multi-staged cascade classifier. It was one of the first algorithms for real-time human detection and some implementations are available in the OpenCV library like for example “Full Body Detection”, “Upper Body Detection” and “Lower Body Detection”. Another important early approach is based

on Histograms of Oriented Gradients (HOG), proposed in 2005 by N. Dalal and B. Triggs [14]. These methods, although less precise than modern ones, are highly used in industry because they require little computing power and they are readily available in computer vision libraries such as OpenCV [1]. Modern methods are mainly based on deep Convolution Neural Networks (CNN). In general they consist in training personal CNN able to recognize and localize people inside images. If primitive methods included the use of already trained models, CNN needs to be trained with a large number of samples, in order to significantly improve the performance. They often require GPU acceleration to provide comparable frame-rates to earlier approaches. However, these methods are far more robust and their accuracy is close to perfection [2]. A strong point of image-based techniques is certainly the high level of precision obtained in people counting. For example Zhang [41] utilized the depth-frame data from a Kinect camera to detect people with a precision of 99.7%. Petersen [32] applied the same approach reaching an accuracy of 99%. However, installing cameras, can be perceived as a privacy violation and often represents an additional investment and running cost to a building project. Other problems can derive from the impossibility of counting people occluded by some objects, or standing behind other people.

2.2 Detection and radio-based sensors

A second group of techniques is the one that makes use of detection sensors and radio-based sensors. Currently, the most commonly used sensor for occupancy detection is Passive Infrared Sensors (PIR), generally installed in buildings for an automatic management of lighting or for security purposes. However, relying solely on PIR sensor, is rather uncertain since sensors do not capture immobile occupants. For this reason PIR are much more used to detect people passing through a door and the direction of their movement, compared to the absolute count of people. These methods are generally based on a couple of sensors [38]. A different system, able to reach 95% of accuracy is explained in [20]. An array of 8 digital PIRs is positioned up in the center of the door, parallel to the moving direction, in order to detect passage and establish movement direction of people. Other works, like [22], use a large PIRs sensor network with the aim of monitoring people movements in a large building. Radio-based devices instead include WiFi, Bluetooth, and any electromagnetic waves and gamma rays. Radio measurements are also very used as data for occupancy estimation, managing to distinguish the state of wireless signal between an empty environment and an occupied one. For example, the work in [15], uses a couple of transmitter and receiver devices to assess the impact of a certain number of people on the signal strength indicator at the receiver (RSS), blocking the line of sight. Authors developed a model for the probability distribution of the received signal amplitude as a function of the total number of occupants and use that as estimation methods in indoor spaces with a maximum of 9 people, having an average error below 2 for 95% of the times. A low-power pulsed radar was utilised for people counting in [21]. The model, based on support Vector Regression (SVR), was able to find a correlation coefficient of 0.97 and a Mean Absolute Error (MSE) of 2.17 between ground truth and estimations up to

40 occupants. Beside being very cheap, such systems have the benefit of being able to work both in indoor and outdoor spaces. Sometimes detection and radio-based sensors are integrated with other different type of sensors, in a more complex model, trying to further increase accuracy.

2.3 Sniffing network packets

If the radio-based methods previously treated were free from assumptions about devices owned by people, there are methods based on sniffing packets sent on the network just by personal devices. These methods are mainly for people tracking purposes in indoor spaces, where GPS is not appropriate. In fact indoor environment requires finer granularity and precision of that offered by the GPS (5-10m). The GPS signal is also often not present inside buildings. Considering indoor tracking, Di Domenico et al. [16] were the first investigating on possibilities to estimate crowd in an indoor environment. In their work they analyzed LTE signals, with three receivers placed in different positions, with a maximum accuracy of 92%. Another work by Abbott-Jard et al. [3] describes an interesting method for tracking vehicle using Bluetooth Media access control Scanner (BMS) and Wi-Fi Media access control Scanners (WMS), doing real surveys along an arterial corridor in Brisbane. WLAN signal is instead used by Handte [19] in order to track people movements on public transports, to improve their organization and transport experience. His accuracy was able to reach around 49%. Other works extend the sniffing of signals in order to estimate number of people present in an area within a certain radius. In particular they propose machine learning methods based on the number and characteristics of sniffed Bluetooth and Wi-Fi probe requests [39, 30]. A common problem of all these approaches is that are all based on the assumption that every person has one and only one device with Wi-Fi or Bluetooth turned on, while in reality people often turn off connections devices to save battery. Also privacy problem must be considered.

2.4 Environmental sensors

Some techniques that mix the previously treated methods have often been used. For example Zhao et al. [42] obtained convincing occupancy detection results in offices using a Bayesian Belief Network (BBN) together with information from Wi-Fi, GPS location, chair sensor, and keyboard and mouse sensors. However, some occupants may still consider these sensor data to be intrusive. The only completely non-intrusive approach ever adopted is the use of environmental sensors. They include variables such as temperature, humidity, noise and Co2. Lot of works, summarized in table 2.1, focus their attention on this kind of data, mixing different type of sensors and different machine learning methods. The Building Level Energy Management Systems (BLEMS) project from University of Southern California [40], for example, uses a combination of sensors (light, sound, motion, CO2, temperature and humidity sensor), each of them connected to an Arduino board which sends data to a central database. They uses Radial Basis Function

(RBF) for the machine learning prediction. An 87.62% accuracy was achieved for self-estimation (train and test in the same room) and when the model was implemented in another room, the cross-estimation result showed a 64.83% of accuracy. Other machine learning algorithms, such as Multi-Layer Perceptron (MLP), Gaussian Processes (GP), Linear Regression (LR), Support Vector Machine (SVM) and Ensemble Voting (EV) were implemented with an accuracy between 46% and 95% [28]. Another work [13], using temperature sensors only, achieved 85% of accuracy to predict a maximum of two persons in a single room. Their accuracy can be up to 95%-99% adding features such as light, Co2 and humidity data. They uses an Arduino with a ZigBee radio for data communication to a remote storage system and different machine learning algorithms. A further framework was created to produce occupancy estimates at different levels of granularity and provide confidence measures for effective building management in [24]. By using K-Nearest Neighbor (KNN) and SVM, their accuracy reached a maximum of 94.7%. A real-time occupancy detection system by using Decision Trees (DT) with multiple types of data such as light, sound, Co2, motion and computer power sensors was conducted by [18]. The lowest accuracy in detecting the binary presence/absence of a person was received from sound sensors (90.79%) and the highest one from motion sensors (98.44%). Another model based on temperature, humidity, Co2, sound, pressure and illumination sensors was used to carry out the binary prediction concerning the state of the room (occupied/vacant) in [5]. By using feature engineering and various machine learning algorithms such as MLP, GP with RBF, SVM, and Naive Bayes (NB) the human occupancy can be detected above 95%. These are just some of the multitude of jobs regarding occupancy prediction through environmental sensors and machine learning, and are summarized in table 2.1.

Source	Data Type	Algorithm	Max.P	Evaluation
[40]	Light, sound, motion, Co2, Temperature, Humidity	Radial Basis Function	9	87,6%
[13]	Temperature, Light, Co2, Humidity	Random Forest, Gradient Boosting Machines, Decision Tree	2	95%-99%
[24]	PIR, Noise, Temperature, Light, Humidity	K-Nearest Neighbor, Support Vector Machine	20	94.7%
[18]	Co2, Computer current, Light, PIR, Noise	Decision Tree	1	98.4%
[5]	Temperature, Humidity, Co2, Noise, Pressure, Light	Multi-Layer Perceptron, Naive Bayes, Gaussian Process with Radial Basis Function, Support Vector Machine	Binary Occupancy	95%

Table 2.1: Multi-sensor occupancy detection research

2.5 Co2-based prediction

Since Co2 sensors are often already integrated with ventilation infrastructure in buildings, in this work we focus on utilising only Co2 sensor data. Lot of works have identified Co2 to be one of the most correlated feature to estimate the number of indoor human occupants and that it is sufficient to reach an acceptable level of precision. Consequently, operational cost can be reduced by not purchasing and installing extra sensors such as PIR or motion.

Two approaches can be used to create a reasonable model:

- Physically modeling human emissions
- Machine learning methods

2.5.1 Physically modeling human emissions

Physically modeling human emissions means exploiting knowledge about the composition of the air and about the human metabolic system in order to create a correct expiration model. It is known that persons exhales Co2 as the natural process of breathing, but unfortunately the amount varies from person to person, depending on multiple variables such as height and weight. Equation 2.1 gives the exhalation rate of Co2 per person based on these parameters [8].

$$\lambda = \frac{M \times RQ \times \sqrt{H \times W}}{21131 \times (0.23 \times RQ + 0.77)} \quad (2.1)$$

$$\begin{aligned} \lambda &= \text{Co2 production rate} \\ M &= \text{metabolic rates (in } W/m^2) \\ RQ &= \text{respiratory quotient} \\ H &= \text{height (in } cm) \\ W &= \text{weight (in } kg) \end{aligned}$$

In particular this relationship is non-linear in person height and weight and it must be considered in our estimation. However having a complete model of the exhalation flow rate of a single person is not sufficient to determine the relationship with the absolute concentration of Co2 in the room. Other equally important aspects in creating a good physical model are in fact the characteristic variables of the room itself. In particular, a simplistic study [17], modeled the state of an indoor space at time t as function of its volume and its ventilation rate (fig. 2.1). Volume is important due to the fact that, as the size of a space changes, the emissions of a single person have a different impact in terms of absolute carbon dioxide concentration. Ventilation instead is important due to the fact that if indoor air is mixed with clean outdoor air the absolute internal Co2 concentration inevitably changes.

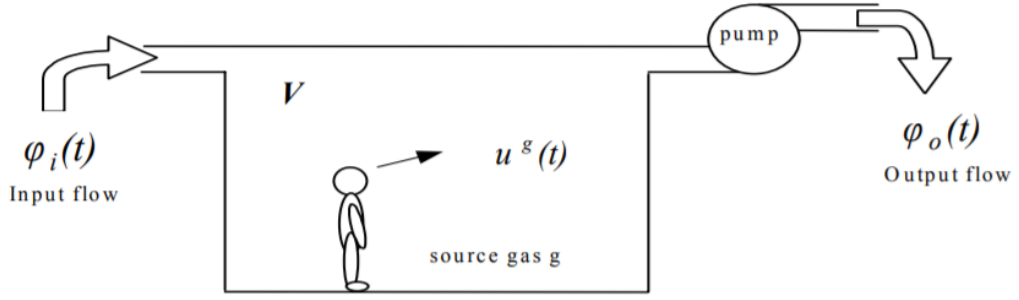


Figure 2.1: Simplified scheme of gas exchange in a room

Going to analyze more in-depth studies, like the one in article [11], this relationship is better explained. In particular equation 2.2 shows the relationship between the variation of total internal Co2 mass (proportional to the concentration through room volume), Co2 production rate and characteristic variables of the room, valid for values of m found in everyday reality.

$$\frac{dm(t)}{dt} = \lambda + \left(c_0 \rho - \frac{1}{V} m(t) \right) Q \quad (2.2)$$

- $m(t)$ = total Co2 mass at time t
- λ = Co2 production rate
- c_0 = Co2 concentration in ventilation flow
- Q = ventilation rate
- V = room volume

When the ventilation rate is null, the variation is simply directly proportional to the emission rate. When the ventilation rate is not null and someone is exhaling in the room, instead, the variation will be higher for low values of m , to become completely stationary when m is high enough. When no one is present in the room, finally, the internal mass will decrease until reaching a steady state, in which the ventilation flow will be balanced. Figure 2.2 shows Co2 concentration curves of a 21m^3 space during a 8 hours night, with one person sleeping, with different ventilation rates. Analyzing all these information we can clearly understand how for creating an accurate physical model we must discover many unknown parameters and make strong assumptions. Furthermore we note that, due to the dependence on the room volume and its air flow rate, the model inevitably changes from an ambient to another. However, this analysis leads to interesting considerations that can be exploited in the construction of a machine learning model, such as the tendency of the concentration curve to stabilize due to the presence of external air flows, and the possible correlation of the number of people not only with the absolute quantity of carbon dioxide, but also with its variation.

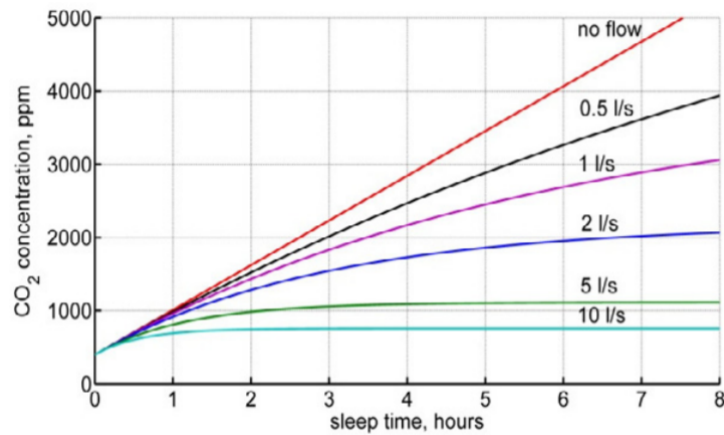


Figure 2.2: Theoretical Co₂ concentrations during 8-hours sleeping of a person in a 21m³ room, with different flow rates

2.5.2 Machine learning methods

Due to the impossibility of knowing all physical parameters, several studies in the literature have used machine learning techniques. Machine learning allows you to create accurate models simply by analyzing previously collected data, even without any physical knowledge of the subject. The main disadvantage is the need for a data collection and model train phase, which can be dependent from the specific prediction ambient. Table 2.2 presents a detailed analysis of related works on indoor occupancy detection using machine learning and Co₂ data. Sensors and technologies used are also analyzed here.

Source	System	Sensor	Algorithm	Max.P.	Evaluation
[6]	Cloud upload + MATLAB, WEKA, R	Netatmo Urban Weather Station	CD-HOC (Serial Decomposition + Regression)	4-300	Accuracy 94%-73%
[26]	Local storage + WEKA	K30	Sensing by proxy	7	RMSE 0.6311
[10]	Download data from BACNet server	unknown	PerCSS + Ensemble Least Square Regression	42	NMSE 0.075
[18]	Local storage + KNIME	K30	Decision Tree	1	Accuracy 94.7%
[26]	Co ₂ sensor network + central database	unknown	Hidden Markov Models, Neural Network, Support Vector Machines	4	Accuracy 65%-80%

Table 2.2: Carbon dioxide sensor-based occupancy detection research

A possible approach is the one followed in [6]. They use a Netatmo Urban Weather Station, upload data to cloud for integration purposes, and then utilize Weka,

Matlab and R to perform experiments. They use Seasonal-Trend Decomposition (STD) and Seasonal-Trend decomposition based on Loess (STL) with a different ML method for each singular component obtained from decomposition. They test the system in two different scenarios, a room (maximum 4 people) and a cinema (maximum 300 people), obtaining respectively 94% and 73% of accuracy. In the cinema case the accuracy is intended considering correct an estimate with a gap of ± 10 compared to ground truth. Same authors also carried out a study about how to extend a model trained in a specific room to make predictions for other different rooms, without the need of retrain the entire model [7]. Research in [26] argues the validity of the sensing by proxy method, a “sensing paradigm which infers latent factors by ‘proxy’ measurements based on constitutive models that exploit the spatial and physical features in the system”. Data are collected locally with a K30 Co2 sensor, and then sent wireless to a remote database for analysis. Sensing by proxy turned out to be better compared to other classic machine learning techniques in 3 selected scenarios, reaching a Root Mean Square Error (RMSE) of 0.6311 for occupation up to 7 people in the same room. An interesting data pre-processing method is proposed in [10]. They present the PerCSS method, an algorithm that “uses task-driven Sparse Non-negative Matrix Factorization (SNMF) to learn a non-negative low-dimensional representation of the Co2 data in the pre-processing stage”. Then they use an Ensemble Least Square Regression (ELSR) based on this denoised data to learn a model. Denoising data increased baseline performance. In an experiment conducted downloading data directly from a building management BACNet server, they reach a Normalised Mean Square Error (NMSE) of 0.075, in a classroom of capacity 42, and an accuracy of 91% and 15% for exact occupancy estimation, when the room was respectively unoccupied and occupied. Machine learning algorithms such as Hidden Markov Models (HMM), Neural Network (NN) and Support Vector Machine (SVM) were implemented in [26], learning to predict an occupancy of maximum 4 people with an accuracy between 65% and 80%. HMM seemed to be the most precise method. From information gain analysis, among the possible features concerning the carbon dioxide, the most correlated to number of people seemed to be the sample-by-sample variation of Co2. Other studies have been performed to identify the presence of a single person in his usual work station [18]. With Decision Tree (DT) and using only Co2 data an accuracy of 94,6% has been achieved with, also in this case, a K30 sensor.

2.6 Considerations and common problems

After an in-depth analysis of the literature, we can deduce the following considerations:

- In human detection without cameras, PIR is the most used sensor. However, although very cheap, PIR is not suitable for counting many people.
- Each research uses different prediction methods, either machine learning based or not, but often the evaluation metrics are different or even accuracy is not provided. This makes difficult a direct comparison between each technique.

- Co2 is one of the best ambient variable for human occupancy detection in an indoor space. Lots of Co2 related feature can be considered for prediction, such as difference between indoor and outdoor concentration or absolute variation respect to previous instants.
- Datasets are generally not publicly available, for a complete research in this area it is necessary to build your own system for data collection.

The goal of this work is to achieve a good occupancy estimation using only carbon dioxide data, developing a reliable system for Co2 level measurement and samples collection, solving the most common problems found in literature in an innovative way and improving the performance achieved by similar studies. The main problem and possible solutions regarding Co2 people counting are listed below.

Latency There is an intrinsic delay between human exhalation and room Co2 increasing. This delay is inevitably present in data, which therefore need a preprocessing phase. An interesting and simple method for managing this delay is proposed in [6].

Uncertainty Co2 level in an indoor space is not only related to human who are breathing in the ambient, but also to the opening and closing of windows and doors. To overcome this problem, a model is proposed that is able to detect and adapt to the eventual windows opening.

Measurement reliability Even with the most accurate Co2 sensor it is difficult to have an accurate measurement of the carbon dioxide concentration in the whole environment, the measure is often affected to the nearness of a person who is exhaling. For this reason, a solution with several sensors in the same environment has been adopted and different methods for data aggregation and filtering were considered and compared.

Precision Most of related works are limited to the binary prediction of the occupation state of a room. Precise people counting is a difficult problem, which is complicated in an exponential way with the increase of the capacity of the room. For this reason we will use as evaluation metrics both Root-Mean-Square Error (RMSE) and an accuracy with tolerance bands. The combined analysis of these two measures allows us to better evaluate the performance of the model.

Scalability Each environment differs from another in terms of size, but above all due to the presence of different air recirculation systems. This does not allow to create a universally valid model, making it necessary to create a new model in every new ambient in which you want to make a prediction. An analysis was made in this work about the possibility of creating a model simply by mixing data from different scenarios. A more complete method has instead been studied in [7].

Chapter 3

IoT system implementation

3.1 Theoretical background

In this section are explained some important theoretical concepts about hardware and software used for the development of the entire IoT system, starting from how a carbon dioxide sensor works, up to the tools used for data acquisition.

3.1.1 Co2 sensor

A Co2 sensor is an instrument able to measure the carbon dioxide concentration in the air. It is measured in percentage Parts Per Million (PPM) and is one of the most important indicator in monitoring air quality. Fresh air, which is breathed outdoors every day, is generally around 400ppm. The most common types of Co2 detectors are Nondispersive Infrared Sensor (NDIR) and chemical gas sensors. NDIR is the most often used type of sensor.

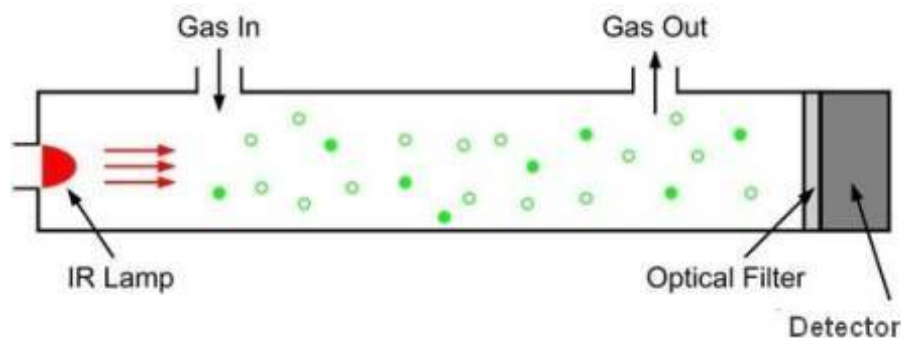


Figure 3.1: schematic of an NDIR sensor

As shown in figure 3.1, they are composed of an infrared (IR) lamp, a tube filled with a sample of air, and an IR light detector. As the IR light passes through the tube, the molecules of Co2 absorb a specific band of the light while letting other wavelengths pass through. The remaining light arrives to the optical filter, able to absorb every wavelength except to the exact wavelength absorbed by Co2. So, the IR detector can read the amount of light that was not absorbed by the

Co₂ molecules or the optical filter. The difference between source and destination amount of light is proportional to the concentration of Co₂ in the air inside the tube. The discovering of the relation between this variable and the exact value in PPM is known as sensor calibration. Calibration is usually done by fixing a conversion curve between output voltage of the sensor and the corresponding PPM value. The best sensitivities reached by NDIR sensors are 20-50ppm [27]. The chemical sensors, on the other hand, are based on sensitive layers made up of polymer or heteropolysiloxane and are more energy efficient. They are generally much smaller than NDIR but have the disadvantage of having a shorter lifetime. They are in fact generally used in microelectronic-based systems.

From the analysis of similar studies (table 2.2), K30 carbon dioxide sensor, by Co2Meter, turned out to be the most reliable and also the most used (fig. 3.2). It is a low cost, infrared, and maintenance-free sensor, integrated on a specific board dedicated to conversion and transmission via UART of a digital signal. An important feature of this type of sensor is the ability of self-calibrating thanks to the built-in self-correcting Automatic Baseline Correction (ABC) algorithm. This algorithm constantly keeps track of the sensors lowest reading over a 7.5 days interval and slowly corrects for any long-term drift detected as compared to the expected fresh air value of 400 ppm. Some other properties relevant to this research are summarized in table 3.1.

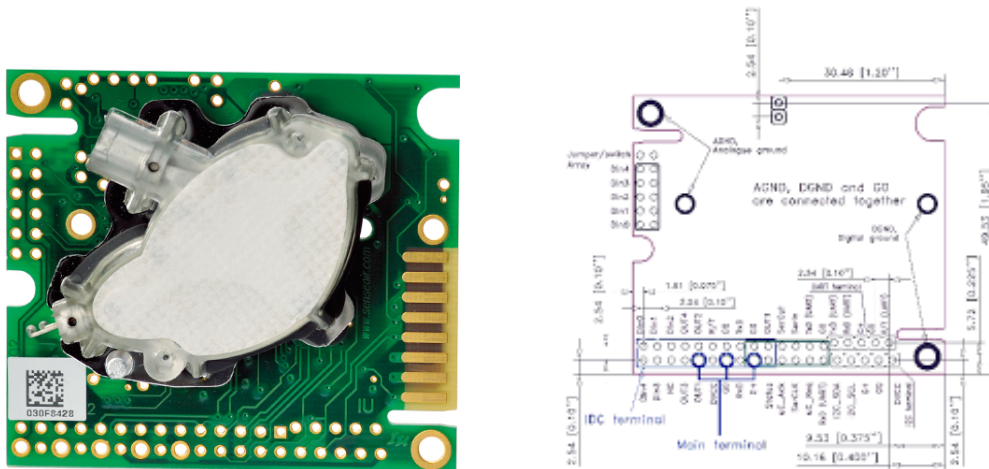


Figure 3.2: Co₂ meter K30 sensor

3.1.2 Microcontrollers

A microcontroller (also called MCU - MicroController Unit) is an electronic device integrated on a chip and used in embedded system. It is actually a programmable complete system that integrates on a single chip a processor, a program, a RAM memory and also some I/O channels (pins). In the following are summarized some of the main features of the electronic devices used for this research.

Power Input	5.5-14VDC, stabilized to within 10%
Current Consumption	40mA average (max 300mA peak on start-up)
Dimension	5.1x5.7x1.4cm (Length x Width x approximate Height)
UART(TxD,RxD)	CMOS, ModBus communication protocol with 3.3V powered logics, 9600 baud rate
Measurement Range	0-10000ppm
Response Rate	2 sec
Sensitivity	$\pm 20\text{ppm} \pm 1\%$ of measured value
Accuracy	$\pm 30\text{ppm} \pm 3\%$ of measured value

Table 3.1: K30 sensor properties

Arduino Uno

Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller (fig. 3.3). It is equipped with 14 digital pins and 6 analog pins to communicate with other boards or other circuits. It can be powered by a USB cable or by an external 9 volt battery and can be programmed via a type B USB cable through a dedicated Integrated Development Environment (IDE) called Arduino IDE. It is probably the most used microcontroller for reading data (both analog and digital) coming from sensors thanks to its simplicity of fast programming. In this basic and cheap version, Arduino does not have an already integrated wireless interface.

