"CO2 based room occupancy detection : an IoT and machine learning application"

Bockstael, Nicolas ; Jadin, Alexandre

ABSTRACT

A CO2 based detection occupancy solution is of great use in smart buildings which automatically control heating, air conditioning,... with help of sensors. An empty building should lower its heating if it detects a pattern of non-occupancy or raise an alarm if it finds an abnormal behaviour. At first, we want to record environmental variables gathered by IoT sensors. Then, build a complete data set of samples composed of co2 ppm, temperature, humidity, light and the actual number of people occupying a room in order to have a solid training set suitable for supervised machine learning. We will then use machine learning algorithms to retrieve more useful and meaningful information such as deducing the precise number of people in a room based on CO2, temperature, humidity,... Those can then be valuable to reduce energy consumption, find patterns of occupancy in buildings and take actions based on those results. We also want to extend our work to other usages than room occupancy and find similar possible deployments. In this work, we will present how to gather valuable data from IoT sensor, semi-manual systems and a raspberry pi equipped with a camera, how we pre-processed the data and feature engineered the samples. We then evaluate various machine learning techniques and combine them in order to predict the exact room occupancy. We estimate the number of occupants using a hard voting technique combining Random Forest, K-Nearest Neighors and Multi-Layer Perceptron classifiers and regressors. While training the model and testing it against samples from the same room, it can a...

CITE THIS VERSION

UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

CO2 based room occupancy detection

an IoT and machine learning application

Dissertation presented by
Nicolas BOCKSTAEL , Alexandre JADIN

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Pierre SCHAUS

Reader(s)
John AOGA, Ramin SADRE

Academic year 2017-2018

Abstract

A CO2 based detection occupancy solution is of great use in smart buildings which automatically control heating, air conditioning,... with help of sensors. An empty building should lower its heating if it detects a pattern of non-occupancy or raise an alarm if it finds an abnormal behaviour. At first, we want to record environmental variables gathered by IoT sensors. Then, build a complete data set of samples composed of co2 ppm, temperature, humidity, light and the actual number of people occupying a room in order to have a solid training set suitable for supervised machine learning. We will then use machine learning algorithms to retrieve more useful and meaningful information such as deducing the precise number of people in a room based on CO2, temperature, humidity,... Those can then be valuable to reduce energy consumption, find patterns of occupancy in buildings and take actions based on those results. We also want to extend our work to other usages than room occupancy and find similar possible deployments.

In this work, we will present how to gather valuable data from IoT sensor, semi-manual systems and a raspberry pi equipped with a camera, how we pre-processed the data and feature engineered the samples. We then evaluate various machine learning techniques and combine them in order to predict the exact room occupancy. We estimate the number of occupants using a hard voting technique combining `Random Forest`, `K-Nearest Neighors` and `Multi-Layer Perceptron` classifiers and regressors. While training the model and testing it against samples from the same room, it can achieve a strict (with no tolerance) accuracy of 85%. However, we also found that while trained on several rooms and tested on the same set of places, the method can actually keep its accuracy. We also found empirically that the most valuable features for this task are the CO2 level, the temperature, the humidity and the light of a room.

Contents

Acknowledgements

We would like to thank Professor Pierre Schaus, who proposed this master thesis subject and supervised it from its genesis to its conclusion.

We are grateful to John Aoga and Professor Ramin Sadre who directly accepted to be part of the jury of this thesis and to review this work.

We also wish to thank the students, teaching assistants, researchers, professors and visitors who accepted to use the manual information gathering systems implemented and helped to constitute a data set.

Finally, we are grateful to any person that, near or far, contributed somehow to this master thesis.

Introduction

Knowing how many people are in a room can be very interesting. It can be used to optimize heating or ventilation systems, or to help to manage a building (a room used intensively could mean a lack of work space while it might be better to close some rooms if they are little used). There are several techniques to count the number of people in a given place, but most of those are either intrusive (face recognition camera, access cards), difficult to set up or lack accuracy (laser tripwire). In this master thesis, we will develop another people detection technique which is not intrusive and quite promising as it is easy to setup and should yield decent accuracy. By measuring some environmental variables such as CO2, we will try to derive the *person count* using supervised machine learning techniques.

In this work, we will present how we managed to configure the IoT sensors to measure environmental data, the different techniques involved to count the actual number of people occupying the room, needed to build a dataset suitable for supervised learning. We will then expose how we formatted the data from different sources to make a consolidated data set that is readable and suitable for further investigations. We then detail our pre-processing and feature engineering techniques, involving among others, outliers discarding, class balancing, data scaling and feature extraction. The following step will be about our model validation technique, performance assessment and evaluation. We will next disclose the different machine learning techniques we considered, describe some of them in more detail regarding their mathematical formulation and the impact of some of their parameters. The decision and choice between the different algorithm will be explained and we will build a final model being a voting procedure between the best performing classifiers and regressors. Along these last sections, we will consider various methods for the estimation of the performance of our final model. Finally, we will assess the accuracy of the produced technique, discuss about the results, the limitations and reproducibility of the task before presenting some related work that inspired this thesis and concluding.

Chapter 1

Gathering training data

Before finding useful correlations and extracting knowledge, we need to retrieve data that will be used to predict the occupancy of a given room and to train machine learning models. To this end, we will measure what we call `environmental variables` (*CO2, Temperature, humidity, light, motion*) along with the number of people inside a given room.

1.1 Environmental variables

The figure 1.1 below shows an overview of the process used to handle the data produced by the sensors.