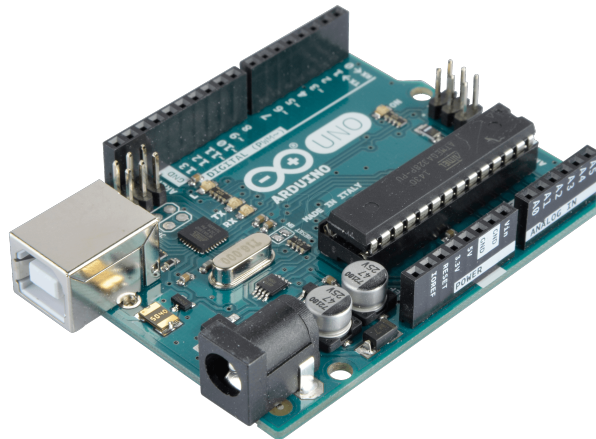


Figure 3.3: Arduino Uno board

ESP8266

ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack, produced by Espressif Systems in Shanghai (fig. 3.4). It has 8 pins that include powering (3.3V) and communication via UART (Tx and Rx pin). Thanks to these pins it

is able to communicate with classical microcontroller through AT commands or dedicated libraries, extending their functionality and allowing them to communicate wireless with other devices. Even if modern microcontrollers come with a Wi-Fi chip integrated, using ESP8266 with a classical Arduino Uno remains the most economical option. To be able to use ESP8266 correctly, however, a phase of installation of the most updated firmware is necessary.

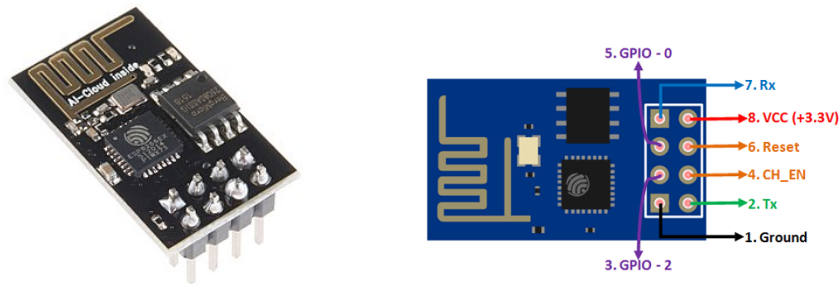


Figure 3.4: ESP8266 Wi-Fi module

Raspberry Pi 3

Raspberry Pi 3 is something more than a simple microcontroller, it is a small computer which hold an operating system, integrated on a single board (fig. 3.5). It is able to host operating systems based on Linux kernel, like for example Raspbian, exclusively designed for this board. In this way it is able to execute most of the programs that can be run on a normal computer, like a web server, a database server or a python script. Raspberry Pi 3 comes with 40 GPIO pins, but also 4 USB 2.0 ports and network interfaces like Ethernet 10/100, WiFi 2.4 GHz and Bluetooth 4.1. It is also equipped with an HDMI out port, which allow you to control it through a locally connected screen, but it can also be controlled remotely, via Secure SHell (SSH) or Virtual Network Computing (VNC). This small-computer is very used in IoT projects as a small central server for processing, saving or simply displaying data.



Figure 3.5: Raspberry Pi 3

3.1.3 MQTT protocol

Message Queue Telemetry Transport (MQTT) protocol is a standard ISO messaging protocol. It is based on the publish-subscribe paradigm and it runs over TCP/IP. It is designed to be light weight, simple and easy to implement. These characteristics make it ideal for use in many situations such as for communication in Machine to Machine (M2M) and IoT contexts, where small code footprint is required, bandwidth is reduced and low power consumption is needed. It is nowadays widely used also by big corporations, like Facebook (with Facebook Messenger) or Microsoft (with Azure IoT).

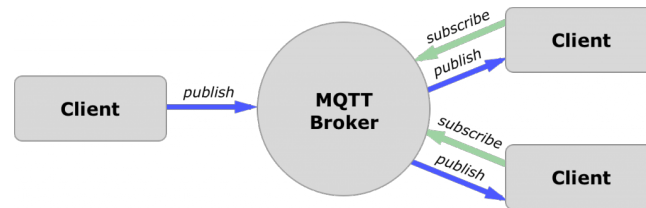


Figure 3.6: MQTT protocol structure

An MQTT system is composed of a server, usually called “broker”, communicating with an indefinite number of clients (fig. 3.6). Each client connected to the broker could be either an information publisher or a subscriber. Information is organized in a hierarchical way in different topics. A topic is uniquely identified by a string, which represents the hierarchy of topics themselves, in a way similar to how folders are identified on a computer. An example of topic could be:

$$\text{Polimi}/\text{ANTLab}/\text{sensor1}/\text{Co2} \quad (3.1)$$

In order to communicate via MQTT a client must first be connected to the broker. A connection phase, in which the client provides the broker with information like his ID or his authentication credentials, is so required. Once connected to the broker each client could either publish data on a specific topic or subscribe to a specific topic. Every time a publisher will post a new message on a topic, the MQTT broker forwards that message to all clients connected and subscribed to that topic. In this way publisher and subscriber could send or receive data ignoring the existence and the locations of other clients.

Another important MQTT feature is the possibility of setting different Quality of Service (QoS) in client connection to the broker. A different level of QoS determines a different security level regarding the delivery of the message to destination. Class 0 is the best effort approach, the message is sent only once without waiting for any acknowledgment from destination. If QoS is set to 1 the publisher continues to transmit the message until it is acknowledged by the broker. Class 2, on the other hand, guarantees not only the delivery of the message to destination, but also that it does not arrive twice.

3.1.4 Node-Red

Node-Red is a development tool for visual programming launched in 2013 (fig. 3.7). In fact, users can develop programs not only by writing textual code, but also by graphically manipulating predefined or customizable program elements (called nodes). It is flow-based and it is designed for wiring together hardware device, APIs and online service, providing fundamental utilities in IoT world. Node-Red runs on Node.js platform and his flow editor is so accessible also remotely via web browser on its predefined port (1880). Each flow is stored locally using JSON format and automatically executed when Node-Red is launched.

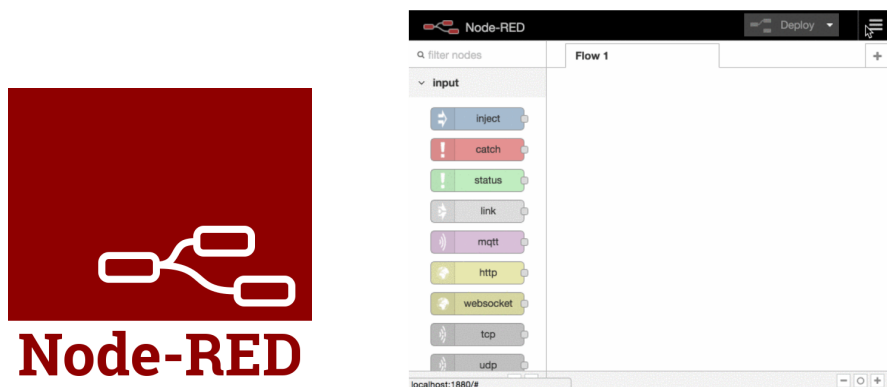


Figure 3.7: Node-Red development tool

Node-Red aims to let users with poor programming background to create IoT application, simply by wiring together function flows, using the wide range of nodes that are present in the palette. There are nodes for every type of service, and new nodes are developed everyday. MQTT nodes, for example, allow users to create an MQTT connection simply by specifying the IP address of the broker. Also Dashboard nodes are widely used, as they allow to create an advanced graphical interface for data visualization and program control in a very simple and intuitive way. Node-Red is very suitable for use on Raspberry Pi, so much so that in the latest versions of Raspbian it is already pre-installed.

3.1.5 MySQL database

MySQL is a Database Management System (DBMS) for relational databases developed by Oracle Corporation from 1995. It is composed by a server and a command-line interface, dedicated to perform data reading and writing operations in relational schemes. It is a cross-platform software and can be installed in all modern operating systems, including Raspbian. The server manages the storage of relational data and once authenticated a user can perform all the classical operations available on a SQL database, such as:

CREATE to add new tables to the relational schema

INSERT to insert a new row in a specific table

SELECT to retrieve some specific data from a table or an aggregation of tables

DUMP to download database in a desired format

There are several methods for accessing a MySQL database server in addition to the command line, both locally and remotely. For example almost all programming languages have their own libraries for the connection to a MySQL server and query performing. Also Node-Red has dedicated nodes for this purpose. Another access option is the usage of PhpMyAdmin. It is an open source graphical administration tool written in PHP and so accessible via web browser. It runs on a web server like Apache and is able to access locally the database, creating an intuitive user interface, accessible both from local and remote, after an authentication phase.

3.2 System implementation

In this section is presented how the IoT system for data acquisition has been implemented, focusing on the main choices and explaining in details how the tools presented in 3.1 have been exploited and interconnected between them.

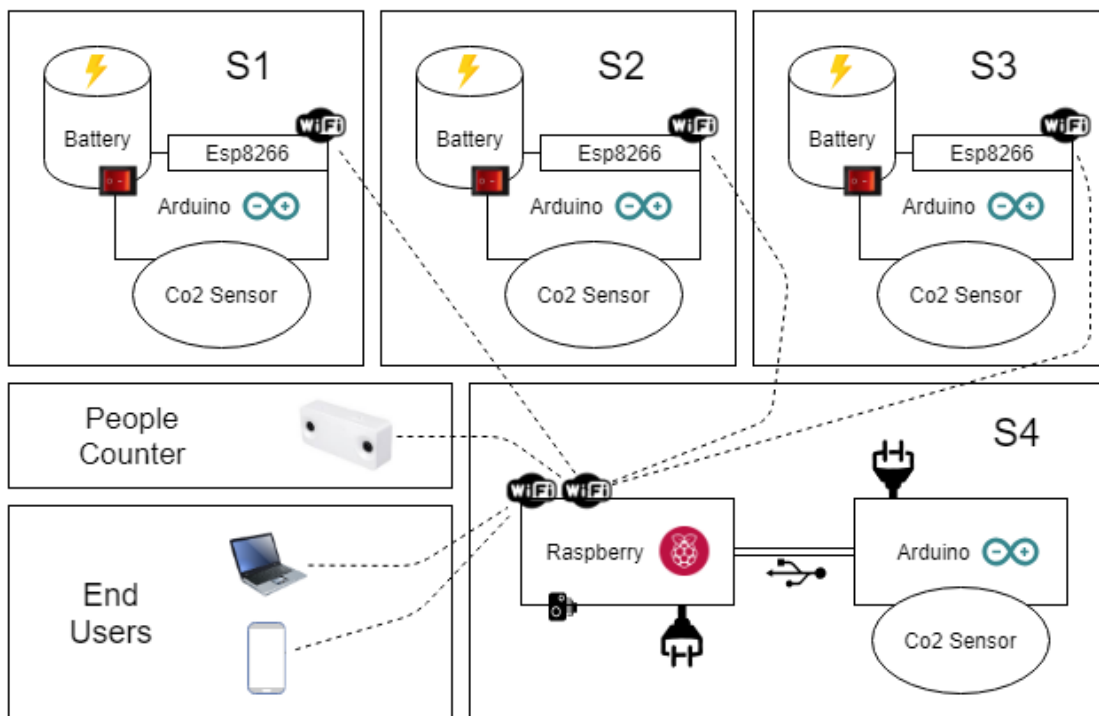


Figure 3.8: General hardware schema

In figure 3.8 is represented the general scheme for the entire system, from sensors to central acquisition unit. It is evident that the initial choice of Wi-Fi as communication method between parts played a fundamental role in the choice of all other utilized technologies. If a mono-sensor system was initially tested with serial data communication, later, the creation of a multi-sensor system with four detectors located in different areas of the room, necessarily entailed the use of wireless

technologies, maintaining communication cable only for one of the four sensors, which is located near the server itself. Another important choice was to use a local network for data collection, in order to be completely independent from internet connection. To better understand the physical structure of the system, this section is divided into two main parts:

1. Sensor-side: this part is dedicated mainly to sensor power supply, circuits implementation for information reading and sending to the server, and related code.
2. Server-side: this part is focused on the creation of the local network and on acquisition software, implemented on Raspberry Pi, exploiting Node-Red and MySQL tools.

All aspects concerning the effective data collection carried out in the various scenarios, and related encountered problems, are then discussed in chapter 5.

3.2.1 Sensor Side

Sensor side part is based on a central Arduino Uno, programmed to periodically read data from sensor and send them to server through the Wi-Fi connection. An overview of the connections is presented in figure 3.9. Three identical groups of this type have been implemented.

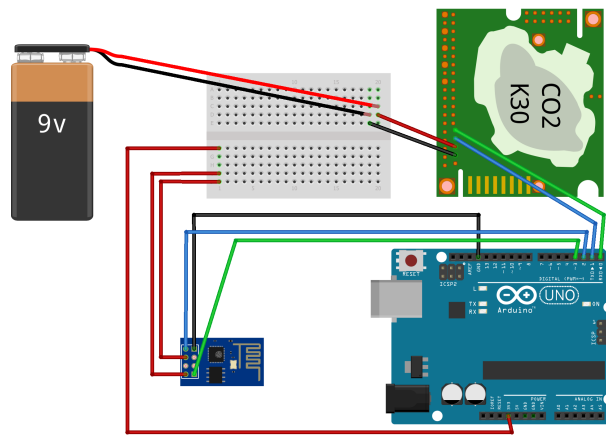


Figure 3.9: Sensor side wireless configuration

Power supply

An Arduino Uno works on a voltage of 5V and an amperage of maximum 800mA. Since K30 sensor, as specified in table 3.1, needs a power input greater than 5 volts, and perform at best at a voltage of 9V, an external power supply is required. K30 sensor current consumption is instead on average 40mA, but it reaches a peak of 300mA during the IR lamp start-up (for a duration of 50ms). Two different type of power supplies have been taken into consideration:

1. Socket power supply: this method is the simplest, because not affected by problems of energy depletion, but obviously it obliges to allocate sensors near current sockets or to pull cables at long distances. A 9 volt power pack, with DC adapter and at least 350mA of amperage (in order to guarantee a correct sensor start-up), can be attached to the 2.1mm Arduino Uno power plug. This voltage will thus be available on the Vin pin, which can be connected directly to the sensor.
2. Battery: this method can be very useful in situations in which current is not available or socket is too distant from sensor location. A standard 9V battery can power the sensor with a direct connection, or with a DC jack adapter connected to the Arduino Uno, as in the case analyzed above. With an average power consumption of 40mA, 800mAh will last about 20 hours of continuous work. The choice regarding the use of this second method obviously depends on the application context.

Sensor to Arduino

As specified in K30 sensor documentation, this kind of sensor is ideal for operating an industry standard UART TXD-RXD connection with Arduino microcontrollers. Two cables were welded to the sensor Tx and Rx pins, and connected respectively to the selected Rx and Tx pins of the Arduino board (fig. 3.10).

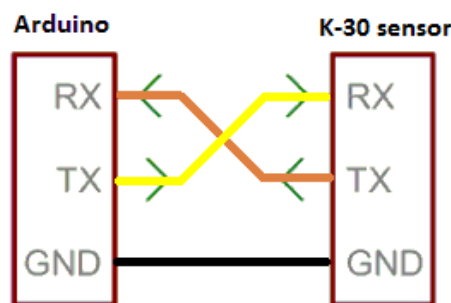


Figure 3.10: Serial connection between Arduino and K30

Arduino to Esp8266

Esp8266 needs a preliminary phase of firmware updating, in order to correctly interface with proper Arduino's libraries. Updating was done through Arduino itself and through a Windows dedicated software. This method exploits Arduino as an adapter from classical USB serial port to UART TXD-RXD connection, needed by Esp8266 for firmware flash. Arduino is also exploited as 3.3V power supply and to connect GPIO-0 to ground, to bring the Esp8266 into firmware update mode (fig. 3.11). Once the firmware is updated, it is possible to communicate to Esp8266 through Arduino IDE serial monitor and AT commands. For example AT+UART_DEF was used to set some parameters fundamental for UART communication, such as the baud rate (number of data transmitted in a second on

the communication channel). The final connection, which brings the Esp8266 into operating mode, is obtained by connecting its Tx and Rx pins respectively to the selected Rx and Tx pins of the Arduino board, both its VCC and its CH_EN to the 3.3V pin, and its Ground to Arduino's GND pin.

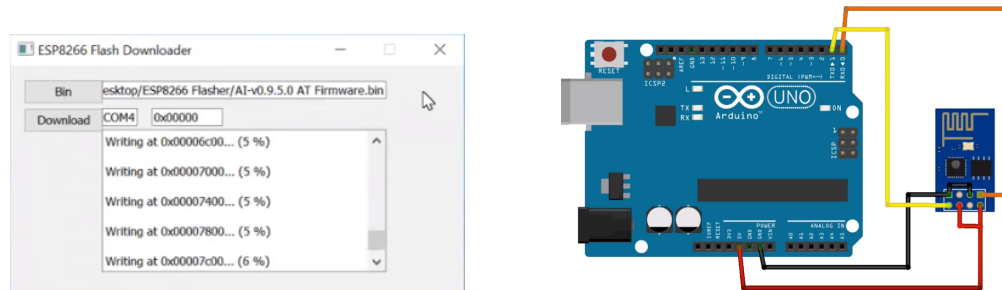


Figure 3.11: Esp8266 firmware update mode

Arduino software

Arduino Uno has been programmed with the dedicated environment Arduino IDE, connecting the microcontroller directly to a windows laptop through a serial USB port. A simplification of the software schematic is in figure 3.12.

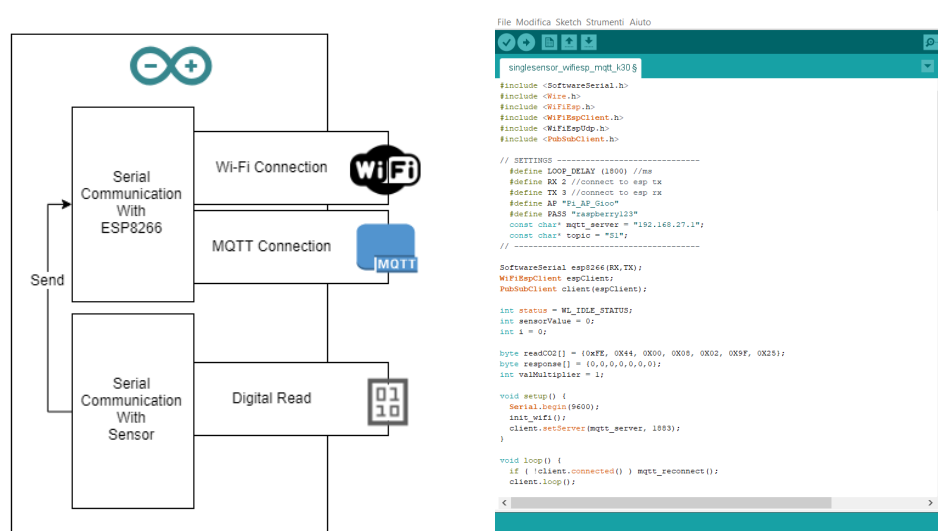


Figure 3.12: Arduino software

When the program is launched two serial communication are initialized, one with the sensor and one with the Esp8266. Since Arduino Uno does not support the management of two serial channels open at the same time on standard digital pins, the Esp8266 communication is started on Tx and Rx Arduino pins. Then a Wi-Fi connection is established to the local network access point and, if successful, is created an overlying MQTT connection to the broker in the network. Command sending to Esp8266, for performing all this operations, is managed by an Arduino

library. When all connections are established program execution jumps to the main loop. A data request is sent periodically to the sensor, by writing a string of bytes in the serial interface. On every request program waits to get a 7 bytes response on the same interface and save bytes on a dedicated buffer. Buffer content is then converted in a decimal number and published via MQTT through Esp8266 on the proper topic. The value in seconds of the loop delay, the pins for serial communication to sensor, the sending topic, the address of the MQTT broker and the SSID and password of the Wi-Fi connection are all configurable parameters in program settings. A re-connection manager has also been implemented, for the proper management of cases in which Wi-Fi or MQTT connection is accidentally lost.

Serial alternative

This variant provides direct connection of the USB interface of Arduino Uno to one of the USB ports of the Raspberry Pi. Data transmission in this way is simpler and more reliable because it doesn't suffer from any problems of lost signal (fig. 3.13). A simpler version of the Arduino software has been implemented for this alternative, that only establish communication with sensor for data retrieval and send them through standard serial interface to the server.

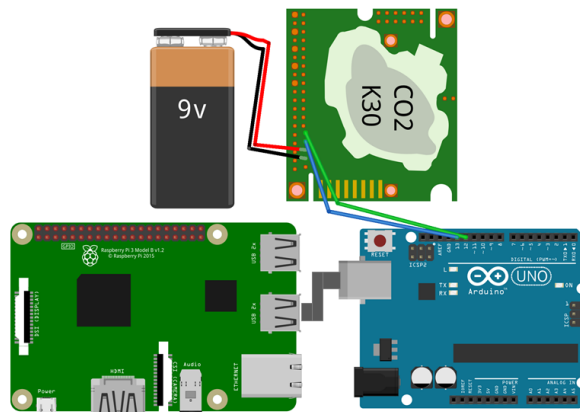


Figure 3.13: Sensor side cable configuration

SmartGate

SmartGate is the result of a project by two Politecnico di Milano students, developed in ANTLab until December 2018 (fig. 3.14). It is a complete system on its own, able to detect people passage through a gate, and related direction [4]. This system was found to have an accuracy of 97% in indoor environments and, if positioned at the door of the experiment room, can be very useful for ground truth computation. SmartGate is also Wi-Fi and Arduino based, and so programmable through an USB cable and Arduino IDE. Source code has been shared with this project and so SmartGate has been properly configured to connect via Wi-Fi to the local network access point. Data about people entering and leaving the room are

so instantly sent to the MQTT broker on a dedicated topic and are so immediately made available to the server to keep track of the number of people actually present in the room during the data collection phase.



Figure 3.14: SmartGate used for ground truth computing

3.2.2 Server Side

Server side part consists in a single Raspberry Pi 3 acting as access point for the local network, as MQTT broker, as database server and also implements, with Node-Red, the MQTT client and all the features necessary for data acquisition. A simplified block diagram about server functionalities is shown in figure 3.15.

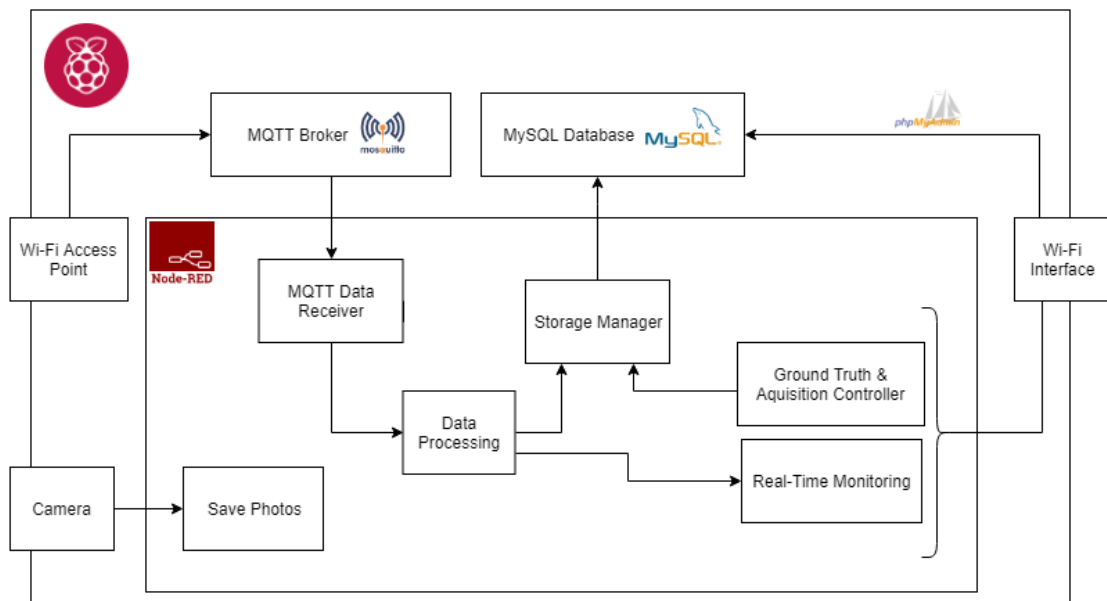


Figure 3.15: Raspberry Pi block diagram

Raspberry Pi has been configured to have two Wi-Fi interfaces available: the first is the standard one, integrated into Raspberry, and is used to connect the server to an external network, making it possible to access remotely database server and Node-Red graphic interface. The second is obtained by connecting a Netgear Wi-Fi USB adapter (fig. 3.16) and configuring it to act as an access point interface in a standalone network.



Figure 3.16: Netgear WNA3100 Wi-Fi USB adapter

Access point configuration consists of the following steps, directly executable from a local Linux terminal or remotely via SSH:

1. Setting a static IP address to the interested wlan interface in the `dhcpcd.conf` file.
2. Installing and configuring a DHCP server, specifying the range of addresses that will be assigned to devices connected to the network. For this purpose, thanks to its ease of configuration, Dnsmasq server has been used.
3. Installing and configuring the access point host software, specifying interested wlan interface, network SSID (the name of the network) and network wpa password. Hostapd software was used as host software and it was set up to run as a “daemon” process on every system launch.

A Mosquitto MQTT message broker was chosen to manage data communication. Mosquitto is an open source project written in C which stands out for being a very light and easy to use server, compatible with Raspbian operating system. Once installed and configured to be accessible both locally and remotely, it automatically runs in background on every system launch, and is so always available for interested clients.

A camera was also properly configured to interface via USB port with the Raspberry Pi, in order to provide the server with periodic images of the surrounding environment useful, in some cases, for manual data labeling.

Another fundamental component of Raspberry Pi software is the database server, implemented with MySQL. Database was configured to be accessible, after an authentication phase, both locally and remotely. This allows both the Node-Red software to create new tables and insert real-time data, and external users to access the same tables and download them in the desired format. In particular, for remote access, a classic Apache web server has been installed on Raspberry and configured to run PhpMyAdmin database administration application. More information about database and table structure are provided in the following sections.

Node-Red client

The core of the server activities is certainly made up of the Node-Red client. It is able to interact with all other components listed above to perform data acquisition

with automatic labelling, and also to implement an interactive Graphical User Interface (GUI) for real-time data monitoring and control. The application is developed in a single flow that deals with all these aspects. An overview of the final flow is shown in figure 3.17. Each node, coming from a library, is a program feature, that may accept an input message and activate upon its arrival. Each node could also have one or more output messages, which can in turn activate other nodes. Nodes usually require to be properly configured specifying some functional parameters. The orange nodes are instead customizable ones, able to execute any Javascript function.

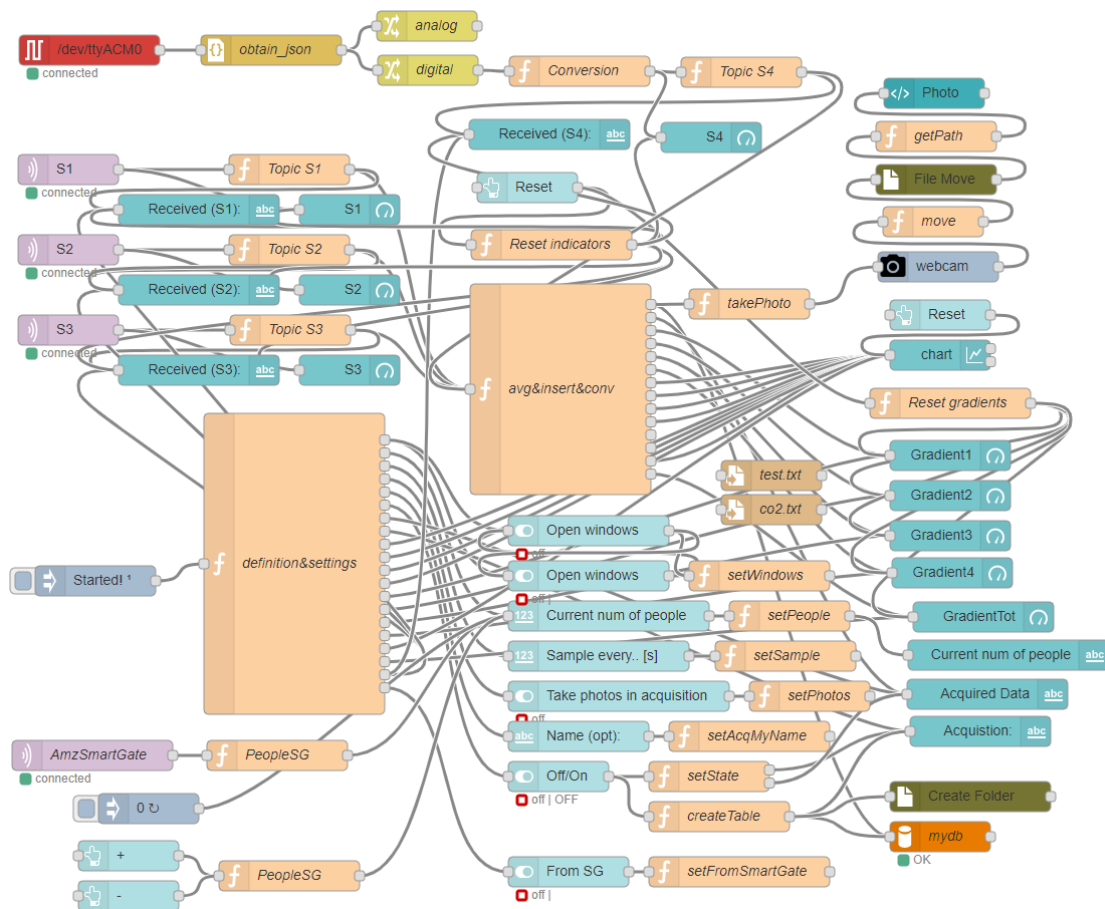


Figure 3.17: Node-Red application code

Following is a list of the main components that are part of the application.

- *Initializer*: the flow is mainly based on concurrent update of common global variables. An initializer function is performed on flow start, with the purpose of creating all the necessary variables, setting them to a specified initial value. Also some functional parameters are specified in this block.
- *MQTT receivers*: Each MQTT receiver connect to the local (127.0.0.1) broker at predefined port 1883, and subscribe to a specific topic. In particular there is a receiver for each wireless sensor and a specific receiver for the SmartGate

topic. Co2 data arrives as simple strings and specific functions are then charged to transform them into the appropriate numerical format, adding as information the origin topic. SmartGate string data are instead on a single topic and possible values are “*entry*” or “*exit*”.

- *Serial receiver*: A single serial receiver is configured to read Co2 data from the directly connected Arduino. The connection is opened on the interested Raspberry COM port. Data arrives as single bytes and when a special character, called separator (`\n`), is read, the corresponding string is forwarded to the following node.
- *Data processing*: A big Javascript node deals with management of input data and global variable update. It also perform some average operations. In fact data arrives from sensor at high frequency and we want to save them in database at a lower frequency, specified by users. Whenever the sampling period expires all data collected in global arrays are averaged and results are passed as outputs for visualization and saving purposes. Also a measure of the gradient with previous sample and a total average between all sensors are calculated, just for visualization purposes.
- *Database manager*: A specific node is dedicated to authenticated connection with the local database server, on the default 3306 port, and accepts incoming specific queries. More information about database tables structures and performed queries are provided in next section.
- *Dashboard*: A specific GUI has been developed using Node-Red official dashboard library. It is not only for data visualization purposes, but also to control some basic parameters of the program. A screenshot of the dashboard is reported in figure 3.18. A chart is responsible for representing the time sequences of the various sensors data and at the same time representing the average between them. Some indicators represent instead last values received and current Co2 gradient from each sensor. A numerical counter of data received from each sensor is also available in order to monitor if sensors are currently sending data. Another fundamental block is the acquisition controller, through which it is possible to start and stop acquisition phase, also specifying a name for next acquisition and monitoring the number of acquired data. If acquisition is off, data are still received and plotted in the chart, but not saved in the database. There is also a photo viewer, which shows the last picture taken by the webcam, with a switch to select if we want to save photos or not during acquisition phase. Last but not least is the ground truth controller. It allows user to manually adjust the current number of people present in the room and to specify whether or not to adjust this variable based on the data received from the SmartGate. From here, it is also possible to specify the opening status of windows.
- *Mobile dashboard*: A dashboard optimized for mobile devices has also been implemented, just for ground truth setting (fig. 3.19). It provides a controller for the number of people, and one for windows state, connected to the respective controllers of the main dashboard.

- *Photo saving*: The photo saving block is in charge of periodically taking photos from the webcam and saving them in a specific folder. A folder is created for each new acquisition, and photos are saved at the same frequency as data saving in the database. The name of each photos is the current timestamp.

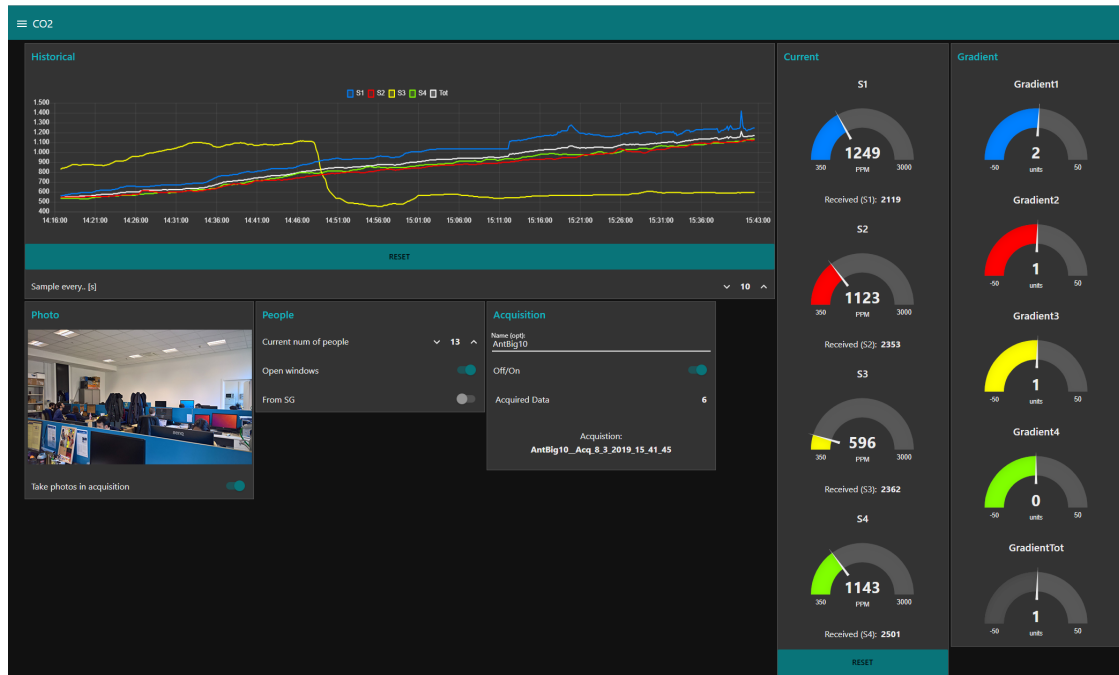


Figure 3.18: Node-Red GUI screenshot

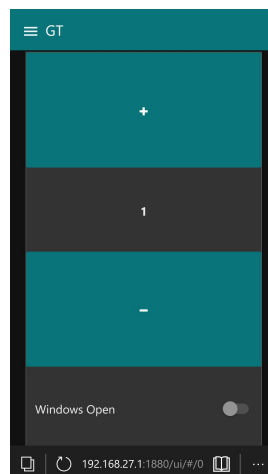


Figure 3.19: Node-Red mobile GUI screenshot

GUI interface is accessible also remotely via web browser, by typing in the URL bar “*raspberryIpAddress:1880/ui*”. A dedicated menu let user to select the visualization mode, between laptop and mobile.

Database

Database is divided into individual tables, one for each single acquisition. Table name is the same name of the acquisition and is composed by a string, chosen by user, concatenated to the start time of the acquisition. Each tuple consists of the following fields:

(timestamp, S1, S2, S3, S4, people, windows)

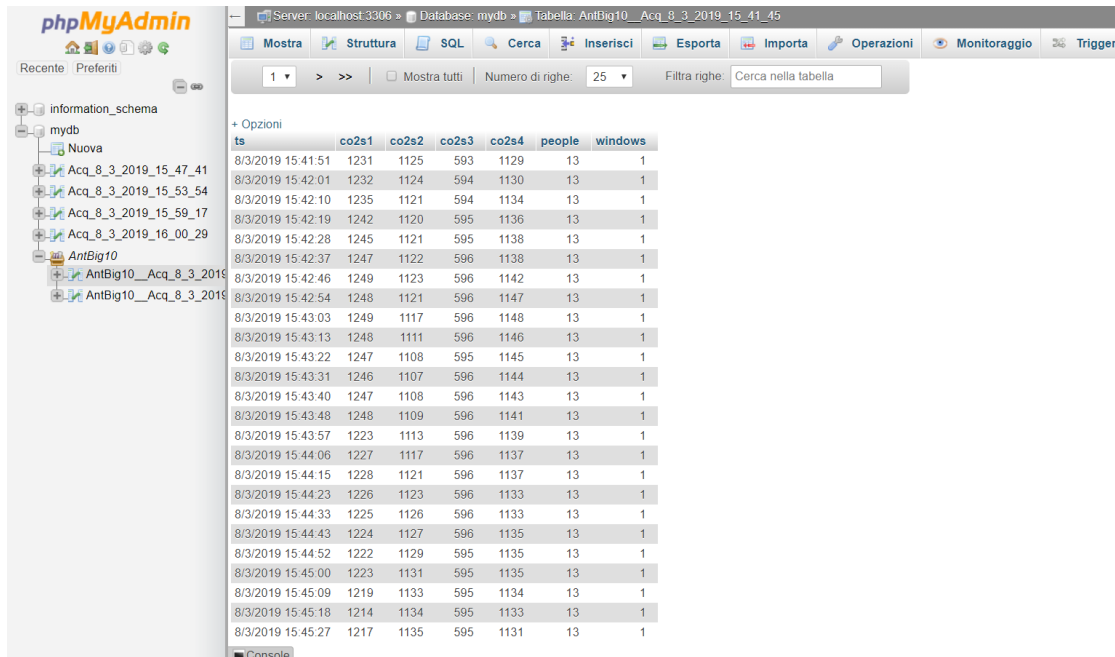
Timestamp is the key, to uniquely identify the tuple, and consists of the date concatenated to the hour of its insertion in the table. Data from S1 to S4 are integer that represent carbon dioxide concentration in PPM read by the corresponding sensor. People is the ground truth about how many people are in the room and windows is a boolean parameter concerning the status of windows opening. The following query is so performed by Node-Red when a new acquisition begin:

```
CREATE TABLE tableName (ts varchar(256), co2s1 int(32), co2s2 int(32), co2s3 int(32), co2s4 int(32), people int(32), windows boolean)
```

An insertion query is instead executed whenever a new tuple is ready to be inserted in the table. An example can be:

```
INSERT INTO tableName VALUES ('25/02/2019 18.41.34', 727, 715, 785, 698, 5, 0)
```

Once an acquisition is terminated the corresponding table can be easily downloaded in CSV format from PhpMyAdmin. In figure 3.20 is shown a screenshot of a table.



ts	co2s1	co2s2	co2s3	co2s4	people	windows
8/3/2019 15:41:51	1231	1125	593	1129	13	1
8/3/2019 15:42:01	1232	1124	594	1130	13	1
8/3/2019 15:42:10	1235	1121	594	1134	13	1
8/3/2019 15:42:19	1242	1120	595	1136	13	1
8/3/2019 15:42:28	1245	1121	595	1138	13	1
8/3/2019 15:42:37	1247	1122	596	1138	13	1
8/3/2019 15:42:46	1249	1123	596	1142	13	1
8/3/2019 15:42:54	1248	1121	596	1147	13	1
8/3/2019 15:43:03	1249	1117	596	1148	13	1
8/3/2019 15:43:13	1248	1111	596	1146	13	1
8/3/2019 15:43:22	1247	1108	595	1145	13	1
8/3/2019 15:43:31	1246	1107	596	1144	13	1
8/3/2019 15:43:40	1247	1108	596	1143	13	1
8/3/2019 15:43:48	1248	1109	596	1141	13	1
8/3/2019 15:43:57	1223	1113	596	1139	13	1
8/3/2019 15:44:06	1227	1117	596	1137	13	1
8/3/2019 15:44:15	1228	1121	596	1137	13	1
8/3/2019 15:44:23	1226	1123	596	1133	13	1
8/3/2019 15:44:33	1225	1126	596	1133	13	1
8/3/2019 15:44:43	1224	1127	596	1135	13	1
8/3/2019 15:44:52	1222	1129	595	1135	13	1
8/3/2019 15:45:00	1223	1131	595	1135	13	1
8/3/2019 15:45:09	1219	1133	595	1134	13	1
8/3/2019 15:45:18	1214	1134	595	1133	13	1
8/3/2019 15:45:27	1217	1135	595	1131	13	1

Figure 3.20: PhpMyAdmin database screenshot

Chapter 4

Machine learning study

4.1 Machine learning overview

Machine Learning (ML) is an artificial intelligence branch, covering all algorithms and statistical methods that a computer system uses to perform a specific task without having specific sequential instructions, but having knowledge of a large amount of data. These methods start from the last decades of the XX century in various scientific fields, such as artificial neural network, data mining, computational statistics etc.. Machine learning methods are nowadays used in a wide variety of applications, such as computer vision, natural language processing, weather forecasts and many others. This section is an introduction aimed at better understanding the methods used for the specific implementation in 4.3.

A formal and widely quoted definition about a machine learning algorithm was provided by Tom M. Mitchell in [29]: “*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E* ”. Machine learning techniques are mainly divided into these categories, dependent on the type of dataset available and the specific scope:

Supervised Learning is the technique that allows an algorithm to build a mathematical model from a set of data that contains both input and target information. This model is then used to make target predictions on new input data.

Unsupervised Learning aims to create the mathematical model from a set of data which contains only inputs and no desired output labels. It is commonly used to find some structures in data, like repeated pattern, or to group input data into categories. Some semi-supervised learning methods can be used in situation in which only a portion of data have no label information.

Reinforcement Learning is a technique based on performance. The purpose is to place an agent in a dynamic environment, able to make decisions and improve its performances on each new input data, not based on a specific target value but based on a reward/penalty function. This involves finding a balance between exploration (of uncharted territory) and exploitation (of

current knowledge) on each new input data. Some typical application are in autonomous vehicles or in learning to play a game against a human opponent.

Due to the form of our dataset and the scope of our study, we used supervised learning techniques, so let's focus on details and specific algorithms for this kind of problems.

As specified above, supervised learning method involves the use of a labeled dataset to obtain the best model which maps input parameters, called features, on a specific output parameter, called target. The model will be able to make precise predictions on new data, provided that they come from a sample space similar to the one with which the model was built. Supervised learning is further divided into two categories based on required output: regression is used for numerical continuous targets, while classification for discrete targets divided in classes. The whole ML process can be divided into two basic parts: data pre-processing and best model calculation.

Pre-processing is about preparing input features in such a way as to be as appropriate as possible to represent the problem of model building. This phase includes operation such as horizontal outliers filtering of noisy or redundant information, data dimensionality reduction or new feature computation, feature normalization and transformation. These steps are essential to the subsequent creation of a good model and generally take a considerable amount of time in the entire process. The product of data pre-processing is the final set used to train the model.

The search for the best model is instead characterized by the choice of the best learning algorithm, which identifies a specific class of model, and the choice of some model parameters, which define some characteristics concerning the particular model structure. A generic model can be represented as a function:

$$f_{A_1..A_o}(X_1, \dots, X_n, W_1, \dots, W_m) \quad (4.1)$$

where A_o are parameters about model structure, X_n are the input data features and W_m are the weights which the model creation algorithm is able to optimize based on available data. The output of the function is the desired target value. In order to be able to make the above specified choices about model class and structure, and have an effective feedback regarding their correctness, data are generally divided into three strictly disjoint sets, which characterize also the three main phases of this process:

Train This part of data is the one used by the learning algorithm to discover the best model weights (W_m), which are calculated trying to reduce the gap between the predicted and the real target value. A common problem to be solved in this phase is the “overfitting”, which is related to a training phase which adheres too much to the train dataset, without considering its imprecision, and is therefore inadequate for predictions on future data.

Validation This part of data is the one used by the learning algorithm to discover the best model parameters (A_o). This is typically obtained with a “brute force” approach, that is cycling on the possible values of the parameters in a set, or on possible combinations of values in case of multiple parameters,

and repeating the training phase each time. Validation set is then used to evaluate the various models and choose the best one.

Test This part of data is the one dedicated to the final evaluation of the model found (fixed W_m and A_o), based on various error metrics, regarding a direct comparison between what the model predicts on test input data, and their real target value. In order to have a realistic evaluation it is important that this part of data has not been used either for parameter setting (validation) nor for weight optimization (train).

This division is just for the analysis phase. Once the algorithm and the best parameters have been chosen, a final model will be trained with all available data, and implemented with specific tools.

4.2 Problem definition

The aim of the ML part of this project is to define a process that, on the basis of historic data collected with the system presented in chapter 3, is able to find the best model M to predict future value of the number of people in the selected room, having future information about carbon dioxide concentration (fig. 4.1, 4.2).

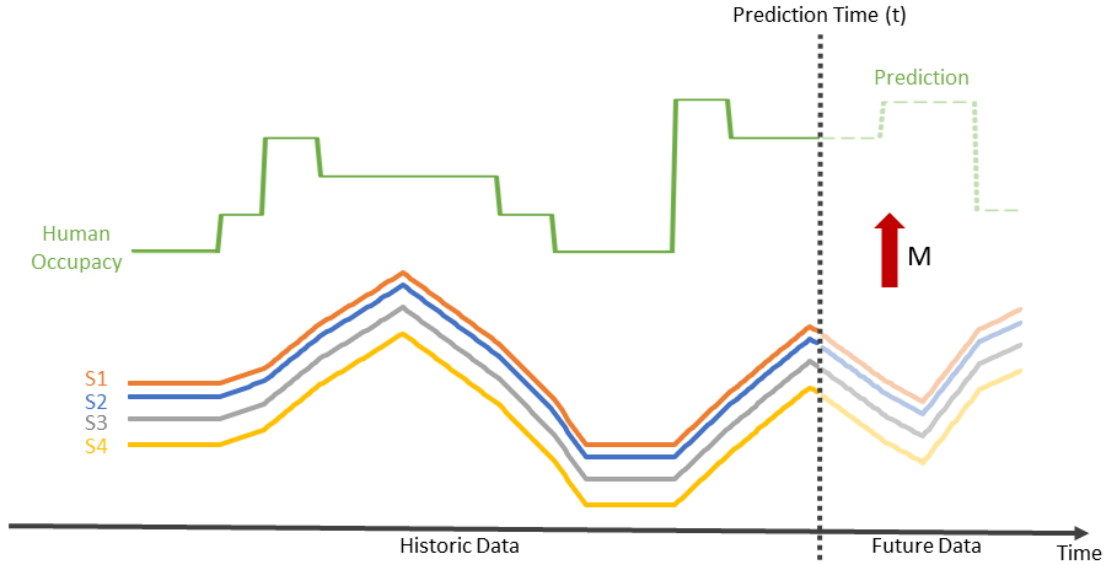


Figure 4.1: Real-time prediction scenario

Assume we have a set A of N disjoint acquisitions:

- $A = \{A_1, A_2, \dots, A_n\}$

Each acquisition A_n consists of a different Q number of samples, identified by the timestamps serie $TS_n = \{ts_{n1}, ts_{n1}, \dots, ts_{nq}\}$. The time series associated with the n -th acquisition are:

- $C_n = \{C_{n1}, C_{n2}, \dots, C_{nm}\}$ where each $C_{nm} = \{c_{nm1}, c_{nm2}, \dots, c_{nmq}\}$

- $O_n = \{o_{n1}, o_{n2}, \dots, o_{nq}\}$

C is the serie of the carbon dioxide concentrations collected from M sensors. O are the human occupancy numbers. Also boolean values about windows situation were considered in some parts of this work:

- $W_n = \{w_{n1}, w_{n2}, \dots, w_{nq}\}$

A single acquisition A_n can so be finally defined as:

- $A_n = \{TS_n, C_n, O_n, W_n\}$

The output model of our ML process must be in the form $M = f(c_1, c_2, \dots, c_m)$.



Figure 4.2: ML process definition

4.3 Implementation

Machine Learning process was structured as a flow of functional blocks, where each block is configurable in its basic parameters and can be completely deactivated and bypassed. The language chosen is Python, an interpreted scripting language, which perfectly fits this programming style and provides many libraries for machine learning (such as scikit-learn) and graph plots. Python nowadays is certainly the most used language for ML purposes. A .py file was created for each functional block and a main function has been programmed to call specific functions in the correct order. Spyder IDE was used for programming and execution purposes. In figure 4.3 is shown the entire flow for the machine learning process. Details on the implementation of each block are provided in the following sections.