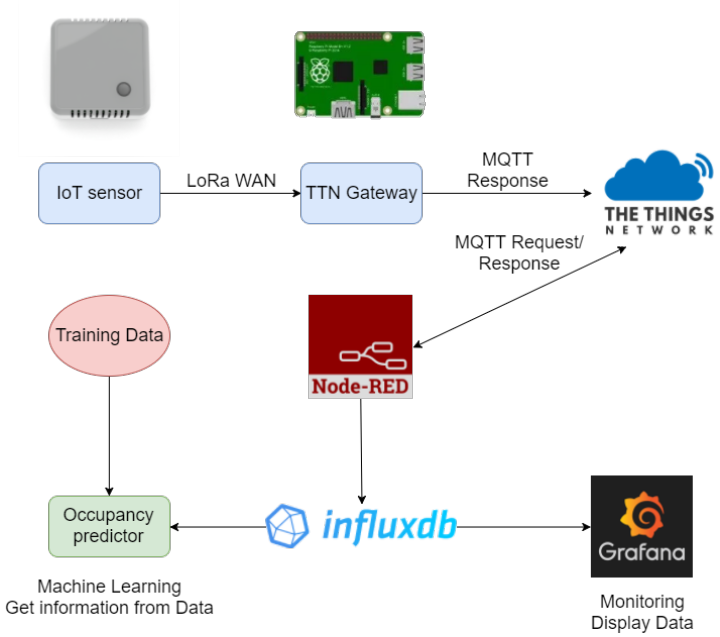


Figure 1.1: Overview of the Tools used to receive the data

Here is a list of the tools used and a brief explanation of their utility before detailing them:

- IoT sensor: Retrieve the environmental data.
- TTN Gateway: Can be implemented on a Raspberry PI with LoRa hat[1].

[1]In order to retrieve data quicker, we relied on another (TTN) gateway than ours.

- The Things Network (TTN): Platform for centralized management of the gateways and sensors.
- Node-Red: Wiring together hardware devices, APIs and online services.
- InfluxDB: A time-series database.
- Grafana: Open platform for analytics and monitoring.

1.1.1 The sensors

The first step to consider is the retrieval of environmental variables. To this end, two Elsys ERS sensors[2] were used (see the top-left picture on figure 1.1).

Those can be easily configured via NFC using the smartphone application[3] (example on figure 1.2). The following parameters were applied:

- Timebase equal to 60 seconds.
- Every sensed attribute is sent at 1 timebase i.e. we sample everything available every minute.
- Over the air activation is enabled.
- The AppEUI and AppKey are set to correspond the values given in `the things network` (see section 1.1.3).

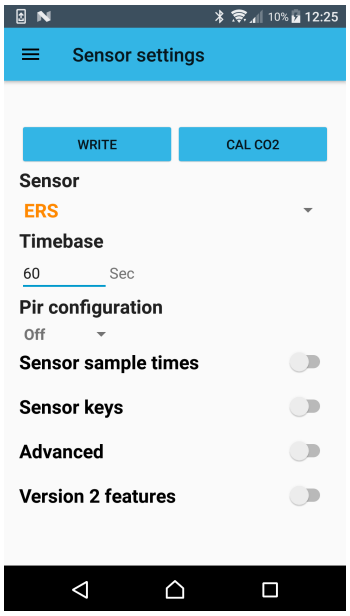


Figure 1.2: Screenshot of the application

Those sensors communicate via LoRa (LoRaWAN in particular), which increases their range along with their battery lifespan. The timebase we configured is quite short for an IoT device, it was mainly to enable faster testing and debugging. Later, we changed it to 300 seconds in order to meet the recommended maximum number of packets per day. Two sensors were placed:

[2] `https://www.elsys.se/en/ers/`
[3] Available on the playstore at `https://play.google.com/store/apps/details?id=se.elsys.nfc.elsysnfc`.

one in the room Paul Otlet in the Réaumur building in Louvain-la-Neuve, and one in the room Parnas in the same building.

The first room is convenient because it regularly holds meetings of several persons (ranging from a few to a theoretical maximum of around 25). Thus it is a good choice for approximate person counting. Furthermore, this area has no ventilation system and only a few openable windows. It was placed on the farther wall from the them at approximately 2.5 meters height.

The Parnas room is smaller and thus has a reduced capacity, it is used by master students needing a space to work or study. The major reason it was chosen is the absence of openable windows and the presence of a classic ventilation system. Here we placed the sensor on the ceiling, at approximately 3.5m height. Those peculiarities, in our opinion, should reduce the outliers and corner cases to be considered in the person detection algorithm (training data gathering) and make the occupancy more predictable (someone opening the window for a few minutes while there are 25 persons could reduce the CO2 level without reducing the temperature for instance).

1.1.2 Receiving the data

The next step is to actually retrieve the data and send it towards `The things network` platform (see section 1.1.3) through which raw payloads of the LoRa packets can be received.

To recover the packets sent by the sensor, we need a LoRa capable device that will then forward this information on the regular Internet network. A first approach considered is using a Raspberry Pi 3 with a LoRa HAT on top of it. Following the sample code given at the website of the LoRa HAT[4]. However, this provides information on how to create a single-channel TTN gateway; as stated on TTN website, this is not LoRaWAN compliant and thus not supported by TTN[5].

Some other solutions involved using different, generally more advanced and expensive hardware, also exist[6].
However TTN provides some interesting alternatives[7] that could be used in the future.

Figure 1.3: Raspberry Pi with a LoRa hat mounted on it

[4] `http://wiki.dragino.com/index.php?title=Connect_to_TTN`
[5] `https://www.thethingsnetwork.org/wiki/Hardware/Gateways/Single-Channel-Gateway`
[6] `https://www.thethingsnetwork.org/labs/story/rak831-lora-gateway-from-package-to-online` `https://frightanic.com/iot/build-a-lorawan-gateway-for-the-things-network/`
[7] `https://www.thethingsnetwork.org/docs/gateways/start/single-channel.html`

The system finally rely on the gateway already present near the offices where the IoT sensors were installed. It worked flawlessly during the duration of this master thesis (10 months). We chose this approach since this aspect is not the central part of the work.

1.1.3 Parsing and storing the data from TTN to InfluxDB

To get and store the data in our server, we first need to create the device in the The Thing Network console and register the physical sensors. Then we use Node-Red[8] to recover the packets and store them in our time-series database (InfluxDB[9]).

Node-Red is program that runs a web server that can be used to easily wire together the node sensors and the database. So here we just need to create a node for a sensor (with all its authentication information) and wire it to a node for the database (also with the database and table information) passing by some function nodes decoding the sensor payload and encoding it to the InfluxDB format. The connection to the sensor is done by a $MQTT$[10] pull that retrieve the payload when TTN receives it. The graphical view of the wiring is shown of figure 1.4[11].