4.3.1 Data read

The first step of the whole ML process is the reading of the CSV files. Each CSV file refers to a specific acquisition. Reading is performed through the library Pandas and each acquisition is saved in a structure of the type $A_n = \{TS_n, C_n, O_n, W_n\}$ as specified in section 4.2. In order to represent this type of structure, each tuple is saved as a Pandas Series, an array with axis labels. Labels are represented by an index of non repeated integers. It is important that all series in the same acquisition are associated to the same index.

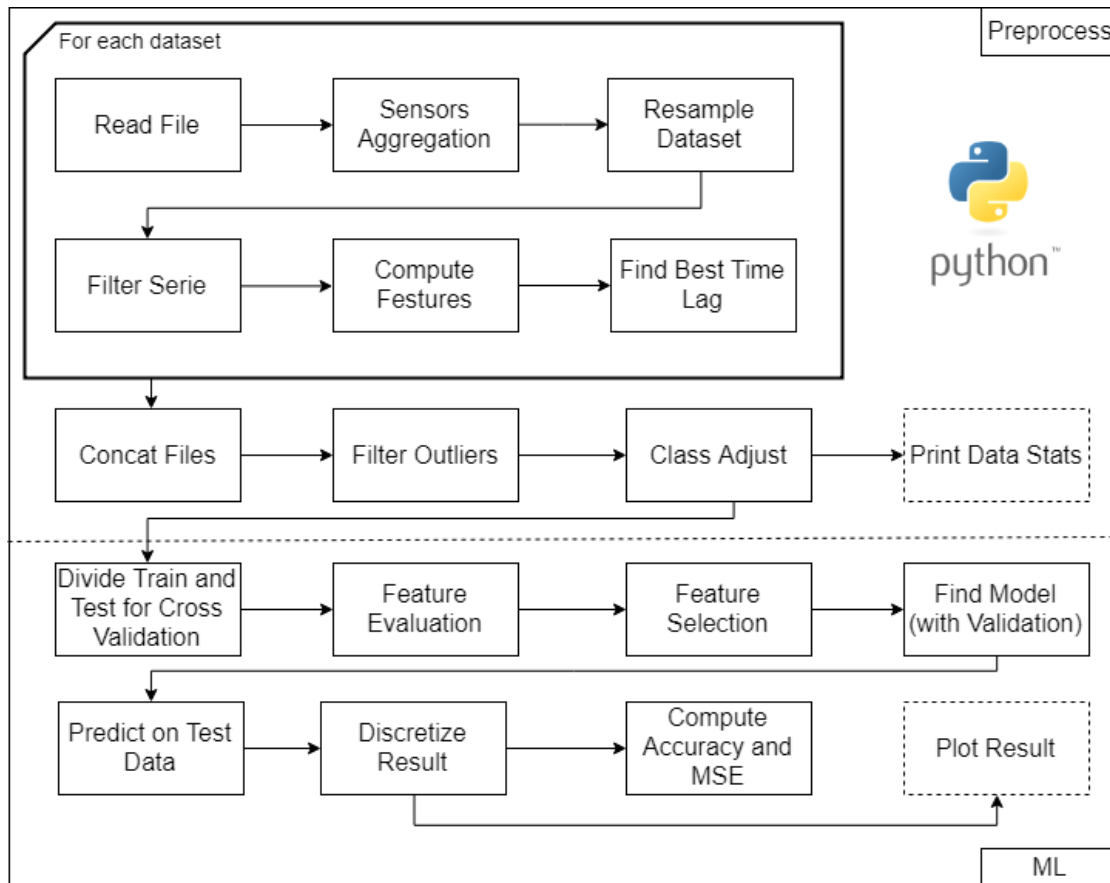


Figure 4.3: ML process execution flow

Binarization

During this phase, if specified by a global parameter, a binarization process is available. It consists of modifying the ground truth tuple O_n in order to represent the binary free/occupied state of the room. The multiple classes, concerning the number of people present, are therefore reduced to two with a simple algorithm (alg. 1). The 0 class represent a vacant room, the 1 class represent an occupied room.

Algorithm 1: Data binarization

```

for  $o$  in  $O$  do
  if  $o < 1$  then
     $o = 0$ ;
  else
     $o = 1$ ;
  end
end

```

4.3.2 Sensors aggregation

This step concerns the aggregation of the sensors data $C_n = \{C_{n1}, C_{n2}, \dots, C_{nm}\}$ in a one or more C_n series. Aggregation is performed horizontally between sensors samples with corresponding index. Aggregation does not necessarily include all the M sensors. A specific sensor, or even all the sensors, can be excluded from aggregation and then considered as independent features. After this step we will use M to indicate the number of aggregations present in the tuple C_n . Three aggregation methods have been implemented and are discussed below.

Average

This method consists of a simple average of the corresponding samples, in which all sensors have the same importance.

$$C_{nq} = \frac{\sum_{m=1}^M C_{nmq}}{M} \quad (4.2)$$

Median

This method consists in taking the median between the corresponding samples. Median in a series of N ordered items is defined as the central value of the series, which is the value occupying the position $\frac{N+1}{2}$ if N is odd, or the average between values in positions $\frac{N}{2}$ and $\frac{N}{2} + 1$ if N is even. An advantage of the median is that is independent from the possible error of a single sensor, which could greatly increase the value acquired due to a person who expires momentarily in its proximity.

Penalize gap

This method consists in a weighted average between the corresponding samples, in which are penalized samples that are far from the average itself. This method points to solve the same problem of sensor error, in a lighter way than the median.

$$C_{nq} = \frac{\sum_{m=1}^M C_{nmq} \times \frac{1}{\frac{1}{M} \times \sum_{i=1}^M |c_{niq} - c_{nmq}|}}{\sum_{m=1}^M \frac{1}{\frac{1}{M} \times \sum_{i=1}^M |c_{niq} - c_{nmq}|}} \quad (4.3)$$

4.3.3 Resample dataset

In this phase each dataset can be resampled at a lower desired frequency with respect to the acquisition frequency. If f_n is the sample frequency of the n -th acquisition a tuple of integers factors $R = \{r_1, r_2, \dots, r_n\}$ can be set as a program parameter, and new frequency of each acquisition will be:

$$f_n^{new} = \frac{f_n^{old}}{r_n} \quad (4.4)$$

An aggregation method was defined specifically for each series, in such a way to keep data consistent with the new frequency.

Timestamps TS_n is aggregated by taking the first ts_{nq} of each temporal window as a time indicator for the entire window.

Co2 C_n is aggregated by calculating the average of all samples of each temporal window. Result is then approximated to the nearest integer. This operation is repeated for each aggregation of sensors data.

Occupancy O_n is aggregated in the same way as the carbon dioxide data. In case of binary occupancy the most common value is considered.

Windows W_n is aggregated by taking the most common value of each temporal window.

4.3.4 Filter series

This step regards the filtering of each C_{nm} temporal series aggregated in step 4.3.2. The Butterworth filter was chosen for this purpose. It was first described in 1930 by the British engineer and physicist Stephen Butterworth in [12]. A Butterworth filter is one of the simplest type of signal processing filter, designed to have a frequency response as flat as possible in the passband, and to roll off towards zero in the stopband. When viewed on a logarithmic Bode plot (fig. 4.4), the response slopes off linearly towards negative infinity. The order of the filter defines the magnitude of this decrease. A first-order filter's response rolls off at -6 dB per octave (-20 dB per decade). A second-order filter decreases at -12 dB per octave, a third-order at -18 dB and so on. Another important parameter of this type of filter is the cutoff frequency, which is the frequency that divides the passband from the stopband.

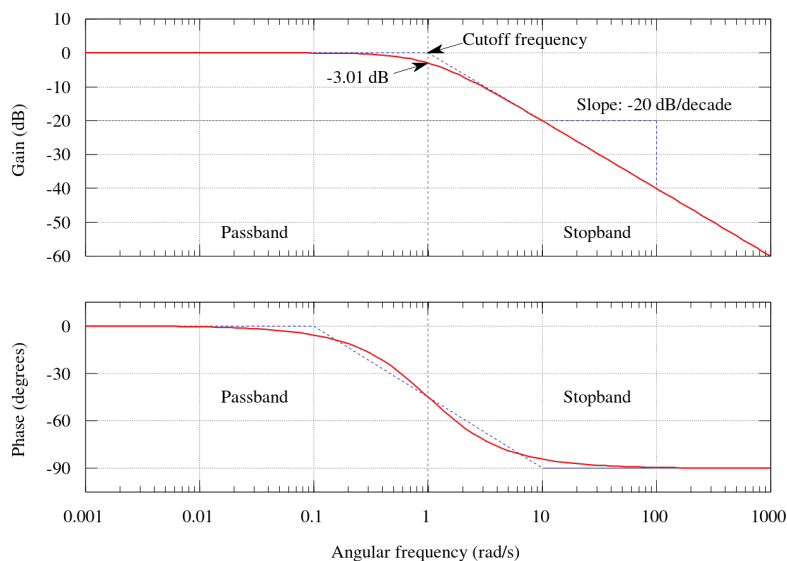


Figure 4.4: Bode plot of a first-order Butterworth filter

Series filtering has been implemented using butter function of the scipy.signal library. This function provide a classical implementation of the Butterworth filter, settable in its order and critical frequency parameters. Filtering Co2 data, together with the resampling and aggregation operations, can lead to innumerable advantages. All these operations tend in fact to flatten sudden changes in Co2 quantity signal, which are often due to sensor measurement errors or to people breathing near the sensor itself. However, it is necessary to find the optimal exploitation of these operations. If resampling frequency factor is too large, or if the Butterworth filter is too invasive, it will result in an imprecise model. Optimal may change from an acquisition to another. In figure 4.5 is shown an example of application of a second order Butterworth to an imprecise short acquisition.

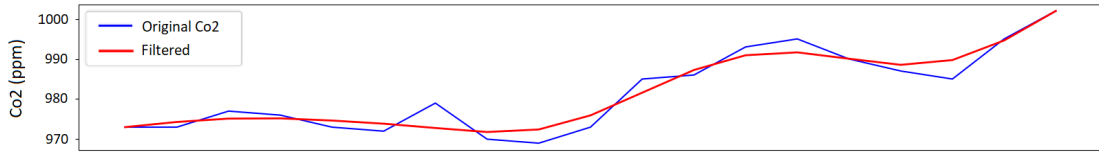


Figure 4.5: Example of second order Butterworth on Co2 data

4.3.5 Compute features

In this step new features are calculated from standard series and added to an input tuple, related to the n -th acquisition, which we call I_n . Below is a list of selected input features, together with an analysis of the advantages they could bring to the model computation.

Co2 Co2 absolute values are the results of the aggregation and filtering steps. They don't need further operation and each aggregation can be added as an input feature itself, based on the scenario and the aggregation context. A selection of some C_{nm} in C_n tuple is so performed and results are added to the input feature tuple I_n . We expect the absolute amount of carbon dioxide to be the most related variable to the result, but that is not enough to have a precise estimate. In fact the absolute concentration could depend also on other phenomena occurred previously, such as windows opening, and not only on the number of people breathing in the room.

Gradient If the absolute concentration might not be enough, its derivative, which is the way in which it is varying in time, could be another good indicator. For this purpose a number K of Co2 derivatives, where K is a configurable parameter of the process, are calculated on a fixed aggregation m , resulting in a new tuple $G_{nm} = \{G_{nm1}, G_{nm2}, \dots, G_{nmk}\}$, where $G_{nmk} = \{g_{nmk1}, g_{nmk2}, \dots, g_{nmk(q-k)}\}$. Each g_{nmkq} is the average variation of last k sample of Co2, in aggregation m and acquisition n , and is calculated as specified in formula 4.5.

$$g_{nmkq} = \frac{\sum_{i=0}^{k-1} C_{nm(q-i)} - C_{nm(q-i-1)}}{k} \quad (\text{where } q > k) \quad (4.5)$$

Adding G_{nm} to the input tuple I_n , forces to deprive all other input series of the first K samples, not present in the various G_{nmk} , reducing dataset size. We expect the model to improve when the gradient is added as a feature, but only if together with carbon dioxide absolute value. In fact, even if Co2 variation should be highly related to the number of people who are breathing and therefore emitting carbon dioxide, this correlation may differ based on the current saturation of Co2 in the room. Let's clarify this concept with an example. Suppose that a man is breathing in a room, emitting a constant quantity of Co2 called x . The air in the room is currently quite clean. Let's call y the actual low Co2 concentration in PPM. Therefore suppose that each breath contributes to increasing the concentration y by x units. What happens after a long time, if the windows are not opened and the room has no air exchange with outside? When y become very high, air becomes saturated with carbon dioxide, and each breath will result in a smaller and smaller change in the total Co2 concentration. At high absolute concentrations we therefore expect that small gradients are associated with a greater number of people, compared to those we would have had with the same gradients in case of low absolute Co2 concentrations.

Baseline difference Another feature which could be considered is the direct comparison between values of two different C_{nm} aggregation. It could be very useful if a series which refers to Co2 values outside the room is taken as baseline, and subtracted sample by sample to the correspondent indoor concentration. This allows to obtain a relative concentration value with respect to a selected baseline ambient, which can be outdoor or also the corridors of the building, making the model more scalable and adaptable to different situations. This new series is called D_n .

Time class In order to consider a possible periodicity of Co2 values between one day and another, it may be useful to insert a feature concerning samples timestamp. If during the night, for example, the room has always been empty, the model can learn to predict its vacant status especially thanks to this parameter. Time series is created as a discretization of the TS_n series with a daily period, with respect to a configurable parameter L , regarding the number of discretization classes with a maximum granularity in the order of seconds. Let's suppose that each ts_{nq} is expressed in “seconds from midnight” and that selected L is a divisor of 86400 (number of seconds in a day). Corresponding t_{nq} is obtained as specified in equation 4.6.

$$t_{nq} = \left\lfloor \frac{ts_{nq}}{86400/L} \right\rfloor \quad (4.6)$$

Final form of the input tuple for the machine learning algorithm is so, for each acquisition n , $I_n = \{C_{n1}, \dots, C_{nm}, G_{nm1}, \dots, G_{nmk}, D_n, T_n\}$. Ground truth tuple is instead defined as $GT_n = \{O_n, W_n\}$.

4.3.6 Time lag

An intrinsic time delay is inevitably present in collected data. This delay is due to the fact that when one person enters a room, it will take some time before the Co2 level in the air increases proportionally. This effect is gradually increasing with the size of the room. Since we will try to build a model considering singular samples as simple tuples, and not as a temporal sequence of data, ground truth series must be realigned to input series in the best way. To solve this problem we used an approach similar to what proposed in [6], in which best time lag is found by searching in the data themselves. This method involves brute-force cycling on a possible sets of delays, expressed in number of samples, between zero and an upper bound (UB) that has to be manually defined, based on the size of the room and the sampling rate. Separately for each acquisition, for each possible lag in the set, ground truth data GT_n are shifted and line of best fit (LBF) is calculated for each couple $\{C_{nm}, O_n\}$ with a simple linear regression (m is fixed on the most relevant Co2 aggregation). In figure 4.6 is shown an example of LBF on a generic acquisition scatter plot. For each LBF is then calculated the Normalised Root Mean Squared Error (NRMSE) on the same data used to create it, as indicator of linear correlation. Best time lag corresponds to the best NRMSE. Since each acquisition n can result in a different value of best time delay, an average between all lags is finally calculated and the same time shift is applied to all acquisitions. Same process may also be performed with more complicated models than a simple linear regression, losing in computational time. As in the case of gradient computation, a quantity of data equal to the value of the best time lag is lost due to the shift.

4.3.7 Dataset fusion

In the literature, as specified in section 2.5.2, all models that are based on Co2 measurement are build with one long acquisition, which in some cases exceeds 15 days. The aim of this project is instead to reach good results with in a shorter time period and with small fragmented acquisitions. After being processed separately in the previous steps, the N acquisitions must be merged together in a unique big acquisition in order to be processed by the machine learning algorithm. This operation is made in an horizontal way, both on series of the various I_n and on series of the various GT_n . An operation of concatenation is performed for each type of matching series, concatenating arrays of values and creating an index of increasing integers starting from zero up to $Q-1$, called Q the total number of samples. In this way I_n become a unique $I = \{C_1, \dots, C_m, G_{m1}, \dots, G_{mk}, D, T\}$ and GT_n becomes a unique $GT = \{O, W\}$. From this point on, the only big acquisition can no longer be treated as a time series, as a union of discontinuous time lines.

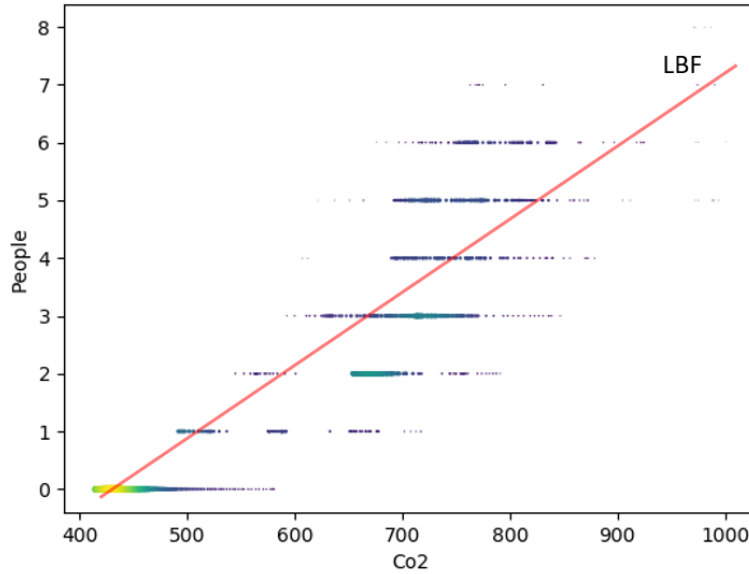


Figure 4.6: Example of LBF on a generic acquisition

4.3.8 Filter outliers

In statistics the term “*outlier*” is used to indicate, in a set of observations, a sample that is distant from all other observed points. This could happen, for example, due to measurement errors. Outliers are generally excluded from the dataset because they can cause problems in creating the model. Different filters were designed in order to horizontally exclude some samples in which one or more features leave their own domain. The extremes of the domain can be manually set, but common values are:

- $400\text{ppm} < c_{mq} < 2000\text{ppm} \forall m, q$
- $-50\text{ppm} < g_{mkq} < 50\text{ppm} \forall m, k, q$
- $o_q \geq 0 \forall q$

This kind of filter is also used in order to exclude all samples in which w_q is equal to 1, if you want to create a model excluding data sampled with open windows. Class adjust can also be seen as a part of the horizontal filtering process. This step deletes some random elements from the larger occupancy ground truth classes. This is due to the large amount of zeros values acquired during the night, which could lead to the creation of a model that predicts zero class more than it should. This step ends the preprocessing phase. Filtered data are passed to next functional blocks for model creation and evaluation.

4.3.9 Two step estimate

The main problem to solve when trying to create an accurate model for indoor people counting starting from environmental data, such as Co2, is that other variables, in addition to people presence itself, could impact on the measurement.

Final model M actually used for prediction of O from I is an aggregation of these three different models. In the following sections all the main components of this method are explained in details.

4.3.10 Cross-validation

First choice is about external validation method, which means choosing how and how many times divide data in the two test and train sets, in order to evaluate the created model. The selected approach for external validation of the model is K-folds cross-validation (fig. 4.8). This technique involves the division of data into K disjoint folds, where K is a manually fixed parameter. In this way each fold consists of Q/K distinct samples. Data division in corresponding folds is completely random. All steps regarding model creation and evaluation, presented in the following sections, are so repeated K times. At each step data coming from a single fold are excluded from model construction and used later only for testing purposes. All other data are instead used in the training phase. This will lead to the creation of K distinct models and K distinct evaluation results, which will be integrated together later, with a specific aggregation method. Evaluate a model on data which were not used to build it, is also a good way to find any overfitting problems, as well as being an insight on how the model will generalize to an independent dataset. Cross-validation method is widely used in ML because it allows to test the validity of a model independently of a particular selection of test data.

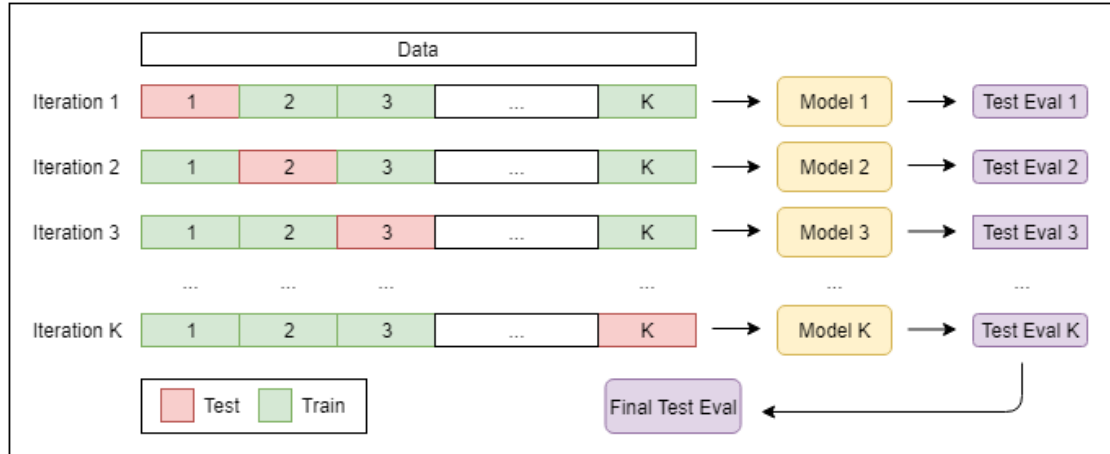


Figure 4.8: K-fold cross-validation

Learning curve variant

In order to plot learning curves, which are other important indicators about model performance, a variant about the external validation method was implemented (fig. 4.9). A learning curve is a graphical representation of how an increase or decrease in learning (performance) comes from greater experience. In ML context performance is measured by the accuracy of the learning system (or the selected

evaluation metric), while experience is the number of training samples. Data are therefore divided into K folds, just like in cross-validation, but iterations start with only a single fold for model training and add at each successive iteration a training fold, up to the $K-1$ iteration which comes with only one test fold. A second substantial difference is that the prediction phase is not carried out only on test data, but also on train data. This allows to evaluate the performance of the model also on data used for its creation. Two learning curves can therefore be plotted. One is the test learning curve, whose accuracy may increase with experience. The other is the training one, whose accuracy is expected to decrease when the number of train samples increase. An ideal aspect for the two curves is shown in figure 4.10. As the number of train data increases it should be more difficult for the model to adhere to all samples. If training score does not decrease with experience model is probably overfitting. On the other hand, if test score does not increase with experience model is not learning. It could be due to the fact that selected algorithm is not adequate to the problem, or that maximum performance for these data can be achieved even without using them all. The two curves should never intersect. If test performance far exceed train performance it means that the model is predicting approximately random values.

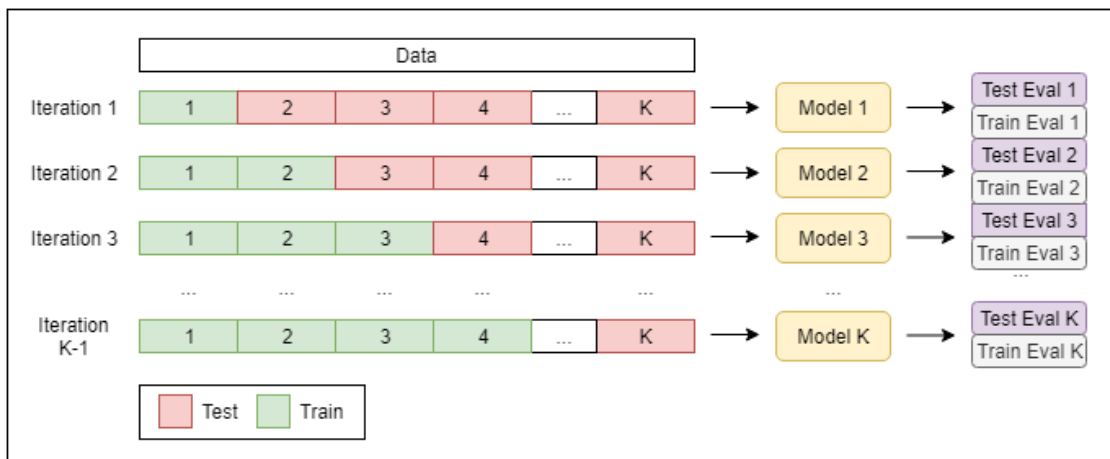


Figure 4.9: Learning curve computation

4.3.11 Feature evaluation and selection

The purpose of this step is to evaluate the correlation of each individual feature in I with the target series. In fact it is advisable to discard features that have a poor correlation with target, as it could result in a more precise model. This approach of evaluating each feature correlation with target individually is called *univariate* feature evaluation. This operation is executed separately both before windows model training and before occupancy models training. In fact we could get very different correlations if calculated with respect to W or O . The metrics available in the Sklearn library for univariate evaluation of the features are basically two:

F-test Method based on F-test estimates the degree of linear dependency between two random variables. This dependency is measured both in term of F value

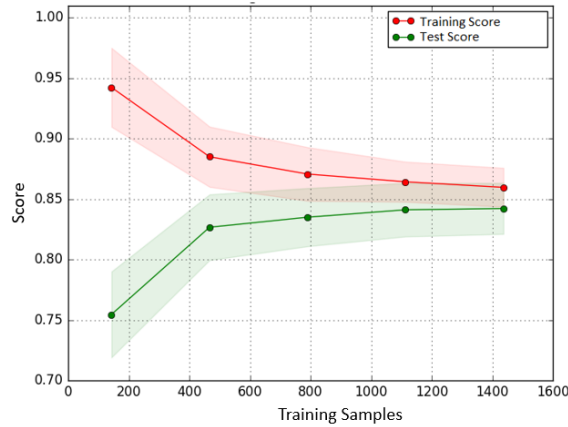


Figure 4.10: Ideal learning curve

and in term of p-value. The F value is the ratio of the mean regression sum of squares divided by the mean error sum of squares, given by formula 4.8, where y_i are real target values, y_i^p are predicted target values and \bar{y} is the mean of real target values.

$$F = \frac{\sum_{i=1}^n (y_i^p - \bar{y})^2}{\sum_{i=1}^n (y_i - y_i^p)^2} \quad (4.7)$$

Its value will range from zero to an arbitrarily large number, and high values correspond to larger dependency. P-value is instead calculated as the probability that the hypothesis of a null model (a model with all of the regression coefficients equal to zero) is true. It can be denoted as $\text{Prob}(F)$. If $\text{Prob}(F)$ is small it means that there are small chances that regression parameters are zero. It would imply that input variable is not purely random with respect to the target. Since it is a probability, its value will obviously be between zero and one.