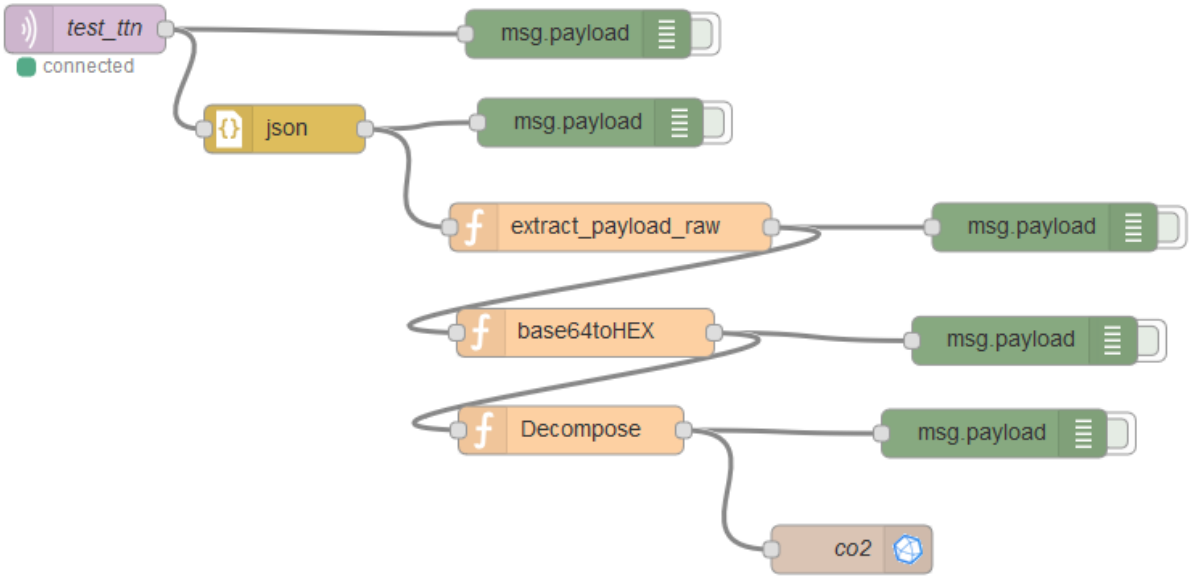


Figure 1.4: The wiring done in Node-Red for one sensor.

The Node-Red server and the InfluxDB database are running on an EC2 Amazon Web Service (AWS) machine, accessible remotely for flexibility purposes.

The values we store in the database for each payload received are :

- The time (created by InfluxDb on insertion).
- The *CO2* level in ppm.
- The *humidity* level (relative).
- The *temperature* level in °C.

[8] `nodered.org`
[9] `https://www.influxdata.com/`
[10] Machine-to-machine Internet of Things connectivity protocol.
[11] The green *msg.payload* nodes are for debugging purposes.

- The *light* level.
- The *motion* detected.
- The battery level of the device.
- The time of the measurement from the device (to detect eventual network latency).
- The identifiers of the device (app and dev ids along with the hardware serial).

The advantage of using Node-Red is that it is easy to use as we just needed to add a *MQTT* node with the sensor information to retrieve its payload and not implement the communication ourselves. And it was also easy for the InfluxDB connection. A very interesting aspect is the facility of adding a new sensor: the second sensor (Parnas room) was added a few months after the first one was installed and "Node-red-configured", and the configuration of the new sensor merely took 10 minutes, testing included.

We chose InfluxDB because this kind of database (time-series) is convenient for IoT applications especially for its flexibility in the tables (called measurements) and easy interpretability. InfluxDB handles well missing values in measurements, allowing multiple various sensors to contribute to a single table.

1.1.4 Basic monitoring

Having raw values inside a database is cumbersome to read[12], a first improvement considered is showing those data in graphs depending on time. To this end, we used Grafana which can make graphs directly and dynamically from a given measurement of our InfluxDB database using a GUI.

Detailing the use of Grafana is outside the scope of this work but here are the main step in building a basic monitoring:

1. Setup a Grafana server (we put it on the same AWS machine as the InfluxDB server) and credentials.
2. Add a data source by providing its location, port and giving it a name (here the data source is our InfluxDB database).
3. Create a Datasheet and add graphs, referencing to the measurement of your database.

The datasheet will graphically show results of a given query (SQL-like or by filling a template) and allow the user to switch the period seen easily. Those are really customizable. Example of Grafana graphs of the collected data are shown on figure 1.5 and figure 1.6

1.2 Number of people in a room

Once the Environmental data is gathered, we can now focus on counting the number of people in a room and the training set will be ready for the CO2-based occupancy detection task. In this section, we consider three[13] different approaches.

It is better to use a fully-automatic approach that would not require any human-intervention after installation. Some scientific articles mentioned light sensors [1] but it was limited to some

[12] e.g. the time in influxDB is represented as the timestamp since EPOCH in nanoseconds.
[13] Although the two last are similar in a sense.

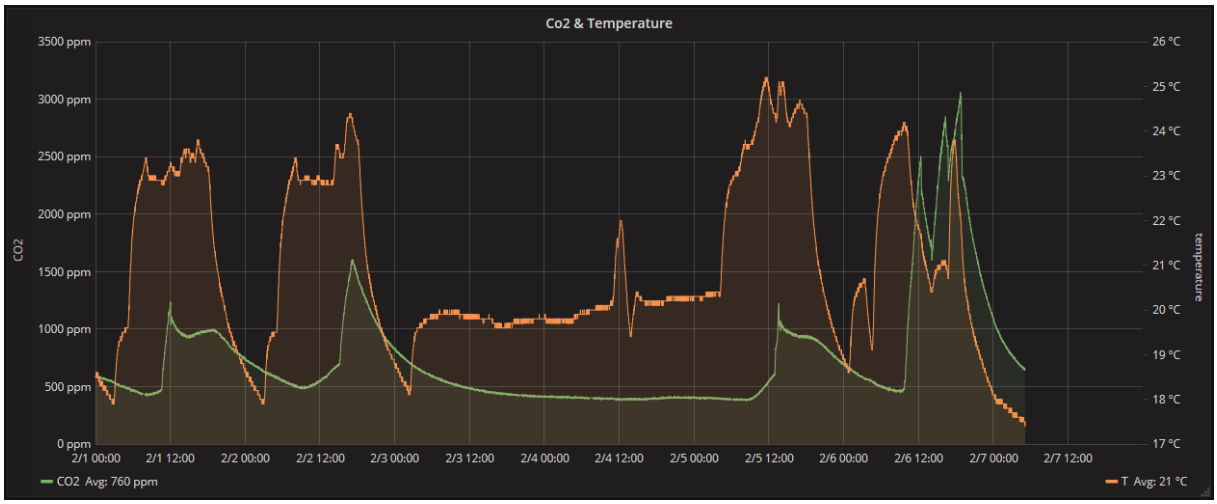


Figure 1.5: Example of Grafana graphs displaying the Co2 and temperature evolution.

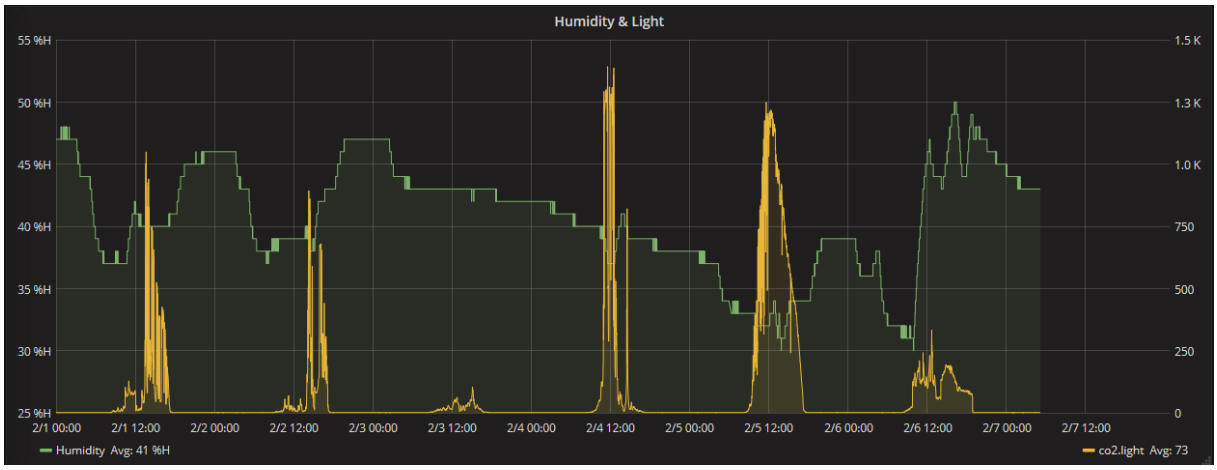


Figure 1.6: Example of Grafana graphs displaying the humidity and light evolution.

rooms with certain kind of door (a windowed door in their example or where the entrance was in front of windows), magnetic sensors [2] to detect when a door is opened (but this does not give much information about the number of people entering/leaving), several lasers to detect the direction of the movement (but this solution seemed quite approximative in counting the people, especially if the door of the room is wide), or using a camera [3]. We first focused on the last option even-though it has some drawbacks:

- It is the most expensive solution we considered.
- It is also the most complex considering the parameters, techniques involved in image processing[14].
- Last but certainly not least, this solution is the most intrusive and require to take image rights into account.

Nonetheless, this is the challenging approach we chose because of its high expected accuracy [3]. A training set should be as close as possible to reality[15].

[14]Some other work in progress is using an infrared camera to avoid performing background subtraction. But we believe this is even more expensive even though more promising.

[15]Not a 100% though since having noise in training data is also a proof that a machine learning model generalize well and is resilient to noise.

1.2.1 Using a camera with motion detection

Considering its main drawback (image rights), some choices concerning the usage of the camera had to be done:

- No image should be stored (except specific videos we made ourselves to configure the camera and test our program) nor sent to a remote server.
- The camera will not be located inside a meeting room but at its entrance. Thus the method used by the camera is not face/person recognition but motion detection.
- The camera will only store one counter: the current number of people inside the room.

Another decision we consider is in which room the system will be applied. To avoid multiple camera's and concurrency, rooms with only one exit were considered. The meeting room Paul Otlet suits this condition and also has a small corridor before its door[16].

The goal is to provide a similar result as in some video found on the internet[17].

Figure 1.7: Example of motion detection application

Hardware used

For cost, availability and portability reasons, we used

- A Raspberry Pi 3 model B with an Micro-SD card (16Go) as shown on figure 1.8.
- A Pi Camera as shown on figure 1.9 and 1.10.
- A power cable 5.2V 2.5A sector micro-usb-B.
- What to fix the camera on the ceiling.

As a side remark, we also thought about using a power bank instead of reliying on the premise power supply that would require a cable. However, a fully charged power bank (estimated capacity of 1800 mAh) can supply a Raspberry performing minimum operations (which would

[16]Of course, we had to choose among the rooms where we had actually placed sensors, this one was the most convenient among the two. The corridor especially, exempt us to differentiate the people just passing next to the door without entering the room from the others that should be counted.

[17]`https://www.youtube.com/watch?v=B29WfqM7iqU`

not be the case with the image processing program) for several hours only. 5 hours and 54 minutes to be exact, the result comes from a simple program logging every minute while the Raspberry is powered by a power bank. Thus using such system is out of question for a fully automated system that should last in the long run.