Mutual information MI method, unlike the previous one, aims to capture any kind of dependency between variables, not only linear dependencies. Result is a non-negative value which measures the dependency between the two variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency. A possible disadvantage of this method is that it requires more samples than the previous one to get a precise estimate.

Since our variables in I seem to be related to the target in a non-linear way, and that the number Q of our samples is generally sufficiently large, MI method seems to be the most appropriate for this problem. Both F-test and MI are available both for a regression problem and for a classification problem.

Second step is about selection of evaluated features. This involves the transformation of the input tuple I in an equal or smaller tuple I' . In this regard, Sklearn

provides a wide range of feature selection methods, based on the statistical distribution of the respective scores. Since for this specific problem the number of input feature is relatively small, we decided to select the H features with best scoring values, where H is a manually fixed parameter. The algorithm also provides a manual selection of the specific features, in order to test the performance of the model created with different inputs.

4.3.12 Model estimation

After horizontal data selection, in order to extract useful data for the calculation of each different model, and vertical selection of most relevant features, the three models are calculated. As specified in section 4.1, a generic model can be expressed as a function $f_{A_1 \dots A_o}(X_1, \dots, X_n, W_1, \dots, W_m)$. This phase of the ML process aims to select, once fixed a specific structure for function f , the best parameters A_o and weights W_m for mapping input tuple (X_1, \dots, X_n) , that is our I , to a specific target in O . Parameters A_o of each of the three models are estimated with an external validation process. Weights W_m estimation is instead the effective model training process, and is different for windows model, which is a classical classification problem, and occupancy models, which are instead regression problems.

Validation

For validation purposes a k-fold cross-validation procedure has been adopted, together with a brute force technique of cycling on all possible value of parameters. Let's suppose our model needs a certain number O of parameters to be fixed. For each parameter we select an associated discrete finite domain D_o of a certain number L_o of possible values. For each possible combination of indices (L_1, \dots, L_o) , correspondent values tuple $(v_{1L_1}, \dots, v_{oL_o})$ is fixed as model parameters (A_1, \dots, A_o) and cross-validation procedure is performed, dividing data into K folds and taking them one by one as a validation set. Every iteration returns an evaluation of its performance on data not used for model creation. These values are collected and tuple $(v_{1L_1}, \dots, v_{oL_o})$ generating best evaluation results is finally selected (fig. 4.11).

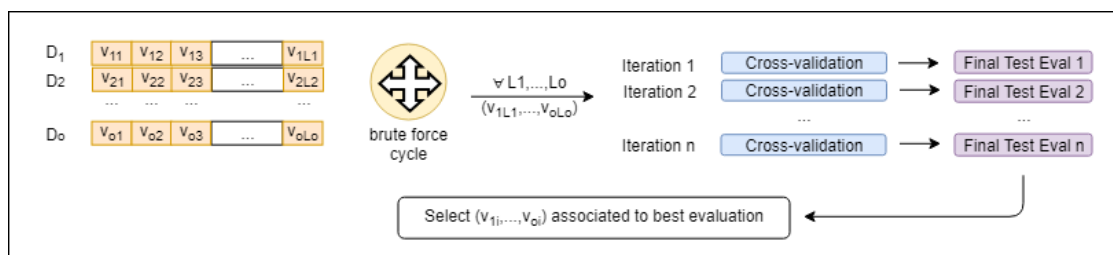


Figure 4.11: Validation procedure schema

Windows model

Windows model is called f_w and aims to predict the windows opening state W starting from the input tuple I' . W is a binary target tuple, its domain is $\{0,1\}$. It

is so a classical binary classification problem. Due to the simplicity of the problem, and to the low number of features available, two rather simple algorithms were used to build the model: logistic regression and decision trees.

Logistic Regression (LR), despite its name, is a linear model for classification. It was born as a method to represent a binary variable, but it also adapts easily to the multiclass case. The membership of a sample X to the specific target class $y=1$ is represented in terms of probability. The training of this model is in fact based on weights (w_j) estimation of a logistic probability function, regarding the probability to belong to one class rather than another, which is in the form:

$$P(X) = \frac{1}{1 + e^{-\sum_{j=1}^P w_j x_j}} \quad (4.8)$$

where P is the number of input feature. This formula is a combination between a linear model in X and a logistic function. In case of a single input feature it usually takes a form similar to that in figure 4.12. A threshold can then be set, with respect to the probability that a sample must reach to be classified as positive. This threshold is a classical validation parameter for this algorithm. As the number of input features increase, the logistic regression model tends to overfit training data. Some regularization is therefore necessary. Sklearn offers the possibility of exploiting regularization of type L1 (Ridge) and L2 (Lasso). A more detailed analysis of this type of regularization is in next section.

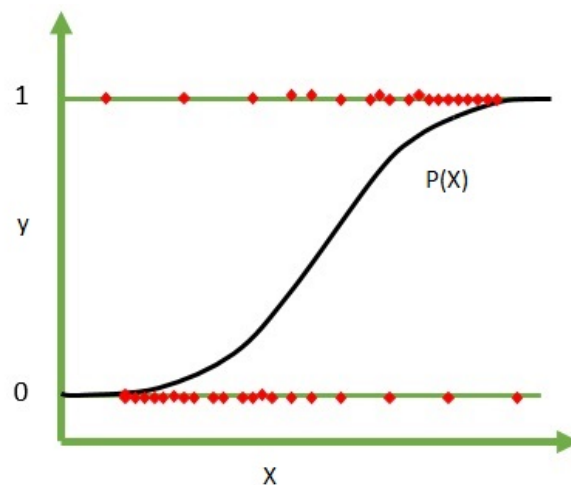


Figure 4.12: Logistic Regression with a single input feature

LR is also extensible to a multiclass case, for example through the One-Vs-Rest (OVR) paradigm. It provides that a different model is created for each class, considering it as positive and the all other class as negative. In order to classify a new input tuple X each model is so evaluated, giving a probability of belonging to each class, and X is finally classified as the most probable class. The main advantage of LR is that is very easy to implement and efficient to train. Moreover it returns conditional probabilities which can be very useful for certain analysis. Anyway, since it is a generalized linear model, its hypothesis space is limited accordingly

and in some cases may not be appropriate.

Decision Trees (DT) are instead a supervised learning method used both for classification and for regression. In particular, in the classification case, the goal is to create a model that predicts the value of a discrete target variable by learning simple subsequent decision rules inferred from data features. Each rule can be represented as split in a tree node, based on the value of a particular feature (fig. 4.13). Leaf nodes represent instead the predicted class. In the training process it is important to set a specific criteria to measure the quality of a split, in order to let the learning algorithm to select the best split. In Sklearn library, in particular, Gini Index and Information Gain metrics are available. Another important parameter to be set is the maximum depth of the tree. In fact, if tree built is too deep, model is much more complex and tends to overfit training data.

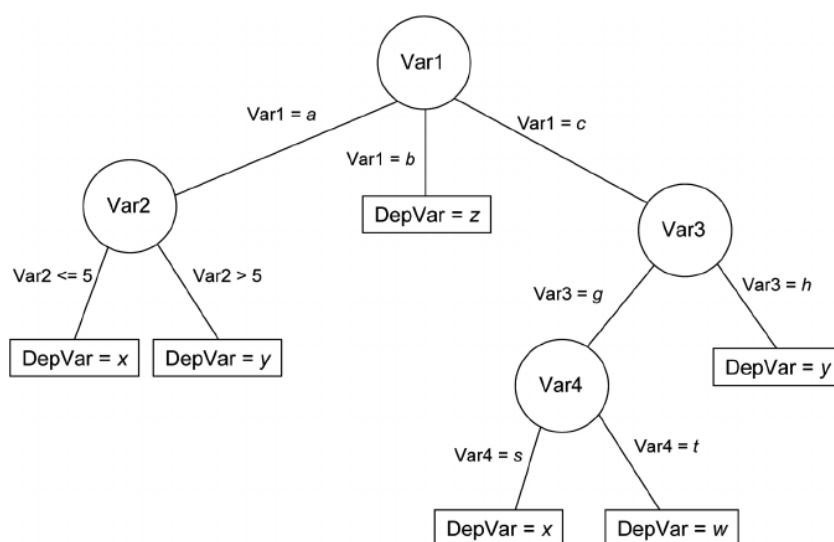


Figure 4.13: Example of a generic Decision Tree

DT are simple to interpret and can be planarly visualized. As LR they do not require data normalization. Another important advantage is that they can adapt to that situations in which LR failed. The main disadvantage is the tendency to create too complex models (overfitting), which can only be solved through pruning operations. Moreover, finding the optimal tree is a very complex problem. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. A second solution to avoid overfitting in decision trees is the usage of the corresponding ensemble method, the Random Forests (RF) algorithm. RF builds multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing prediction variance. Data for the training of each tree are selected with a bagging technique, which involves a repeated draft of the samples with re-entry. It means that a single sample can be used more than once for the same tree. The final prediction is obtained as the mode of the classes predicted by the individual trees. Number of trees or maximum depth of a tree are classical validation parameters.

Occupancy models

Two different occupancy models must be estimated: the first regards estimating people occupancy number in a situation of closed windows, and is called f_0 , the other is called f_1 and regards estimating room occupation in the most difficult case in which windows are open and clean air is coming in. Input tuple is again I' and target O has a positive integer number domains. Even if domain is not continuous, this problem is better suited to the usage of a regression method, together with an appropriate discretization phase, compared to the usage of a classification method. This is mainly due to the fact that occupancy target is numerical, even if numbers can be seen as classes. For example, if we collect training samples in a room which has always been occupied by zero or two people during the acquisition period, a classification model will never predict one occupant in the room, even if it is an absolutely possible situation. A regression model instead may be able to adapt to target not encountered during data collection. For this purpose Linear Regression (LR) and Polynomial Regression (PR) models have been mainly used, with addition of appropriate regularization techniques.

LR is a classical approach to modelling the relationship between a scalar target variable (dependent variable) and one or more independent variables. In particular, this relationship, is modeled using a linear predictor function, that in case of a number P of input feature is in the form:

$$P(X) = w_0 + \sum_{j=1}^P w_j x_j \quad (4.9)$$

where $W = \{w_0, w_1, \dots, w_p\}$ is the vector of weights to estimate in order to minimize the residual sum of squares between the observed responses in the training dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_W \| Xw - y \|_2^2 \quad (4.10)$$

This method, which results in a linear mapping between input and target variables, can easily be extended to the case in which we want to look for a polynomial relation $P^n(x_1, \dots, x_p)$ of an arbitrary degree N . A weight w_i is associated with each polynomial term. For example, for $P=2$ and $N=2$ the associated polynomial is:

$$P(X) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2 \quad (4.11)$$

Sklearn offers for this purpose a specific function called `PolynomialFeatures`, able to generate for each sample a new feature tuple consisting of all polynomial combinations of the features with degree less than or equal to the specified one. N is a classical parameter to be optimized through validation method. High values of N always correspond to a more complex model, which in many cases could overfit. If $N=1$ polynomial regression coincide with a linear regression. In figure 4.14 is shown an example of a polynomial regression on generic data.

As in the logistic regression case, two regularization methods are available to prevent overfitting. Polynomial to train remains the same, but changes the objective

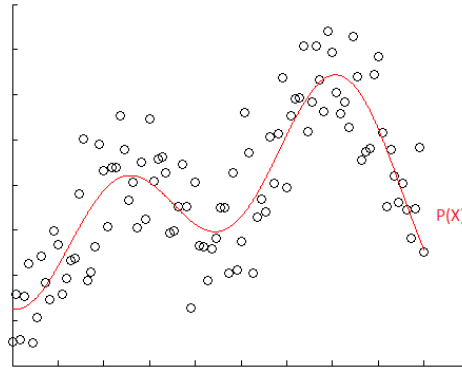


Figure 4.14: Example of a generic polynomial regression

function for weight estimation. Overfitting is generally associated with large estimated values for w_i , which results in a function with large oscillation, very adherent to data used for its creation. Both L1 and L2 regularization methods aim therefore to penalize functions with high weights. L1 regularization is also called Lasso and its objective function has this form:

$$\min_W \| Xw - y \|_2^2 + \alpha \| w \|_1 \quad (4.12)$$

where α is a validation parameter which refers to the amplitude of the regularization. If $\alpha=0$ Lasso is equivalent to a classical linear regression.

L2 regularization is instead called Ridge and its objective function has this form:

$$\min_W \| Xw - y \|_2^2 + \alpha \| w \|_2^2 \quad (4.13)$$

Also in this case α refers to the regularization strength. The choice of a regularization method rather than the other depends on the specific application context. What mainly changes is the assumption that is made on the class of linear transformation they infer to relate input and output data. In L2 the coefficients of the linear transformation are normal distributed. In L1 are instead Laplace distributed. Lasso is therefore more likely to estimate zero coefficients and therefore to automatically perform a feature selection operation. Ridge regularization instead tends to reduce the variance of weight distribution.

The regression methods, presented up to this point, necessarily need a discretization phase, since their output is in a continuous domain (section 4.3.13). Also adaptation of the classification algorithm DT and RF to the regression case have been used for this purpose. For the binary case instead, in which occupancy is expressed in term of a vacant or occupied room state, the same classification algorithms used to predict the status of the windows were used.

Models aggregation

Models f_w , f_0 and f_1 are then re-aggregated in a unique model M used for final prediction and evaluation. Aggregation takes place through equation 4.14.

$$M = (1 - f_w) \times f_0 + f_w \times f_1 \quad (4.14)$$

It simply means that if model f_w predicts a closed windows status, model f_0 is used for occupancy prediction. If f_w predict an open windows status, f_1 is instead used. If the estimate regarding windows status is precise, this makes it possible to use a more precise model for estimating the final occupation. The f_0 and f_1 models tend in fact to be very different from each other.

4.3.13 Prediction

The step which follows the creation of the model, in order to be able to evaluate its accuracy, is the prediction on test data. Input data I , taken from test dataset, are given as inputs to the aggregated model M , and output data are stored in the tuple O' . Another prediction tuple is created from training set in the case in which learning curve mode has been selected. Since required target is in the domain of integer numbers, and regression models output are decimal numbers, a discretization phase is required on tuple O' . For this purpose, each sample is simply approximated to the nearest integer.

4.3.14 Evaluation metrics

Last step of the whole ML process is the evaluation of results. Evaluation means to establish how well the prediction tuple O' represents the true test values present in the tuple O . Different metrics are available for this purpose and are analyzed below. These methods are also used for the internal validation.

Confusion matrix

Confusion Matrix (CM) is an important starting point for all evaluation metrics. It is a specific table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances of the predicted tuple O' , while each column represents the instances of the real values tuple O . Number at each table intersection (o'_i, o_j) represents the number of predicted value of type i corresponding to real values of type j . The higher the numbers gather on the diagonal of the matrix, the more the performance of the model were good. CM takes on a particular meaning in the binary case, or in the case in which each class is evaluated against all others. In this case (fig. 4.15) table has just two rows and two columns, and reports the number of false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN).

Accuracy

Accuracy (ACC) is the simplest evaluation metric. It is defined as the percentage of correct classifications on total number of classified elements. In the binary case it is calculated as:

		Predicted class	
		<i>P</i>	<i>N</i>
Actual class	<i>P</i>	True Positives (TP)	False Negatives (FN)
	<i>N</i>	False Positives (FP)	True Negatives (TN)

Figure 4.15: Binary confusion matrix

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.15)$$

This concept can be easily extended to the multiclass case. Defined $diag(M)$ as the operator that selects all elements on the diagonal of the matrix M , accuracy can be expressed as:

$$ACC = \frac{sum(diag(CM))}{sum(CM)} \quad (4.16)$$

For the specific problem of this work, in which it is difficult, or in some cases even not interesting, to reach a precise estimate on the effective numbers of occupants, a different version of accuracy can be considered. ACC_{tol-t} , where t is a positive integer number, is defined as a classical accuracy in which a predicted value o' , corresponding to a real value o , is considered correct if in the interval $[o-i, o+i]$. This allows to calculate an accuracy value for each relevant tolerance band. If $t=0$, ACC_{tol-t} corresponds to a standard accuracy. However, accuracy is generally not a reliable evaluation metric for a classifier, because it will yield misleading results if dataset target is unbalanced (that is, when the numbers of observations in different classes vary greatly).

Binary indicators

If a binary CM is calculated, some metrics such as Precision, Recall and F1 score can be defined.

Precision is defined as:

$$prec = \frac{TP}{TP + FP} \quad (4.17)$$

It is a measure about how many samples classified as positive are actually positive. Recall is defined as:

$$rec = \frac{TP}{TP + FN} \quad (4.18)$$

It is a measure about how many positive items are actually classified as positive. F1 score is the harmonic average of the precision and recall. In fact it is defined as:

$$F1 = \frac{2 \times prec \times rec}{prec + rec} \quad (4.19)$$

It is an alternative measure to classic accuracy, able to provide relevant information even in the case of classification with unbalanced classes. F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. This measure is interesting also in the case in which CM is the confusion matrix of a class against all other classes. In fact, if calculated for all predicted classes, it is possible to visualize information about classes in which model M performs better.

Mean squared error

Mean Squared Error (MSE) is another important metric for evaluation, especially for regression problems. It measures the average of the squares of the errors between real and predicted values. If vectors O' and O are both made up of a number Q of elements, MSE is defined as:

$$MSE = \frac{1}{Q} \times \sum_{i=1}^Q (o_i - o'_i)^2 \quad (4.20)$$

In order to better understand which is the effective error that our model is expected to predict, Root Mean Squared Error (RMSE) can be defined:

$$RMSE = \sqrt{MSE} \quad (4.21)$$

If applied to occupancy problem, RMSE represents the amount of people that model M will tend to underestimate or overestimate on average.

Chapter 5

Experiments and results

This chapter is a detailed analysis about how methods described above performed in two selected scenarios. Two dataset were created, collecting data in the two corresponding classrooms, and different models have been built and evaluated for both of them. First part is a description of the main features of each scenario. In the second part obtained results are presented and discussed.

5.1 Scenarios

The two rooms have both been selected within the Politecnico di Milano, and differ in various features such as size, number of windows, number of regulars, etc.. A summary of the characteristics of each space is shown in table 5.1.

	AntLAB small	AntLAB big
Dimensions	3x8 mt	8x10 mt
Seats	8	25
Windows	1	3
Doors	1	2
Sensors (In/Out)	3/1	3/1
Ground truth	SmartGate + manual correction	Manual + image capture

Table 5.1: Scenarios properties

5.1.1 AntLAB small

From December 2018 to February 2019 AntLAB was moved due to work in building 20, and fragmented into 3 smaller classrooms at the third floor of building 21. One of these small room has been used for experiments. In figure 5.1 is shown a map of the interested space. It is a 10x8 meters room, that communicates with the corridor through an entrance door and with the external through a single window.

It is provided with 6 tables and has 8 seats. Three sensors have been positioned inside, in order to better cover the entire room, and one sensor have been positioned outside in the corridor. This allowed us to create a representative feature of the Co2 difference between the corridor and the room, useful for understanding the internal quantity with respect to an external baseline. Raspberry (and the sensor connected to it) has been positioned so as to be as central as possible and cover all other three sensors within its WiFi range. Ground truth has been collected through a SmartGate, positioned at the entrance door. Even if it has shown, in indoor situations, to have an almost maximum accuracy, people inside were advised to periodically check and, if necessary, manually adjust the measurement. This was possible via a mobile device connected directly to the Raspberry PI wireless network, also useful for windows ground truth collection. Sensors were originally battery powered, but they were subsequently wired to a power outlet due to sudden discharge problems (fig. 5.2). The small size of this room and the precision with which the dataset was labeled were two fundamental factors for the results presented in the next section.

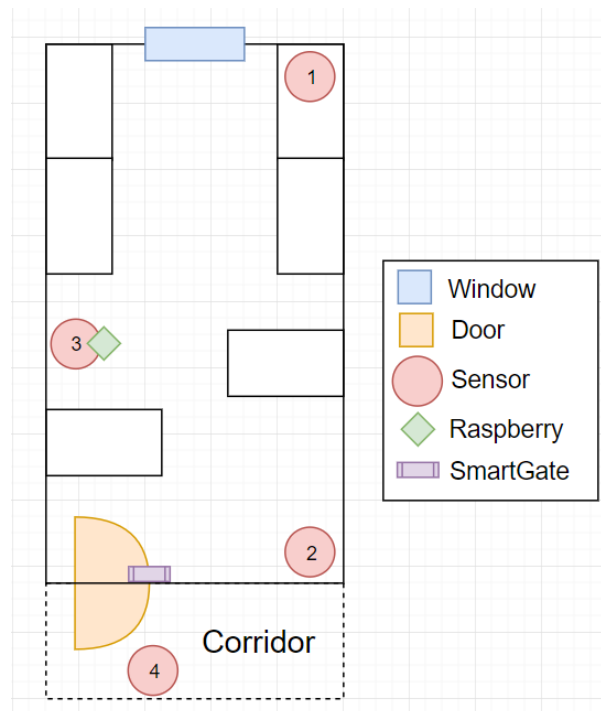


Figure 5.1: AntLAB small map

5.1.2 AntLAB big

This is the classic laboratory office, on the ground floor of building 20. A map of the laboratory is shown in figure 5.4. It is over 3 times larger than the small AntLAB, its dimension in meters is 8x10. It communicates through a door with the main corridor, and through another door to a secondary hall, which will not be taken into account for the occupancy count. This secondary room in turn communicates with the outside through a dedicated door. It was taken as baseline, placing a sensor

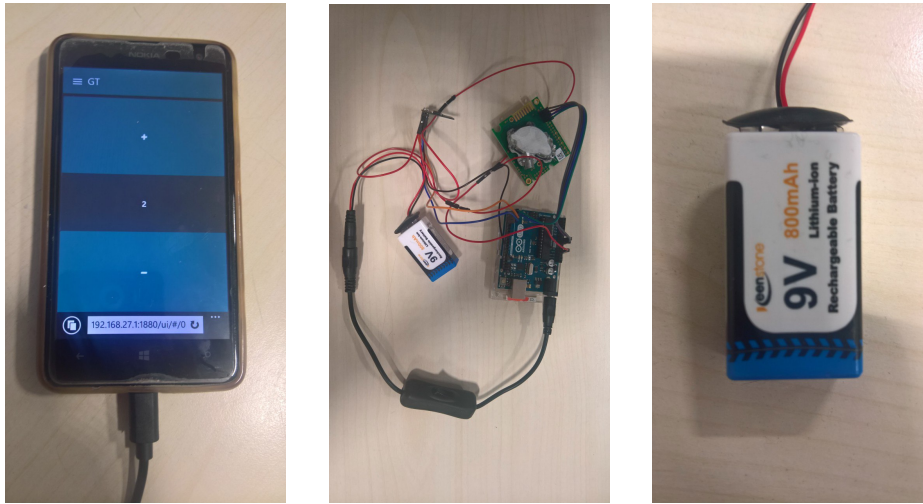


Figure 5.2: Ground truth mobile device and sensor powered from battery

inside it and calculating the difference with respect to the main room average. The laboratory also contains 23 big tables, for a number of about 25 seats which are rarely used all at the same time, and has 3 windows. Raspberry and sensors were positioned with a criterion similar to that of the small laboratory, powered by a wall socket. Due to the presence of a double door it was not possible to use SmartGate for ground truth collection. It was collected manually through fixed and mobile devices connected to the laboratory network. For moments when none of those present could take care of data collection, the camera, wired to Raspberry and placed in a corner of the room, was enabled to take photos periodically. Photos were then checked by hand and dataset was fully labeled (fig. 5.3). This larger room was a good example of the worst performances in the precise estimate that model can have in the event that the environment is more spacious and the number of people is on average greater. Some complications have been encountered due to the fact that AntLAB people used to have launch in the secondary small room. At a certain time it was therefore common for everyone to move together from the main to the secondary room.