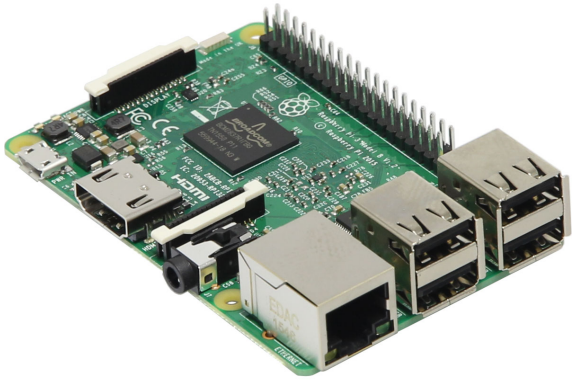


Figure 1.8: Picture of a Raspberry pi 3 model B

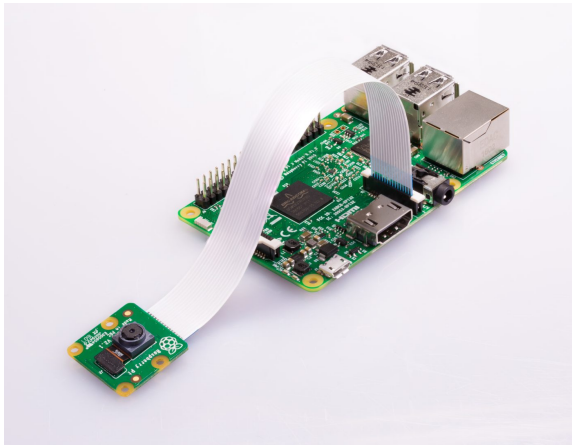


Figure 1.9: Raspberry pi + camera

Figure 1.10: Raspberry pi + camera inside a home-made case. That case can then be placed on the ceiling using Double-sided tape for simplicity

Software and libraries

To avoid re-inventing the wheel, the following softwares were used

- *Raspbian Jessie OS* (2017-09-07).
- Python (the program used to run the camera and the image processing is compatible with 2.7.13 or 3.6.3, but the first one is used on the Raspberry).
- *OpenCV* 3.3.1 for python (and all its latest dependencies)[18]
- *InfluxDB*, *imutils*, *logging*, *numpy* as well as many other (standard) python libraries.

Motion detection program

The main program is contained in the file `pi_counting.py`, it uses a json configuration file in order to ease parameter tuning

Basic usage Using the command line:
`python pi_counting.py -c path/to/conf_file.json [-v path/to/video_file.mp4]`
Since our script uses `argparse` library, one can use
`python pi_counting.py -h`
To display basic help.

Algorithms and techniques The tutorials we found[19] gave us a starting point for the project. However, it required using more complex techniques to achieve better adaptability and precision : the goal of the tutorial was only motion detection and it is not sufficient for this task.

Many opencv methods are used for the motion detection part as well as an history of the points where mouvement is detected: the steps are the following

1. Get an image from the source[20].
2. Resize the image (imutils library) and turn it into black & white (for performance reasons).
3. Apply a `Gaussian blur` to the image to smooth the contours.
4. Apply a background subtraction using an *opencv* object created with the method `createBackgroundSubtractorMOG2`.
5. Erode and dilate the image (more smoothing).
6. Detect blobs using *opencv* `SimpleBlobDetector` object.
7. Update the history of the keypoints, this is used to determine the direction of motions detected (whether a person enters or leaves the room).
8. If a motion has been detected and fulfill certain conditions (length of the path, direction,...), the people counter is incremented/decremented.

[18] For the Raspberry, we followed the explanations given here but applied to version 3.3.1 of opencv: `https://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/`

[19] `https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/` and `https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/`

[20] As a remainder, the program can either capture images from the camera or from a video file.

9. If the time is at midnight, we reset the counter[21].

10. If we have not sent it within the last minute, we push to value of the person counter towards the *InfluxDB* server (the person count can be merged with the data produced by the *Elsys* sensors).

11. Output the frames/values if asked in the configuration (debug purposes) and repeat the operation.

Temporal Performance The performance metric we will consider in this section will be the fps of the running program. On a simple laptop without any optimization, the program run at around 40 fps (using the default parameters or those founds in the tutorial). However, the fps drops at 4 on the Raspberry Pi 3, thus some parameters tuning is required. In fact, resizing the image before applying any modification, filter,... seems to be the simplest and effective way to increase fps, changing the `raw_frame_size` from 500 to 200 increases the fps up to 18 which is enough for this purpose. This even improves the detection in some ways, it increases the smoothing effect of the technique thus avoiding errors from noise in the image.

Parameters Tuning In order to ease the task of parameter tuning and not only rely on trial/errors/adapt, a grid search has been performed to select the best parameters. The process is the following :

1. Record a video where some people enter/leave the room.

2. Annotate semi-manually at each frame the actual number of people in the room.

3. Perform a search and see which set of parameters yield the closest result.

However, the search result (which took several hours of computing) revealed a set of parameters that overfitted greatly the videos provided as training (sometimes detecting multiple entrances and leaves at the same time to be closer to our objective function) and did not generalize at all even while considering the same location. Thus we kept a set of parameters that seemed reasonable to us and that we thought would generalize well.

System accuracy The final step is to assess the performance of the system before considering using the data it will generates.

Even though we used a manually chosen set of parameters, the results were quite deceiving on the few videos we made ourselves, if the parameters yielded reasonable (although not as high as expected following [3]) around 80% of correct detection; it was far lower in other runs and even worse at different time of the day: particularly at the end of the day in winter when nor the sunlight nor the neon lights could sufficiently brighten the area, making it difficult to distinguish between people and shadows, even after modifying the camera brightness and contrast parameters.

[21] This idea comes from [4] to mitigate detection errors effects on the data produced.

Discussion

At first the person count detection using a camera seemed a great idea because it can provide an automated and reliable technique. But we faced two major problems when setting this up.

The first problem we encountered was some reluctance from the people using the room frequently who were concerned about their privacy. Indeed even if our camera was located outside the room and was not storing any image, they were not easy to convince. Eventhough we alerted them that no image would be stored nor sent on the internet because the entire process takes place on the Raspberry Pi itself. The only thing that would be stored are debug logs, and the only thing that is sent is the current value of the people counter. Though, this comforted us in our choice of camera placement and counting technique, since placing one inside the room with person detection would have risen much more concern.

The second problem was that our detection program was quite inefficient. This lack of efficiency could be due to several factors.

- The trade-off between quality (size) and flow (fps) of the video was quite difficult to settle.
- The darkness of the corridor made it hard to distinguish between actual persons and shadows.
- Image processing is quite complex, we would need to spend more time looking for better methods that are simple enough to run on such a small device.

In conclusion, using cameras for the person counting can be a good method if people are willing to accept it and cost is not a big issue; but it requires a bit of configuration and a good understanding and experience in image processing and detection. Also, we think it is highly dependent on the camera location: space, natural light, lighting type, distance from the ground, seasons,...