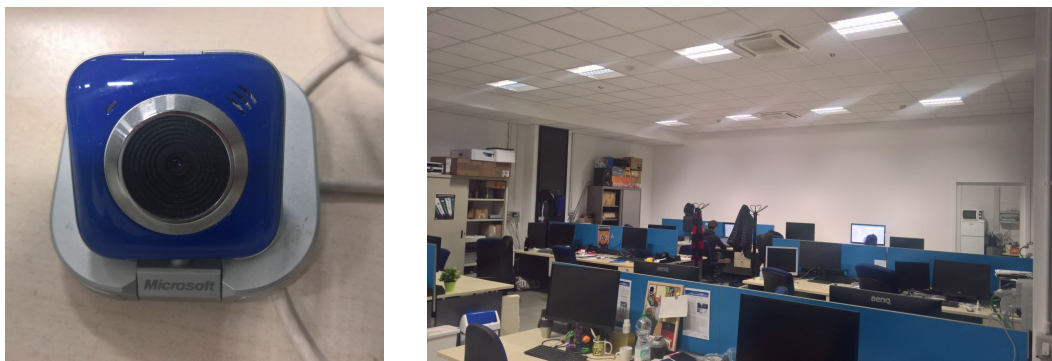


Figure 5.3: Video camera used and an example of photo taken through it

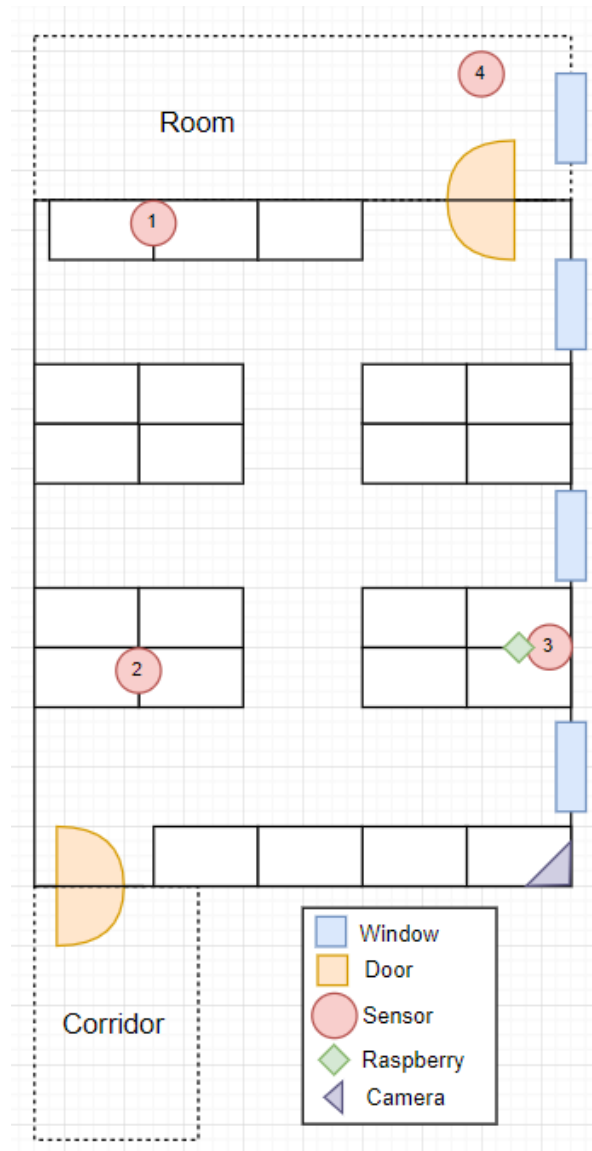


Figure 5.4: AntLAB big map

5.2 Results

5.2.1 Datasets description and pre-process baseline

In both scenarios data are acquired at a frequency of 0.1Hz, that is, a sample is saved in the database every 10 seconds. Among all the acquisitions, the nine most accurate from each of the two rooms have been selected for model construction and evaluation. In both scenarios a sensor was positioned outside the area of interest, while the other 3 were placed inside. Therefore an aggregation of data coming from the 3 internal sensors will always be carried out, while data from external sensor will always be used for the calculation of an extra feature, which is the difference between internal average and external Co2 concentration ($Co2_{in-out}$). Figure 5.5 is an example of aggregation, with a standard mean method, and computation of the extra $Co2_{in-out}$ feature, applied to the first file of the AntLAB small dataset.

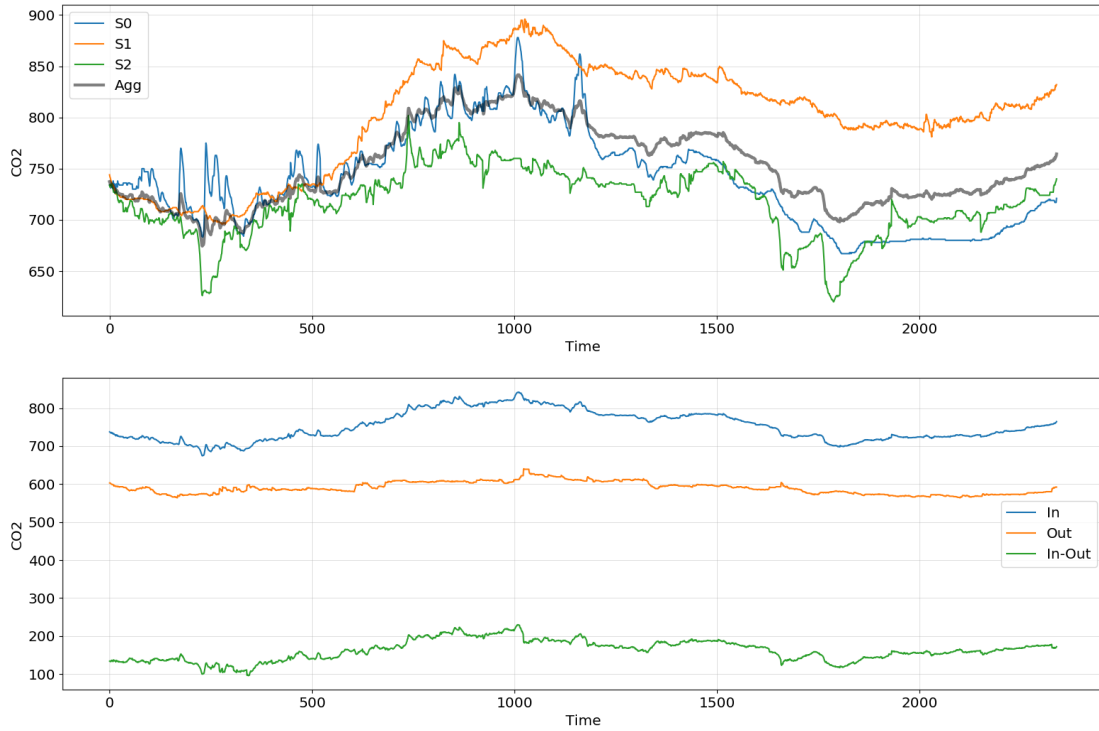


Figure 5.5: Example of aggregation and in-out feature computation

In table 5.2, instead, the overall characteristics of both datasets are summarized. For this purpose are considered only internal data, aggregated with standard mean method, without resampling and without filtering.

	AntLAB small	AntLAB big
Number of file	9	9
Total duration	26h 45min	42h
Number of samples	16058	25182
Windows open samples	269 (1.7%)	945 (3.7%)
Zero samples	4925 (30.7%)	8434 (33.5%)
Min-Max Co2 values	415ppm - 1004ppm	409ppm - 1399ppm
Min-Max people values	0 - 8	0 - 20
Number of nights	1	2

Table 5.2: Datasets properties

Let us now introduce resampling and filtering of data. Baseline data resampling factor is fixed at 12 for all acquisition. It means that frequency is reduced from 0.1Hz to 0.0083Hz, equivalent to a sample each 2 minutes. A baseline Butterworth second order filter has also been designed, with a normalized cutoff frequency of

0.30 (fig. 5.6). An example of resulting series for Co_2 and $\text{Co}_{2_{\text{in-out}}}$ features, on the same acquisition mentioned above, are shown in figure 5.7.

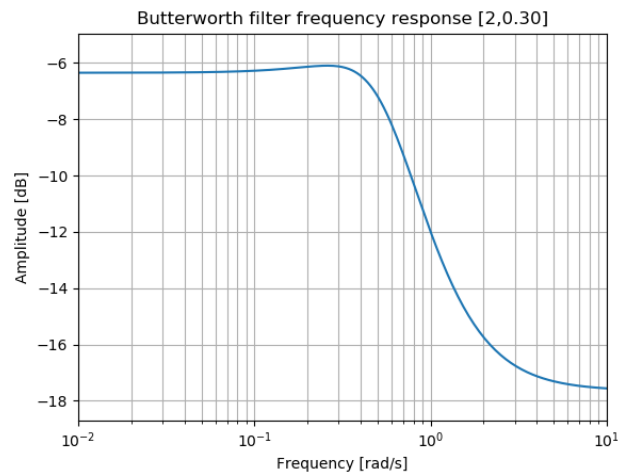


Figure 5.6: Frequency response of a second order Butterworth filter with normalized cutoff frequency 0.30

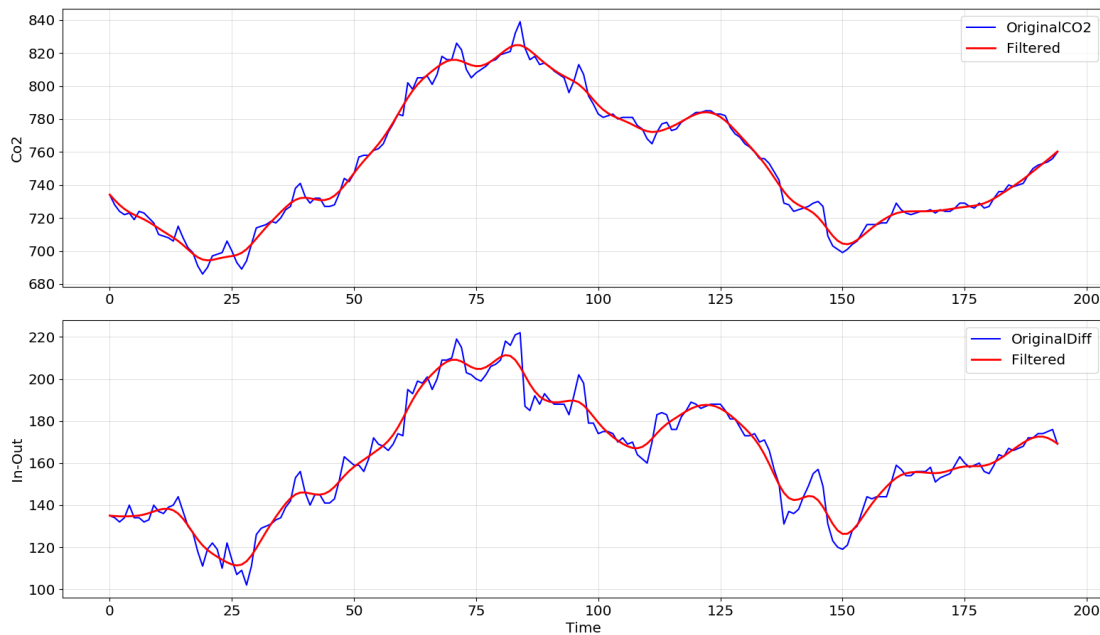


Figure 5.7: Example of Butterworth filter application

With this kind of simple data pre-process some interesting plots have been generated to better understand data distribution. Figure 5.8 and figure 5.9 show 2 histogram of Co_2 and people data distribution. Figures 5.10, 5.11 and 5.12 show respectively Co_2 absolute value, Co_2 first order gradient and time data in a scatter plot, with related number of people in the vertical axis. In these type of plots dimension and color of a point represent both its numerosity in the whole dataset: a larger size and a warmer color represent a more frequent sample. This allows to

highlight the type of relationship that exists between features and target variables. For this purpose time has been divided into 24 classes.

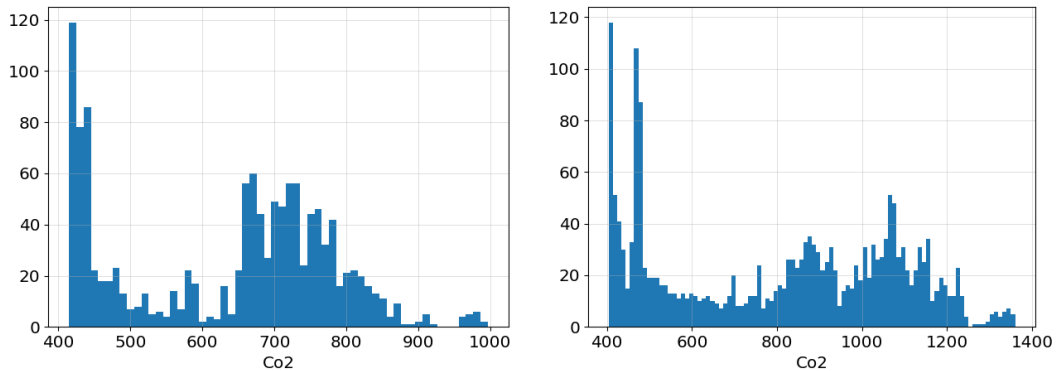


Figure 5.8: Co2 samples distribution (on the left AntSmall, on the right AntBig)

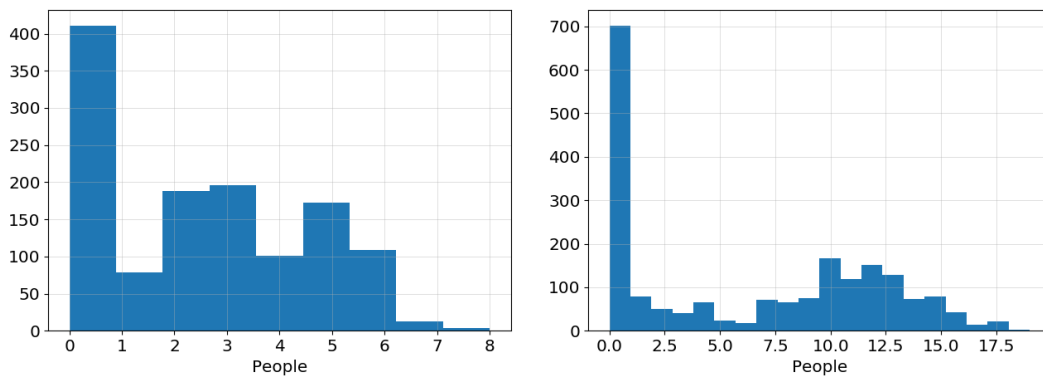


Figure 5.9: People samples distribution (on the left AntSmall, on the right AntBig)

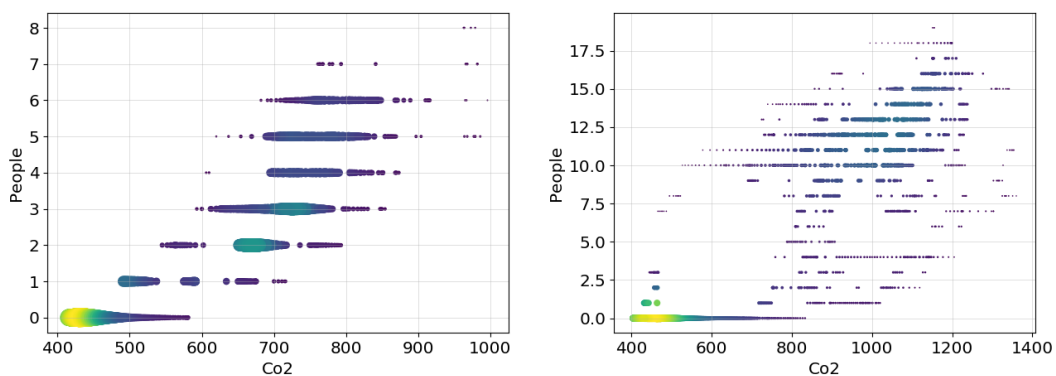


Figure 5.10: Co2-People scatter (on the left AntSmall, on the right AntBig)

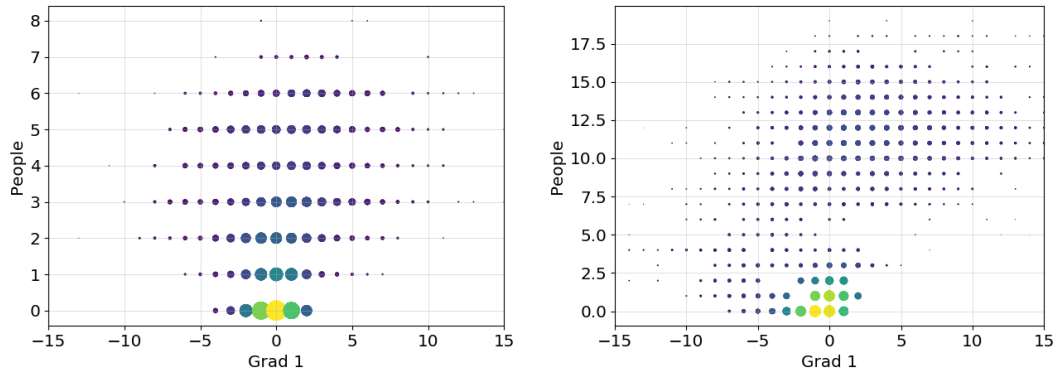


Figure 5.11: Gradient-People scatter (on the left AntSmall, on the right AntBig)

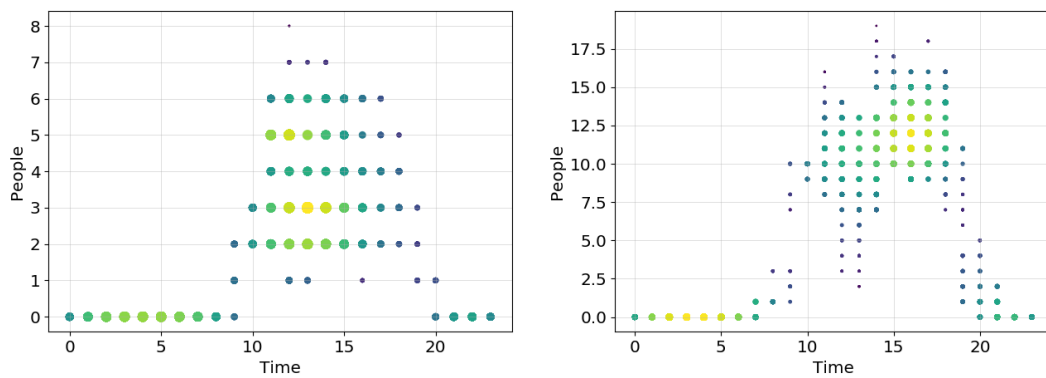


Figure 5.12: Time-People scatter (on the left AntSmall, on the right AntBig)

5.2.2 Baseline model

In this section calculation and evaluation of two basic models are presented, one for each scenario. These models will be taken as a baseline for evaluating improvements or deterioration of performances due to the change of some parameters. Building process complies with the following conditions:

1. Data are resampled so as to have 1 sample every 2 minutes (resample factor equal to 12).
2. Each series is filtered with a second order Butterworth with normalized cutoff frequency set at 0.30.
3. No time-lag is considered.
4. Samples in which windows are open are deleted from dataset.
5. Only data from a single sensor (sensor 1) are used.
6. A 5-fold cross-validation is performed.

7. Absolute value of Co2 is the only feature used for prediction.
8. A simple regression of a second degree polynomial is performed. Limiting the degree of the polynomial is in itself sufficient to avoid overfitting.

In table 5.3 are summarized performance of this execution while in the following figures are graphically represented some important features of these models. Figure 5.13 represents the already discretized model with a continuous line, and test data with points, in order to underline how models will perform on new samples. Figure 5.14 highlights the relation between real and predicted test data, also in this case indicating points numerosity with color and size. The more points are present on the red diagonals, the more model built has good performance. Figure 5.15 represents real train data (blue), real test data (green) and predicted test data (orange) on a temporal line. All these figures refer to the first cross-validation iteration. It can be seen how in general the model on the small room performs better, and how the one on the larger room tends to make wrong prediction even for the zero class. This happens mainly in the first of the two nights.

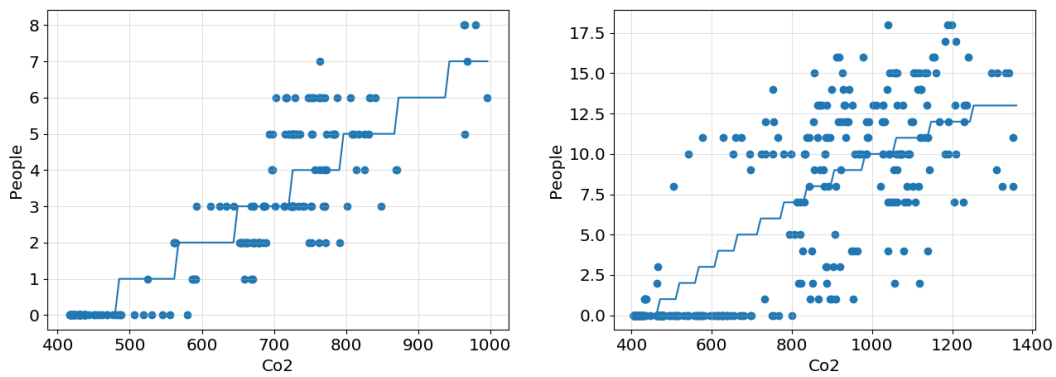


Figure 5.13: Baseline discretized model (on the left AntSmall, on the right AntBig)

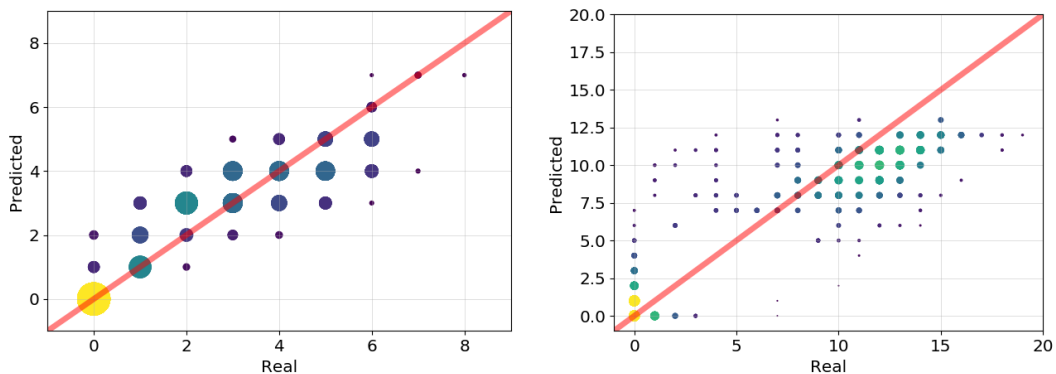


Figure 5.14: Baseline model real vs predicted on test data (on the left AntSmall, on the right AntBig)

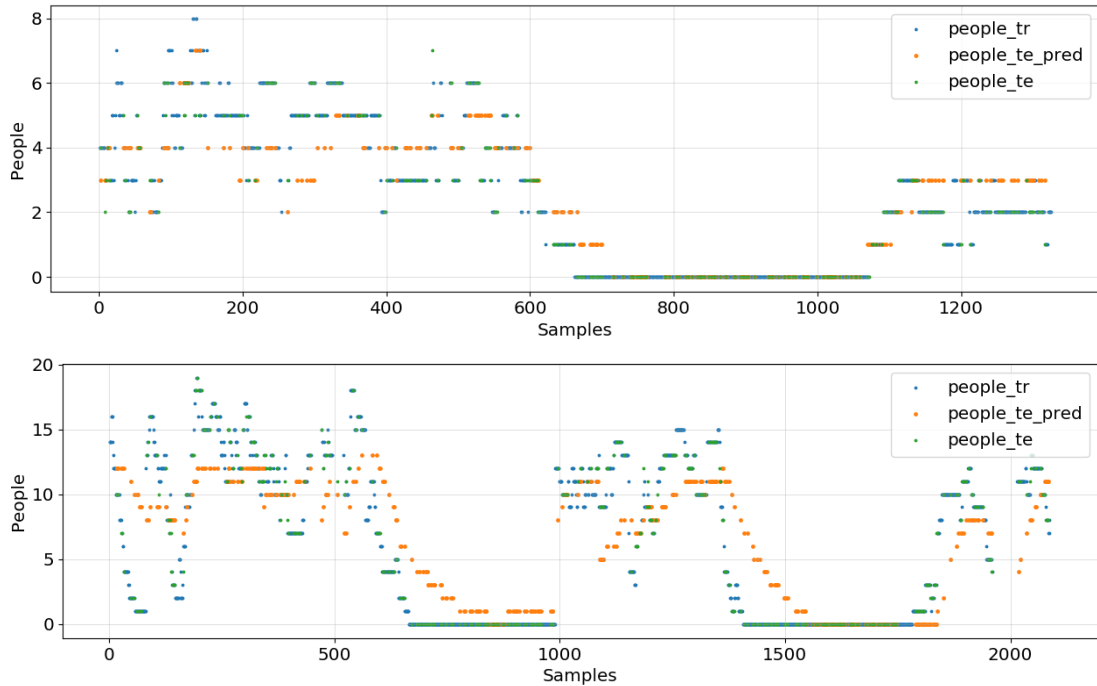


Figure 5.15: Baseline model people train, people test, people test predicted temporal series (above AntSmall, under AntBig)

	AntLAB small	AntLAB big
Accuracy 0-tol	41.25%	17.99%
Accuracy 1-tol	83.45%	40.68%
Accuracy 3-tol	100%	64.87%
Accuracy 5-tol	100%	84.17%
RMSE	1.10	3.69

Table 5.3: Baseline models results

5.2.3 Multi-sensor variation

First improvement is obtained by modifying condition 5 of baseline model. From this point on the feature regarding the internal Co2 value will always be an aggregate of all 3 internal sensors. All average, median and penalize gap method have been tried, obtaining very similar results. Small difference in performance between an execution and another seems to be caused only by randomization of data selection in the cross-validation process. However, there is an improvement in performance compared to the single sensor case, as summarized in table 5.4, partly solving the problem of wrong predictions on first night in AntBIG scenario.