1.2.2 Button system

We then decided to use a simpler way to collect occupation data. The button system allow the users of the room to tell when they enter and leave the room. The system is composed of two buttons used for increasing (the white one) or decreasing (the red one) the counter of people present in the room. This system requires the participation of the users so we informed them and asked them to use our setup. The button installation has been placed in the Paul Otlet room as well.

Hardware used

The system is composed of:

- A Raspberry Pi 3.
- A *RGB LCD Display* With USB Port For Raspberry Pi.
- Two arcade buttons.
- Two LED's.
- Two 50Ω resistor.

We use the LCD screen to display the number of people currently present in the room. The buttons each contains a LED which is lit when the button is pressed. And the Raspberry listen to the buttons and update the counter on each button release.

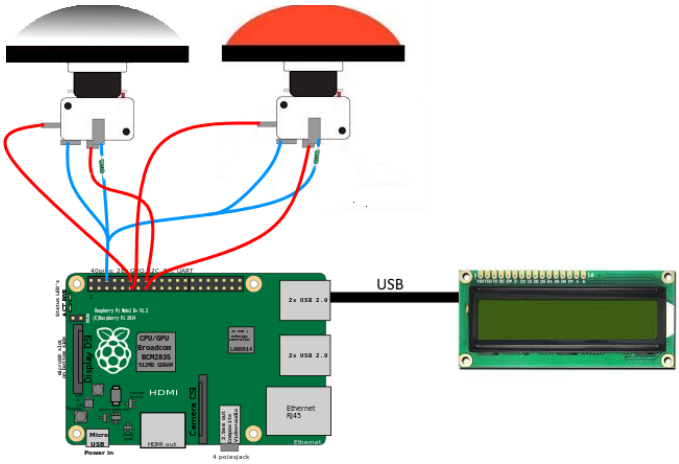


Figure 1.11: Schema of the button system

As you can see on the figure 1.11, we connected the LCD screen to the Raspberry through USB. The white and red buttons are wired to the GPIO 17 and 18 respectively. The LED's are wired to the GPIO 22 and 23 and pass through a $\pm 50\Omega$ resistor to avoid drawing to much current from the Raspberry pi.

On the Raspberry pi, we run a Python script which listen to the GPIO pins of the buttons and update a counter on the buttons release. It also updates the display when the counter changes so users can know its current value. The script also pushes the value of the counter in our influxDB database on every counter change or every five minutes. Note that the value is immediately sent towards the database on each change. This behavior and the 5 minutes rule had to be taken into account while pre-processing the data later on.

Discussion

This button system is a quite easy do-it-yourself to set up but it requires the good will of the room's users. But we noticed, after installing the system and informing the users that they played their role during the gathering and used our system assiduously.

1.2.3 Manual gathering

In order to have our first people count without having to wait for the two first systems to be ready, a QR code was placed near the entrance of the Parnas room that lead to a google form where someone can just enter the number of people in the room at the moment. The google form is able to store such data, as well as the time at which it has been recorded, in a spreadsheet that can be exported as a csv file. Then, a python script can read this file and export the data towards our influxDB database. Another solution we used, is to include those data during the pre processing phase of our machine learning program that will be detailed in the next chapter.

This solution requires more help from people working in this room and can become cumbersome in the long-term. Furthermore, one cannot be sure whether someone will forget to fill the form (or simply lie on the number) while leaving the room, thus the room will be indicated as occupied while it is actually empty. This and the flaws of the other methods presented lead to the data pre-processing phase.

1.2.4 Extracting the data

Before applying pre-processing techniques, we have to extract the sensor data from the database, this is easily done using the following command:

```
influx -username user -password pass -database dbname -format csv
       -execute 'SELECT * FROM measurement_name' > Sensor_data.csv
```

The Google form data can simply be retrieved using the web interface. This gives us the two .csv files that are needed for our computations.

Chapter 2

Cleaning the input data

In this chapter, we will discuss about how we regrouped and formatted our data before applying machine learning methods to it. We will also discuss about the different features we added or removed from the dataset.

2.1 Formatting the data

The first (obvious) step preceding using the actual data is to format and standardize it before actually using it. Also, since our techniques to generate learning data were imperfect, we needed to take that into account before further investigations. The operations can be summarized as followed:

- Read the csv from influx-DB as a dictionary.
- Read, in parallel, the csv of the person_count data (for the manually retrieved information that is, stored in the Google form).
- Discard data outside certain dates: e.g. before we had precise information about. *person_count* or the IoT sensors were correctly configured.
- Add the person count alongside the sensor data. We used the following technique: whenever we have no actual *person_count*, we do not store the data, because it is useless for supervised learning. When we have a *person_count*, we use it for the following sensor data until another *person_count* is given. A modification we made soon is to make a "validity period" for the person_count: since much of our data was made manually, absence of data is most likely to mean inattention of people in the room rather than network or system failure or a really long period of occupancy. We arbitrarily set a "timeout period" of one hour, mostly based on our experience of the rooms usage and people will.
- Since the date is important for our purpose, and the date format are not the same between our *person_count* techniques and the *Elsys* sensors, we had to specify a format for the time-stamps. The most obvious choice for us was to follow the technique used by our data gathering tools: EPOCH time-stamps expressed in nanoseconds (used in influx-DB to avoid conflicts) this also makes it easier to deal with time zones.

2.2 InfluxDB records format

Unlike the classical SQL database scheme, the Influx-DB tables (aka: measurements) are really flexible. However, we stick to the following feature records :

$$name, time, app\_id, battery, co2, dev\_id, hardware\_serial, humidity, light,$$
$$motion, person\_count, temperature, time\_device$$

Although much of this information is unnecessary, it has been used for testing purpose. Also, some data that "feels" irrelevant has been removed in the feature engineering step (see section 2.4).