	AntLAB small	AntLAB big
Accuracy 0-tol	44.58%	20.97%
Accuracy 1-tol	85.34%	42.07%
Accuracy 3-tol	100%	66.95%
Accuracy 5-tol	100%	86.42%
RMSE	1.05	3.53

Table 5.4: Multi-sensors regression models results

5.2.4 Multi-feature variation

This improvement is obtained by modifying condition 7 and so taking into consideration also the other calculated features, such as time, first order gradient and $\text{Co2}_{\text{in-out}}$. First of all a feature evaluation analysis was done and results are summarized in table 5.5 for AntLAB small and in table 5.6 for AntLAB big.

	F-score	P-value	MI
Co2	2936	$1e^{-292}$	0.98
Gradient	44.17	$5e^{-11}$	0.21
Time	92	$1e^{-20}$	0.88
$\text{Co2}_{\text{in-out}}$	3398	$1e^{-292}$	0.96

Table 5.5: AntLAB small occupancy feature evaluation

	F-score	P-value	MI
Co2	2091	$1e^{-270}$	0.97
Gradient	289.64	$1e^{-58}$	0.44
Time	203.29	$7e^{-43}$	1.11
$\text{Co2}_{\text{in-out}}$	3.92	0.047	0.85

Table 5.6: AntLAB big occupancy feature evaluation

Whereas F-score and P-value are linear correlation indices, while MI include also other kind of dependence, these tables lead to the following considerations:

- In small AntLAB Co2 and $\text{Co2}_{\text{in-out}}$ have a linear correlation much higher than the other two features.
- In small AntLAB time become a relevant feature when considering also non linear dependencies.

- In big AntLAB, gradient is more relevant from a linear point of view.
- In big AntLAB $\text{Co2}_{\text{in-out}}$ is not so relevant, probably due to the approach of people at lunch time to the external sensor.
- Also in big AntLAB time becomes relevant when considering non-linear dependencies. In this case it becomes even the most important feature.

Best performances were however obtained with the use of all features. In both scenarios the exclusion of the less relevant features, ie the gradient, has always led to slightly worse results. Performances are summarized in table 5.7. In this case improvements with respect to the single feature multi-sensor model are remarkable.

	AntLAB small	AntLAB big
Accuracy 0-tol	54.87%	34.81%
Accuracy 1-tol	89.87%	63.38%
Accuracy 3-tol	100%	92.30%
Accuracy 5-tol	100%	98.94%
RMSE	0.87	1.88

Table 5.7: Multi-feature regression models results

Improvements are visible also looking at figure 5.16 and 5.17. In particular, thanks to the time feature, AntLAB big problem of wrong predictions in the first night is almost solved. Moreover, in the real vs predicted plot, points get very close to the diagonal, significantly lowering the average error.

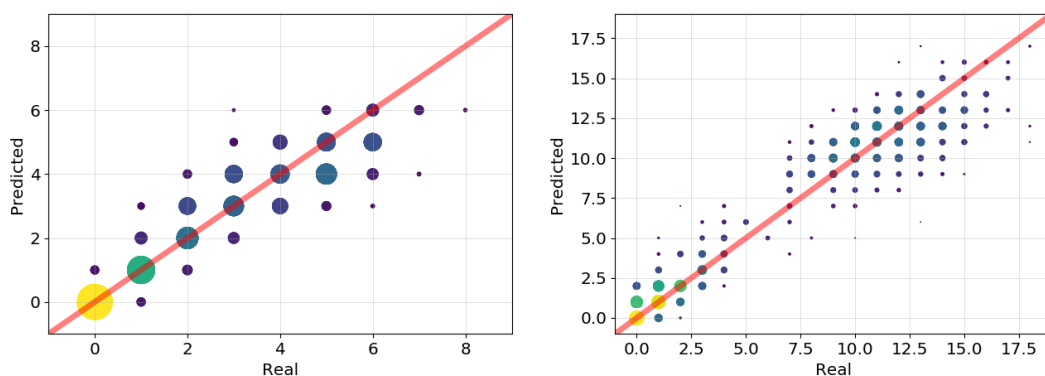


Figure 5.16: Multi-feature model real vs predicted on test data (on the left AntSmall, on the right AntBig)

Due to the impossibility of plotting 4-dimensional models, in order to visualize the actual contribution of gradient to the prediction, a model with only Co2 and gradient features was built and plotted in its discretized version in figure 5.18.

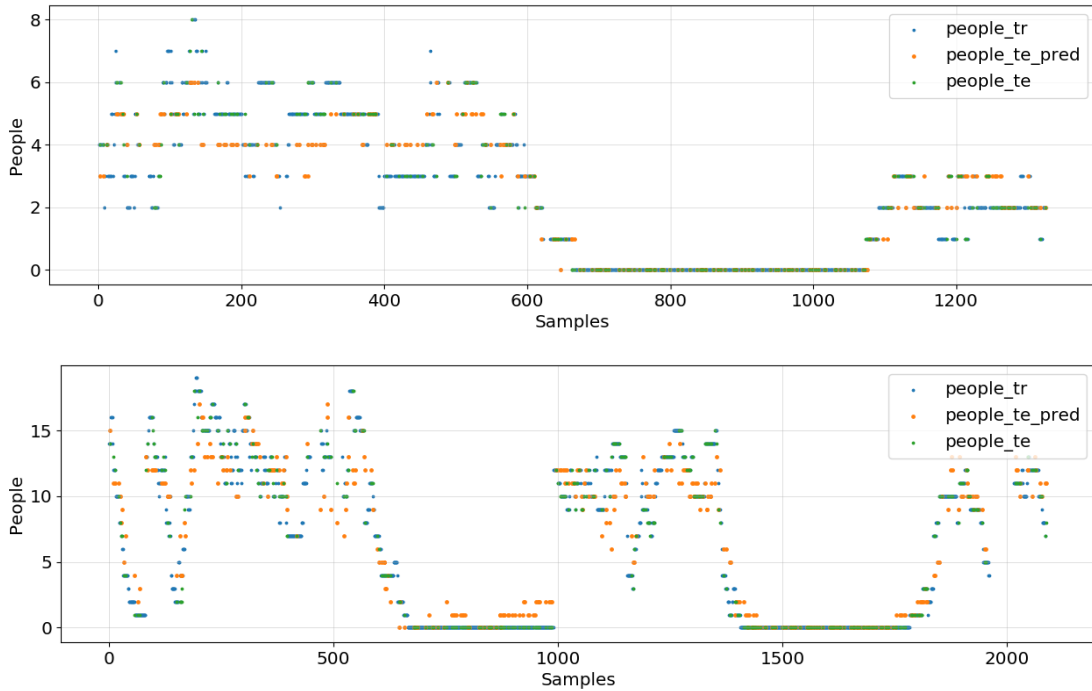


Figure 5.17: Multi-feature model people train, people test, people test predicted temporal series (above AntSmall, under AntBig)

From this plot it is evident how different Co2 variation, at the same Co2 absolute internal value, affects prediction differently. In particular, at a fixed Co2 value, a higher gradient is often sufficient to predict more people. Moreover, gradient is more influential in cases of low absolute Co2 concentration, since it tends to be zero when air tends to Co2 saturation. The model built with just these two features got anyway good evaluations, far better than the single feature one.

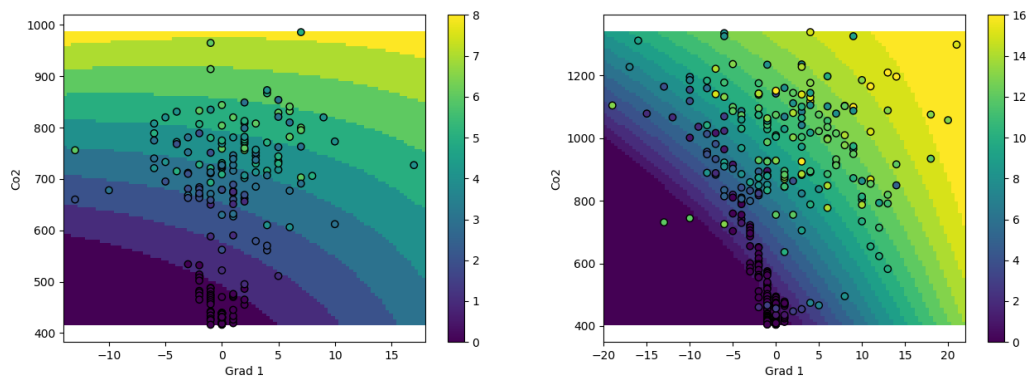


Figure 5.18: Co2-Gradient regression discretized model (on the left AntSmall, on the right AntBig)

5.2.5 Model variation

Different kind of models are evaluated in this section, modifying assumption number 7 of the baseline model. Here models are built with all sensors and all features, comparison is therefore with what presented in section 5.2.3.

A first consideration is about changing polynomial degree, which until now had been set to 2. A polynomial of first degree results in both cases in a decrease of performances, due to the fact that relationships between input and target variables, as expected, are not linear. An increasing in the polynomial degree, instead, in some cases leads to a slight improvement in performance. However, even if evaluated on data not used in weights estimation phase, using high degrees seems to create overfitting models. This is due to the fact that the creation of a complex model is excessive for this kind of problem, and models tend to fit too much to the specific situation in which data were collected.

In order to better understand overfitting problems, L1 regularization has been applied, with validation on its alpha parameter and with different polynomial degrees. Alpha was forced to assume strictly positive values. Figure 5.19 show a typical trend for MSE with alpha variation, in an iteration of the external cross-validation, generally chosen around 0.15 for both scenarios. In table 5.8 are instead summarized results for a polynomial degree equal to 2, highlighting, looking at RMSE, a slight decrease of performance with respect to the not regularized model.

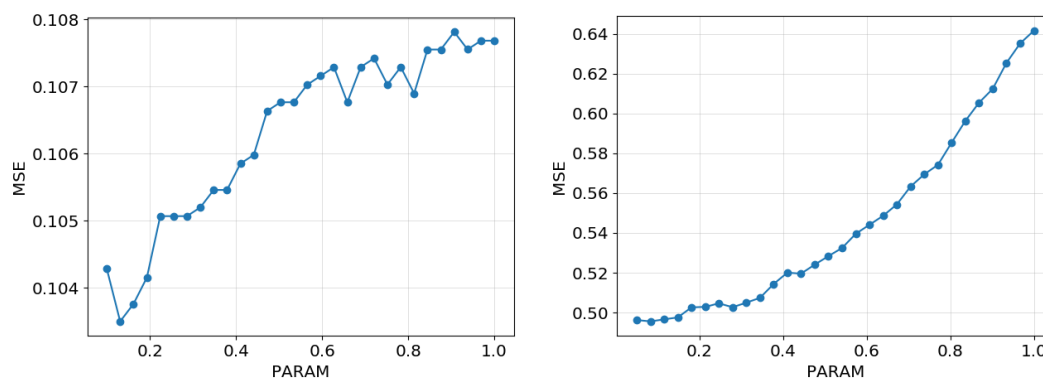


Figure 5.19: Lasso validation on alpha (on the left AntSmall, on the right AntBig)

	AntLAB small	AntLAB big
Accuracy 0-tol	57.62%	36.42%
Accuracy 1-tol	89.72%	64.91%
Accuracy 3-tol	99.92%	91.52%
Accuracy 5-tol	100%	98.14%
RMSE	0.88	1.91

Table 5.8: Lasso models results

As the degree of the polynomial increases, performances do not seem to improve. Degrees of 2,3,4,5 generate roughly the same results. What change seems to be mainly an increase in the value assumed by alpha in the validation phase, corresponding to a greater regularization. Looking at the coefficients we note instead a clear approach to zero of the weights corresponding to the terms of first degree, except for Co2 absolute value, which remains the least regularized feature. This leads to the conclusion that regularization it's not necessary for low polynomial degrees, and that increasing the degree of complexity of the model does not lead to an increase of performance, but only to a greater need of regularization in order to not overfit training data. With the regularization of type L2 very similar results have been obtained.

Other interesting consideration are about DT and RF regressors, when validating on the maximum depth parameter. In both cases and in both scenarios behavior is similar. In figure 5.20 are shown validation performances.

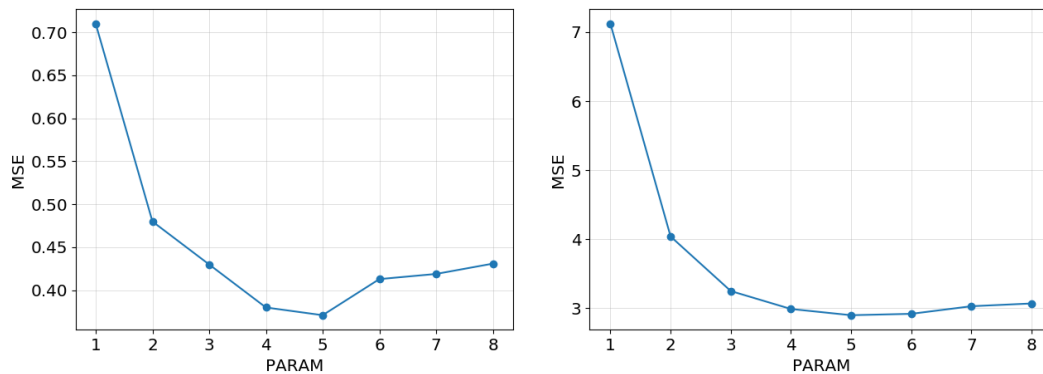


Figure 5.20: Random Forest Regressor validation on max-depth (on the left AntSmall, on the right AntBig)

Results for the case in which max-depth is 3 (table 5.9) are slightly worse than those obtained with standard regression method. When increasing the depth of the trees, MSE seems to be improving a little, to then start to get worse again due to overfitting problems when max-depth is increased too much. For depths equal to 5 in both scenarios performance seems to be slightly better than those presented in table 5.7. However this kind of models seems to be too complex to represent the problem in question, and as in the case of the polynomial regression tend to overfit situations in which data were collected. In order to visualize graphically this phenomenon, in figure 5.21 are plotted RF regression models, for the small scenario, in the case of only Co2 absolute value and gradient taken as input features and max-depth is set respectively to 3 and 5. These models can be compared directly with the left one in figure 5.18. Here appears clear as the division of the space typically generated by trees is not very suitable to represent this kind of problem, and also how deeper trees tend to generate too complex models.

	AntLAB small	AntLAB big
Accuracy 0-tol	53.91%	43.97%
Accuracy 1-tol	85.45%	64.29%
Accuracy 3-tol	100%	87.67%
Accuracy 5-tol	100%	96.68%
RMSE	0.93	2.19

Table 5.9: Random Forest Regressor models results

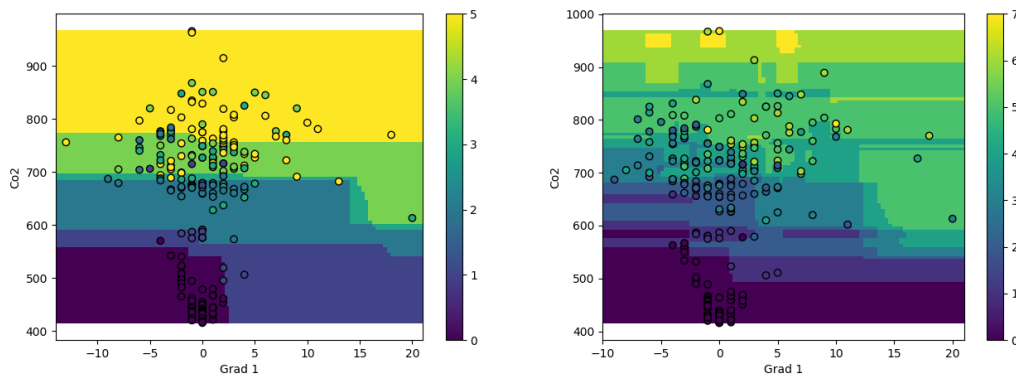


Figure 5.21: Co2-Gradient Random Forest Regressor model for AntSmall (on the left max-depth = 3, on the right max-depth = 5)

5.2.6 Windows

For evaluating the complete model, also making predictions about windows state, in this section condition 4 of the baseline model is modified and windows open data are re-added to the dataset. For people estimation is always considered a multi-feature second degree polynomial. An operation of feature evaluation was performed on windows binary target, resulting for both scenarios in a clear prevalence of the gradient feature, followed by absolute Co2 value. In any case, all features were used to build the model. For example, time is useful to distinguish the nocturnal samples, from those in which the windows have been open for a long time, as both cases involve an approximately zero gradient and an absolute low amount of Co2. Performance on people prediction are summarized in table 5.10 for AntLAB small and in table 5.11 for AntLAB big. Baseline model suffers of a considerable performance decrease, due to the introduction of open windows data, which alter the model considerably. Both for predicting windows model with a second degree polynomial logistic regression, validating on probability threshold, and for predicting windows model with a regression tree, validating on max-depth (limited to a maximum of 3 to avoid overfitting), performance seems to improve. Although the performance improvement is not so considerable in an absolute sense, considering the small percentage of data with open windows present in both datasets, this improvement is sufficient to prove the validity of this method. Moreover, in both scenarios, regression performs better than decision trees.

	Baseline	W LogReg	W RegTree
Accuracy 0-tol	50.17%	51.80%	51.56%
Accuracy 1-tol	87.68%	88.72%	88.38%
Accuracy 3-tol	99.88%	99.96%	99.92%
Accuracy 5-tol	100.00%	99.97%	100%
RMSE	0.99	0.95	0.96

Table 5.10: Models results in AntLAB small considering different methods for windows prediction

	Baseline	W LogReg	W RegTree
Accuracy 0-tol	28.08%	30.65%	30.13%
Accuracy 1-tol	51.99%	55.54%	55.06%
Accuracy 3-tol	80.42%	82.56%	81.63%
Accuracy 5-tol	94.31%	95.14%	94.56%
RMSE	2.72	2.60	2.68

Table 5.11: Models results in AntLAB big considering different methods for windows prediction

Figure 5.22 and 5.23 show respectively the models built with the two respective methods, applied to both scenarios with only gradient and Co2 features, in order to allow a planar view. These models are all taken from the first cross-validation iteration and highlight how regression is more suitable for this application, since input variables are continuous and trees tend to divide the space in a discrete way. Optimal threshold parameter were 0.22 for AntLAB small and 0.34 for AntLAB big, while trees has always chosen a max-depth of 3. Looking at performances of the binary windows prediction only, evaluated on test data with a model built with a polynomial logistic regression, you can see how predictions are better in the small scenario, compared to the big one (table 5.12).

	AntLAB small	AntLAB big
Accuracy	99.6%	95%
F-score	0.93	0.79

Table 5.12: Windows prediction evaluation on test data

Finally, note that the model which predict occupancy in the case of open windows, has always had very bad performance, comparable to a random prediction. This is because the effect of the incoming carbon dioxide is much more relevant than that of human expiration in determining the amount of internal carbon dioxide.

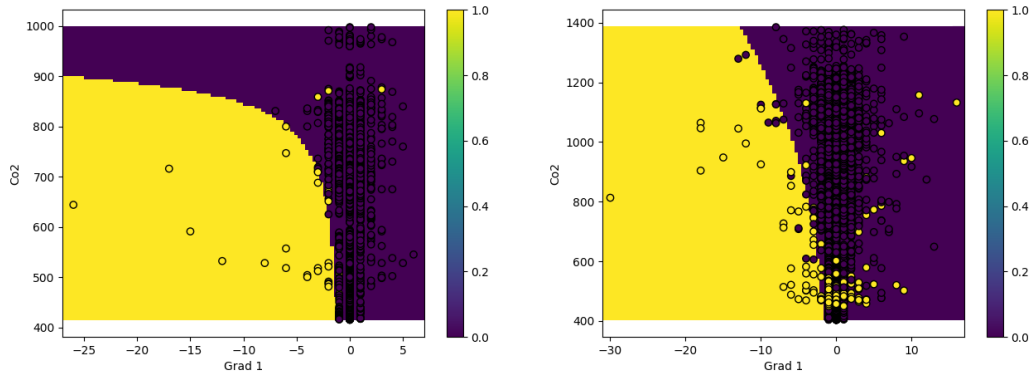


Figure 5.22: Windows logistic regression model (on the left AntSmall, on the right AntBig)

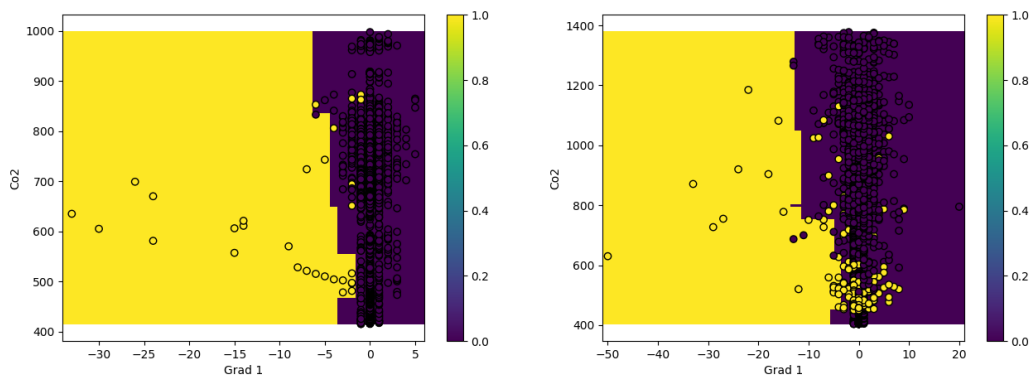


Figure 5.23: Windows classifier tree model (on the left AntSmall, on the right AntBig)

5.2.7 Time lag

This section contains an analysis regarding the search for the delay intrinsically present in data, modifying condition number 3 of the baseline model. Time-lag, expressed in number of samples, is closely related to the re-sample frequency. Considering the frequency used up to now, of a sample every 2 minutes, and an upper bound set to 5 samples of delay, equivalent to a maximum delay of 10 minutes, in both scenario optimal time-lag is null. Figure 5.24 is a plot of NRMSE progress with respect to the number of data shifts, respectively in the small and in the big rooms, taken from one of the 9 files as example. A second analysis was carried out by raising the sampling frequency to a sample every 30 seconds, and appropriately moving the upper bound to 20 samples, in order to discover delays under 2 minutes. In both scenarios best delay took on average a value of 4-5 sample (fig. 5.25), confirming an average delay just over 2 minutes, regardless of the size of the room.