2.3 Google form formats

The Google form has been used to make easier the retrieving of person_count data in .csv files. The features it contains are the followings :

$$time, person\_count$$

But in these files (because they are generated by Google) the dates use the format:

$$dd/mm/yyyyHH : MM : SS$$

This explains the normalization we had to perform in section 2.1

2.4 Discarding outilers and non-representative samples

Some records had to be discarded for obvious reasons: person_count unavailable, irregularities in training data,... But treating all samples individually and manually is out of question for us: For the two rooms, respectively Otlet and Parnas, we retrieved 189374 and 73829 CO2 samples along with 7990 and 320 person count[1] values.
We started with the Parnas data since the training set is smaller, allowing us to have a first grasp of the overall performance of the techniques before handling a larger data set. We performed the following procedure:

- The *person_count* times are changed to EPOCH timestamp as stated in section 2.1.
- The data is sorted out by timestamp to ease the next operations.
- Discard every sample outside certain dates (to filter out the data that we feel is non-representative such as testing measurements).
- Discard every sample that has been recorded before the first *person_count* value.
- Once a *person_count* has been found, it is prepended to all the following CO2 measurements until either another person_count is found or the value is more than 1 hour old. In this last case, the sample is discarded.[2].
- We then are able to perform feature extraction, this will be explained in the next section 2.5.

[1] Remember tha Parnas room is entirely manual, while the Paul Otlet re-submit its value every five minutes.
[2] Note that, as stated before, the one hour value was arbitrarily chosen based on our knowledge of the usage of the rooms.

Since we configured the button system in the Paul Otlet room to send its value whenever it is changed or every 5 minutes, most of the data includes 0. On our first experiments, we found that splitting the data with a 10% testing set gives a batch with all 0's person_count value, thus yielding 100% accuracy for most of the model (because they are always right if they say the value will be 0). For this reason, we had to discard some values, remove the extensive zeroes, from this data set. Because the values for the Parnas rooms are only sent by people, this data set does not suffer from this problem.

This filtering is also important to ensure class-balance. Indeed, some machine learning techniques assume this among the training set: "*Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.*"[3]. The 0 value is quite dominant in our case and thus requires this filtering technique in order to avoid selecting the biased model that always tells the room is empty and gets 80% accuracy over the training set but is useless for our purpose.

Following this process leaves us with a much smaller/manageable/representative dataset of 3346/8612 CO2 samples for room Otlet/Parnas respectively. Note that the measurements from the Paul Otlet room were so unbalanced that it actually now has less samples than the Parnas one. This is also explained by the fact that this room is mostly used for meetings; which occur less often than students working and studying.

2.5 Feature Engineering

First, we performed some "arbitrary" feature engineering for obvious reasons : as said earlier, the name, time, app_id, battery, dev_id, hardware_serial, time_device should not have any influence on the number of people in the room, we thus removed them from our dataset. The next step involved feature extraction. Following [4] we performed some feature engineering : we added the following (quite trivial) features:

- "*smoothed CO2*": mean CO2 level on the five previous minutes (we take the sum of the four previous measurements plus the current one and divide it by 5. The number of minute was arbitrarily set to 5).
- "*derived CO2*" is related to the derivative of the *CO2_smoothed* curve. The actual formula is the following: We take the previous CO2 measurement and the current one and compute:

$$\frac{Y2 - Y1}{X2 - X1}$$

  Where $Y\_$ is the CO2 5-minutes mean value at current time
  And $X\_$ is the time (EPOCH in nanoseconds as said earlier) this is to avoid errors from delayed samples.

These were the main features we thought about before performing some early machine learning experiments. Although some other feature extraction might be interesting (such as changing the formula's for CO2 derivative by computing derivative over 10-minutes means, raw co2 values,...) We decided to limit a bit our work in this sense to get a grasp over the actual performance for those data before going any further.
This leaves us with the following 7 features:

$$co2, co2\_smoothed, co2\_derived, humidity, light, temperature, motion$$

[3] `http://scikit-learn.org/stable/modules/tree.html`

2.6 Dataset analysis

Before applying machine learning to it, we will look at the data in order to have a better grasp about what we work with.

We analyzed the correlation between the features[4]. As you can see in figure 2.1, the *co2* and the *co2_smoothed* features are strongly correlated (±100%), which seems logical as the second feature is just a local mean of the first one. Another interesting correlation (±50%) is between the *co2* and the *temperature* features. It would seem that those two evolve together which can be explained by the fact that people in the room will produce co2 and heat so those values will evolve together.

Moreover, some features are strongly uncorrelated with the others like *humidity* or *co2_derived*. It means that those features could be useful for the estimators to produce a better prediction as the information they carry is not already present in other features. So it confirms that the feature we produced earlier (*co2_derived*) could be useful to give more information to the predictors as that information is not present in other parts of the data.

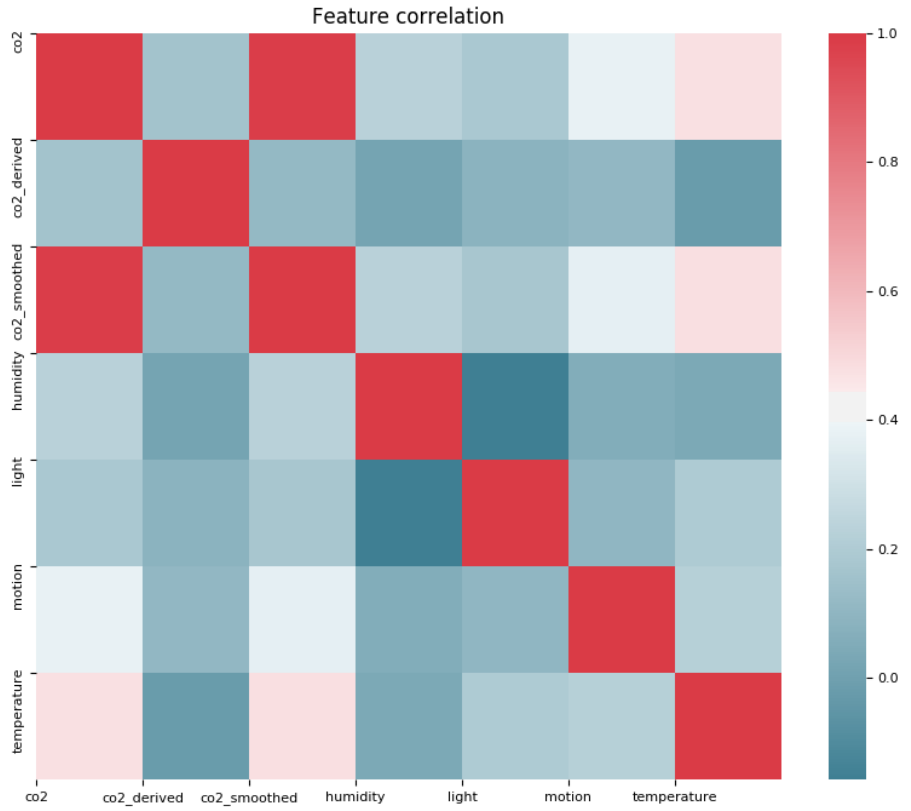


Figure 2.1: Heatmap of the correlation between the features (for the Parnas dataset).