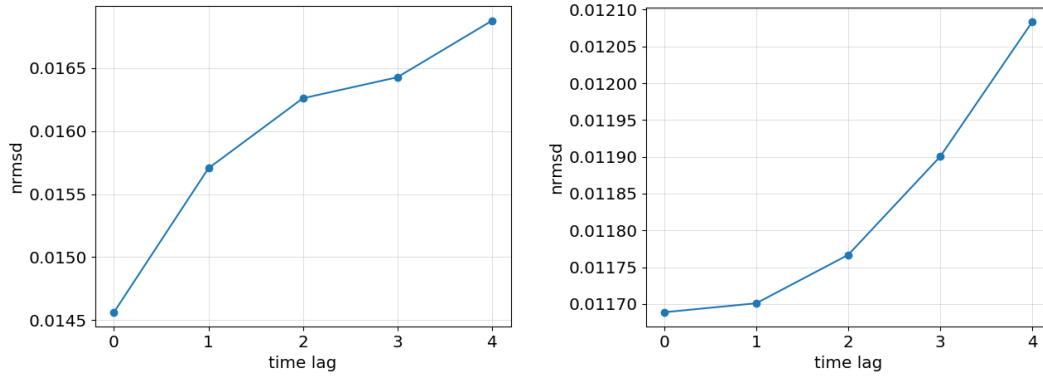


Figure 5.24: *Nrmse vs time-lag with a sample every 2 minutes (on the left AntSmall, on the right AntBig)*

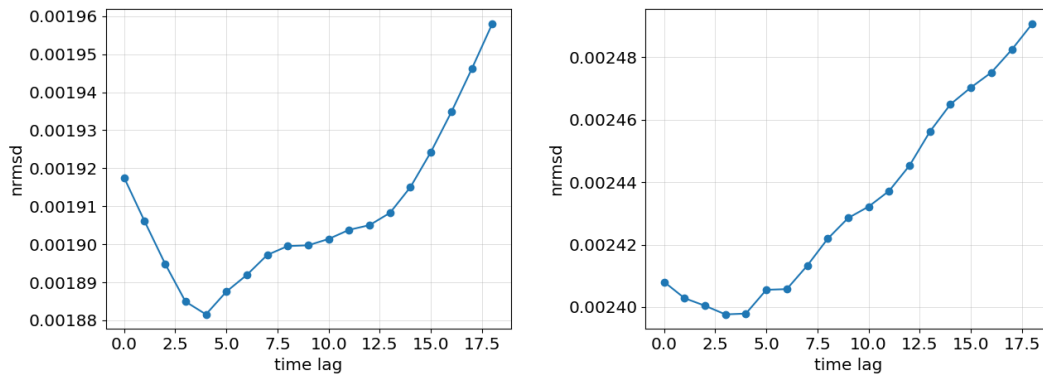


Figure 5.25: *Nrmse vs time-lag with a sample every 30 seconds (on the left AntSmall, on the right AntBig)*

5.2.8 Resample and filter

Last parameters that can be changed to analyze the performance of the model are the resample factor (condition 1 of the baseline model) and the filter normalized cutoff frequency, which determines the amount of filtering (condition 2 of the baseline model). RMSE results are summarized in tables 5.13 and 5.14, always referring to a second degree, multi-sensor, multi-feature polynomial regression, which is the optimal baseline. In the case of the AntLAB small this analysis shows that changing the frequency of the data or changing filter features only leads to worse predictions. This is not true for AntLAB big scenario, in which decrease the frequency of the samples up to 1 every 3 and a half minutes leads to a considerable improvement, and this is true also with a lower normalized cutoff frequency, up to 0.05. This is probably due to a greater amount of noise present in the data collected in the second scenario, which reflects in a greater need of filtering and averaging data.

Cutoff freq	AntLAB small	AntLAB big
3	0.93	2.36
6	0.90	2.18
12 (opt baseline)	0.87	1.88
20	0.88	1.75
30	0.91	1.83

Table 5.13: RMSE evaluation with change of resample frequency

Resample factor	AntLAB small	AntLAB big
No filter	0.92	2.25
0.8	0.89	2.19
0.3 (opt baseline)	0.87	1.88
0.2	0.90	1.85
0.05	0.93	1.84
0.005	0.95	2.46

Table 5.14: RMSE evaluation with change of filter normalized cutoff frequency

5.2.9 Learning curves

Figure 5.26 shows train and test RMSE learning curves for both scenarios, when the optimal model is applied.

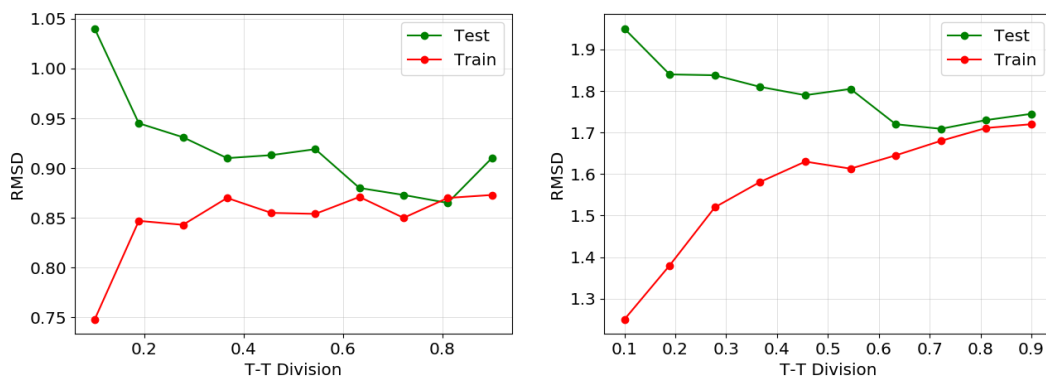


Figure 5.26: RMSE train and test learning curve for optimal models (on the left AntSmall, on the right AntBig)

In both cases the test curves show a good tendency of the model to improve prediction performances as the number of data with which it is created increases, with a slight deterioration when the data with which it is tested becomes less than 10%. Also a clear deterioration in the performance on train data is visible, demonstrating a tendency of the model to not overfit data with which it is built.

5.2.10 Binary case

In this section are analyzed performance improvements when target class is not the number of occupants, but is a binary variable regarding the vacant/occupied state of the room. A first analysis regards data distribution. Looking at each individual feature, data seems to be very related to the binary target variable. Most related features are Co2 (fig. 5.27) and $\text{Co2}_{\text{in-out}}$, followed by time (fig. 5.28). Also in this case the gradient seems to be the least correlated feature (fig. 5.29).

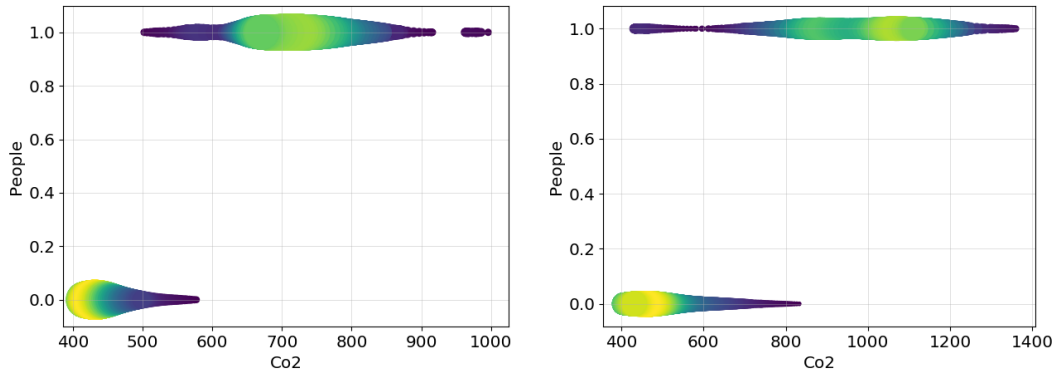


Figure 5.27: Co2-BinaryState scatter (on the left AntSmall, on the right AntBig)

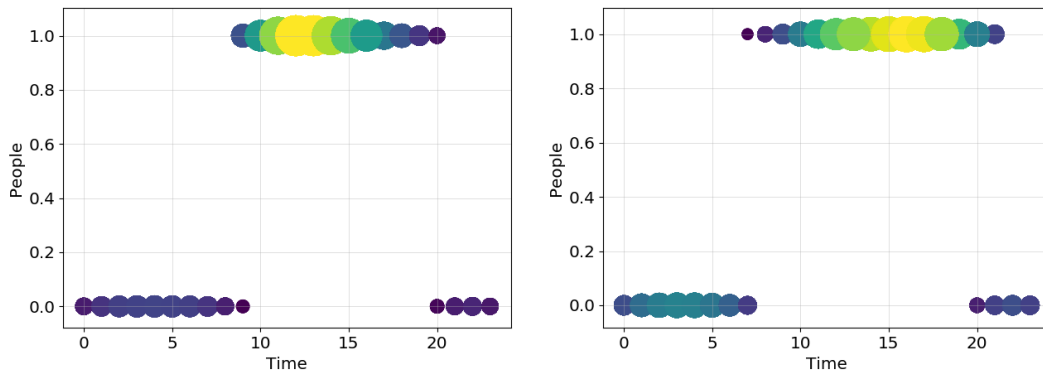


Figure 5.28: Time-BinaryState scatter (on the left AntSmall, on the right AntBig)

The high correlation between features and target leads to good prediction performance. As for the windows case have been applied logistic regression and prediction trees models, getting very good similar results. Also in this case regression models seem to be more suited to the problem, since the input variables are continuous. In figure 5.30 are showed confusion matrices referred to the application of a second degree logistic regression on all available features, with validation on the probability threshold. Results in term of accuracy and f-score are instead in table 5.15. Again, better results were obtained in the smaller room.

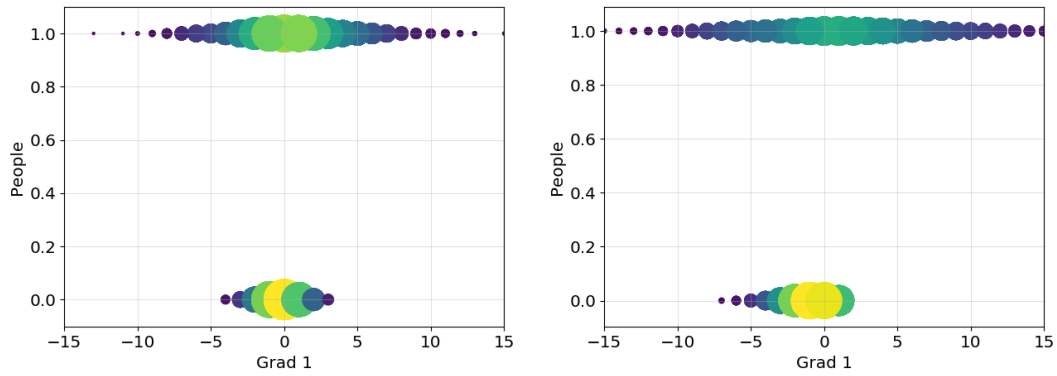


Figure 5.29: Gradient-BinaryState scatter (on the left AntSmall, on the right AntBig)

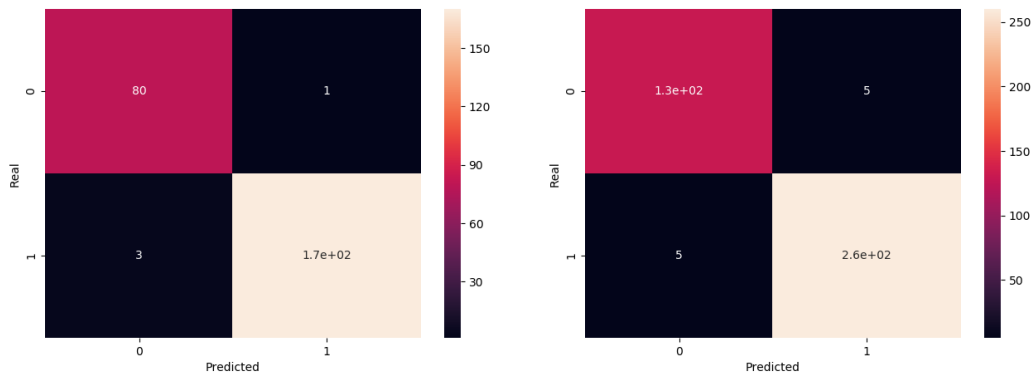


Figure 5.30: Confusion matrix for binary prediction (on the left AntSmall, on the right AntBig)

	AntLAB small	AntLAB big
Accuracy	98%	97.5%
F-score	0.989	0.981

Table 5.15: Binary prediction evaluation on test data

5.2.11 Mixing scenarios

As last case of analysis, the two datasets concerning the two respective scenarios have been mixed, in order to test the possibility of creating a universal model as independent as possible from the context of application. For this purpose all 18 acquisition files were concatenated, making a unique big dataset. Optimal model was applied, obtaining the results shown in table 5.16. In figure 5.31 is also shown the disposition of predictions in a real/predicted scatter plot. From this plot is visible how the new model tends to underestimate the occupation of the largest room. Looking at RMSE, however, worsening is of an acceptable order of magnitude.

Mixed	
Accuracy 0-tol	33.49%
Accuracy 1-tol	61.97%
Accuracy 3-tol	87.87%
Accuracy 5-tol	97.73%
RMSE	2.193

Table 5.16: Models results in a mixed scenario

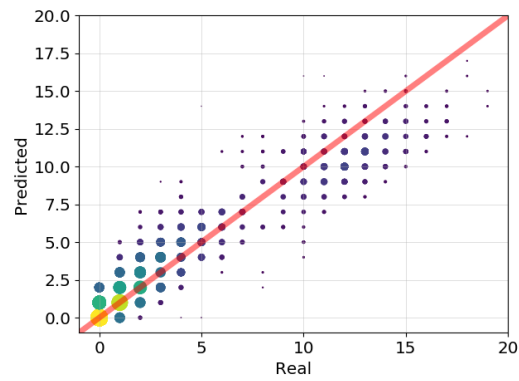


Figure 5.31: Mixed scenario real vs predicted on test data (on the left AntSmall, on the right AntBig)

Chapter 6

Conclusions and future works

This project started with a unique main goal: to make a detailed study about the possibility of estimating the number of occupants in a room, starting from a non-intrusive environment variable, ie the concentration of carbon dioxide. This theme was then developed in two steps, trying to satisfy two secondary objectives:

- Develop a multi-sensor distributed IoT system, able to collect Co2 concentration data related to different areas of the same room, containing as much as possible the total cost of used devices.
- Study the performance of different machine learning methods, pre-processing collected data and training different models with different parameters.

Both objectives have been pursued trying to improve what already exists in the literature. In the following are some considerations about advantages and limitations of used methods.

As regards the hardware part, the designed system proved to be reliable and accurate over the long term. Moreover, the independence of the system from other networks allows it to adapt to different situations. The proposed ground truth collection methods have proven to be quite effective, with a simple and intuitive graphical interface. The main limitation of a system of this type is the accuracy of the sensor. In order to obtain good results it is essential to use sensors that guarantee data quality, which are generally quite expensive. The main cost of the project was therefore in the purchase of the 4 carbon dioxide sensors. However, in real applications, Co2 sensors are generally already integrated into ventilation or air conditioning system of the room in question.

Compared with methods already used in literature, the complex pipeline of our machine learning study has been shown to create more precise models. The main contributions to this improvement are mainly due to the aggregation of data coming from various sensors, to a careful pre-processing phase and to the insertion of new features such as time, gradient and the difference of internal concentration compared to that external one. An added value of this project is given by the fact that the proposed method is independent from the assumption of closed windows, giving a precise prediction of the moments in which the windows are opened, and by the fact that the model can also be trained through many small separate acquisitions, achieving equally good performance. This study has also shown that even

with simple models good performances can be achieved, since the dependence between the variables does not go beyond the second degree and too complex models always lead to an overfitting situation. The main limitations for achieving higher accuracy are essentially due to the poor correlation between variables and the unpredictability of some events, like the stationary presence of a person near a sensor or sudden infiltrations of clean air in the room.

Based on these considerations and on obtained results, the following works are proposed to carry out this study:

- Development of a system able to acquire data from other environmental sensors (such as light, noise, etc.), as well as carbon dioxide data, also integrated with a wireless packet sniffing system. Adding these new features to a process similar to the one proposed in this work would certainly lead to an improvement in predictive performance, solving the main problems related to the usage of Co2 concentration only.
- Implementation of an efficient prediction model, able to real-time predict the occupancy state of the room. In particular, in the smart campus project, this estimate could be uploaded to the university website or could be made available via a chatbot, so that both students and managers can have a complete knowledge of the state of occupation of the entire campus.
- Carry out the analysis regarding the possibility of creating a universally valid model, which can adapt to rooms of different sizes and with different characteristics. In this regard a generic basic model could be created, configurable in some basic parameters used as prediction features (such as the size of the room), or able to adapt to different situations through a second short training phase with a small labeled dataset from the new room.

Bibliography

- [1] Real-time human detection in computer part 1. <https://medium.com/@madhawavidanapathirana/https-medium-commadhawavidaapathirana-real-time-human-detection-in-computervision-part-1-2acb851f4e55>. Last accessed: 2019-02-20.
- [2] Real-time human detection in computer part 2. <https://medium.com/@madhawavidanapathirana/real-time-human-detection-in-computer-vision-part-2-c7eda27115c6>. Last accessed: 2019-02-20.
- [3] Michael Abbott-Jard, Harpal Shah, and Ashish Bhaskar. Empirical evaluation of bluetooth and wifi scanning for road transport. In *Australasian Transport Research Forum (ATRF), 36th*, 2013.
- [4] Daniele Uboldi Andrea Ganassa. A multi-sensor approach to people counting and flow estimation in smart campus. Master’s thesis, Politecnico di Milano, 2018.
- [5] Irvan Bastian Arief Ang, Flora Dilys Salim, and Margaret Hamilton. Human occupancy recognition with multivariate ambient sensors. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6. IEEE, 2016.
- [6] Irvan B Arief-Ang, Flora D Salim, and Margaret Hamilton. Cd-hoc: Indoor human occupancy counting using carbon dioxide sensor data. *arXiv preprint arXiv:1706.05286*, 2017.
- [7] Irvan B Arief-Ang, Flora D Salim, and Margaret Hamilton. Da-hoc: Semi-supervised domain adaptation for room occupancy prediction using co 2 sensor data. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments*, page 1. ACM, 2017.
- [8] RC Arora. Comfort-physiological principles, iaq and design conditions. *Refrigeration and Air Conditioning*, pages 819–871, 2010.
- [9] Bharathan Balaji, Jian Xu, Anthony Nwokafor, Rajesh Gupta, and Yuvraj Agarwal. Sentinel: occupancy based hvac actuation using existing wifi infrastructure within commercial buildings. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 17. ACM, 2013.

- [10] Chandrayee Basu, Christian Koehler, Kamalika Das, and Anind K Dey. Perccs: person-count from carbon dioxide using sparse non-negative matrix factorization. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 987–998. ACM, 2015.
- [11] Piotr Batog and Marek Badura. Dynamic of changes in carbon dioxide concentration in bedrooms. *Procedia Engineering*, 57:175 – 182, 2013. Modern Building Materials, Structures and Techniques.
- [12] Stephen Butterworth. On the theory of filter amplifiers. *Wireless Engineer*, 7(6):536–541, 1930.
- [13] Luis M Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, 112:28–39, 2016.
- [14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE Computer Society, 2005.
- [15] Saandeep Depatla, Arjun Muralidharan, and Yasamin Mostofi. Occupancy estimation using only wifi power measurements. *IEEE Journal on Selected Areas in Communications*, 33(7):1381–1393, 2015.
- [16] Simone Di Domenico, Mauro De Sanctis, Ernestina Cianca, Paolo Colucci, and Giuseppe Bianchi. Lte-based passive device-free crowd density estimation. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.
- [17] Luigi Granato, Amit Brandes, Carlo Bruni, Aldo V Greco, and Geltrude Mingrone. Vo2, vco2 and rq in a respiratory chamber: Accurate estimation based on a new mathematical model using the kalman-bucy method. *Journal of Applied Physiology*, 2004.
- [18] Ebenezer Hailemariam, Rhys Goldstein, Ramtin Attar, and Azam Khan. Real-time occupancy detection using decision trees with multiple sensor types. In *Proceedings of the 2011 Symposium on Simulation for Architecture and Urban Design*, pages 141–148. Society for Computer Simulation International, 2011.
- [19] Marcus Handte, Muhammad Umer Iqbal, Stephan Wagner, Wolfgang Apolinarski, Pedro José Marrón, Eva Maria Muñoz Navarro, Santiago Martinez, Sara Izquierdo Barthelemy, and Mario González Fernández. Crowd density estimation for public transport vehicles. In *EDBT/ICDT Workshops*, pages 315–322, 2014.
- [20] Kazuhiko Hashimoto, Katsuya Morinaka, Nobuyuki Yoshiike, Chjihiro Kawaguchi, and Satoshi Matsueda. People count system using multi-sensing application. In *Proceedings of International Solid State Sensors and Actuators Conference (Transducers' 97)*, volume 2, pages 1291–1294. IEEE, 1997.

-
- [21] Jin He and Anish Arora. A regression-based radar-mote system for people counting. In *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 95–102. IEEE, 2014.
- [22] James Howard and William Hoff. Forecasting building occupancy using sensor network data. In *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 87–94. ACM, 2013.
- [23] Judith Jennings, Nesrin Colak, and Francis Rubinstein. Occupancy and time-based lighting controls in open offices. *Journal of the Illuminating Engineering Society*, 31(2):86–100, 2002.
- [24] Aftab Khan, James Nicholson, Sebastian Mellor, Daniel Jackson, Karim Ladha, Cassim Ladha, Jon Hand, Joseph Clarke, Patrick Olivier, and Thomas Plötz. Occupancy monitoring using environmental & context sensors and a hierarchical analysis framework. 2014.
- [25] Christian Koehler, Brian D Ziebart, Jennifer Mankoff, and Anind K Dey. Therml: occupancy prediction for thermostat control. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 103–112. ACM, 2013.
- [26] Khee Poh Lam, Michael Höynck, Bing Dong, Burton Andrews, Yun-Shang Chiou, Rui Zhang, Diego Benitez, Joonho Choi, et al. Occupancy detection through an extensive environmental sensor network in an open-plan office building. *IBPSA Building Simulation*, 145:1452–1459, 2009.
- [27] Thomas Lang, Hans-Dieter Wiemhöfer, and Wolfgang Göpel. Carbonate based co2 sensors with high performance. *Sensors and Actuators B: Chemical*, 34(1-3):383–387, 1996.
- [28] Sunil Mamidi, Yu-Han Chang, and Rajiv Maheswaran. Improving building energy efficiency with a network of sensing, learning and prediction agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 45–52. International Foundation for Autonomous Agents and Multiagent Systems, 2012.
- [29] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997.
- [30] E. Longo M. Cesana P. Galluzzi, A. Redondi. Occupancy estimation using low-cost wi-fi sniffers. IEEE BalkanCom, June 2018.
- [31] Luis Pérez-Lombard, José Ortiz, and Christine Pout. A review on buildings energy consumption information. *Energy and buildings*, 40(3):394–398, 2008.
- [32] Steffen Petersen, Theis Heidmann Pedersen, Kasper Ubbe Nielsen, and Michael Dahl Knudsen. Establishing an image-based ground truth for validation of sensor data-based room occupancy detection. *Energy and Buildings*, 130:787–793, 2016.

- [33] A. E. Redondi, M. Cesana, D. M. Weibel, and E. Fitzgerald. Understanding the wifi usage of university students. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 44–49, Sept 2016.
- [34] ABI Research. "more than 30 billion devices will wirelessly connect to the internet of everything in 2020". 2013.
- [35] James Scott, AJ Bernheim Brush, John Krumm, Brian Meyers, Michael Hazas, Stephen Hodges, and Nicolas Villar. Preheat: controlling home heating using occupancy prediction. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 281–290. ACM, 2011.
- [36] Robert Tomastik, Satish Narayanan, Andrzej Banaszuk, and Sean Meyn. Model-based real-time estimation of building occupancy during emergency egress. In *Pedestrian and Evacuation Dynamics 2008*, pages 215–224. Springer, 2010.
- [37] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *null*, page 511. IEEE, 2001.
- [38] F Wahl, M Milenkovic, and O Amft. A green autonomous self-sustaining sensor node for counting people in office environments. In *2012 5th European DSP Education and Research Conference (EDERC)*, pages 203–207. IEEE, 2012.
- [39] Jens Weppner, Paul Lukowicz, Ulf Blanke, and Gerhard Tröster. Participatory bluetooth scans serving as urban crowd probes. *IEEE Sensors Journal*, 14(12):4196–4206, 2014.
- [40] Zheng Yang, Nan Li, Burcin Becerik-Gerber, and Michael Orosz. A multi-sensor based occupancy estimation model for supporting demand driven hvac operations. In *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, page 2. Society for Computer Simulation International, 2012.
- [41] Xucong Zhang, Junjie Yan, Shikun Feng, Zhen Lei, Dong Yi, and Stan Z Li. Water filling: Unsupervised people counting via vertical kinect sensor. In *2012 IEEE ninth international conference on advanced video and signal-based surveillance*, pages 215–220. IEEE, 2012.
- [42] Yang Zhao, Wim Zeiler, Gert Boxem, and Timi Labeodan. Virtual occupancy sensors for real-time occupancy information in buildings. *Building and Environment*, 93:9–20, 2015.