Concerning the dataset, another important aspect (introduced a bit earlier) is the proportion of zeros in the person count. Indeed if the dataset contains too much zeros as person count, a model predicting always this value would have a good accuracy while being useless. After the discarding phase in which many zeros were removed (see section 2.4), the dataset of the Parnas

[4]To do so we used the python *panda* library allowing use to compute feature correlation easily.

room has a 15,58% proportion of its data falling in this category (see figure 2.2). It is a good fraction as it is not too high thus avoiding making always-zero predictions too accurate and not too low so the models can actually predict empty rooms (which is also important because many applications might be interested to know if the room is vacant or not with quite high accuracy). However, even after having removed extensive zeroes in the Otlet data set, nearly $\frac{2}{3}$ of the samples have a person count of 0 (see figure 2.3). The figures 2.2 and 2.3 provide histograms of the usual number of people in the rooms. Figure 2.4 shows the same data as 2.3 but without any 0 in order to better see the usual proportion of person count.

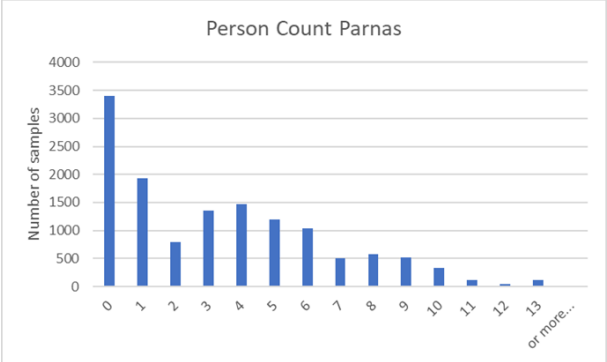


Figure 2.2: Histogram of the number of samples having a given person count value for the Parnas room.

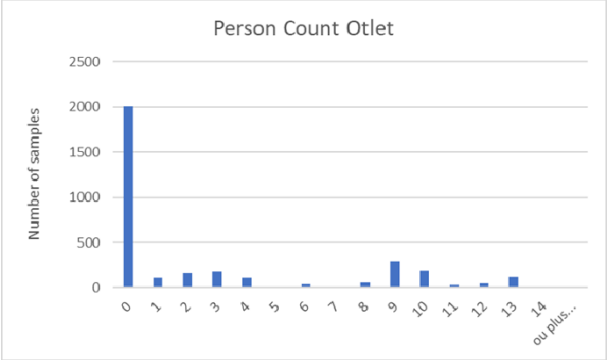


Figure 2.3: Histogram of the number of samples having a given person count value for the Otlet room.

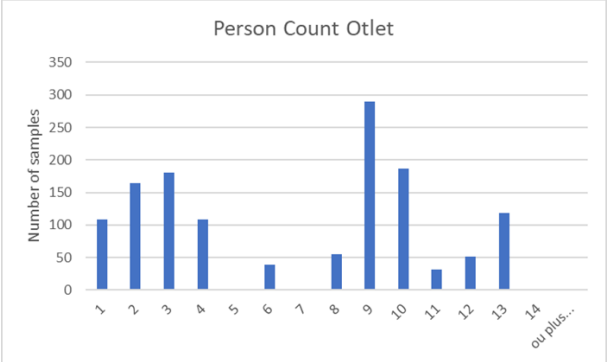


Figure 2.4: Histogram of the number of samples having a given person count value for the Otlet room with the 0 removed.

Chapter 3

Finding a correlation between environmental variables and number of people

Now it is time to manipulate our data and attempt to extract knowledge from it using supervised learning technique

3.1 Early machine learning experiments

In this section, we will detail our methodology, techniques, heuristics used to predict the number of people in a room (*person_count*) based on the *CO2* level, *smoothed CO2* level, *derivative CO2* level, *humidity*, *temperature*, ...

We decided to search the machine learning types of models (e.g. decision tree, SVM,...) in a best-first search manner. To this end, we decided to use the `sci-kit` [5] [6] python library from sklearn[1]. Because it is simple, reusable, built on top of other famous libraries and open-source, it is a popular choice for machine learning. It includes many different and well known machine learning techniques. However, if one is more interested in neural networks or similar techniques, he might consider using other libraries, such as tensorflow or keras, since *sci-kit* does not provide many of those and does not support GPU acceleration or other optimizations.

We computed a basic grid-search of hyper-parameters for each model that do not pose problems. For example, the `Gaussian process`, while *sci-kit* states *"Gaussian Processes (GP) are a generic supervised learning method designed to solve regression and probabilistic classification problems"*[2], caused instant memory error on our machines even with a quite small data set and with different hyper-parameters. Once we have identified some promising models, the idea will be to use some "aggregation", that is combining several algorithms, in order to build a strong and generalizing model. *sci-kit*[3]gives some examples of techniques. Of course, before each grid search, a test set is first taken randomly and separated from the rest of the data: so that the examples used for the final evaluation of a given model have not been seen during the training, parameters tuning phase. We decided to randomly pick 10% of the data for this purpose. We also managed to set an arbitrary seed to ensure the reproducibility of our evaluations.

[1] `http://scikit-learn.org/stable/`
[2] `http://scikit-learn.org/stable/modules/gaussian_process.html`
[3] `http://scikit-learn.org/stable/modules/ensemble.html`

3.1.1 Parameters tuning and score computation

In order to know which machine learning technique would lead to the best accuracy, we implemented a testing script computing the accuracy and variance of several models with several scoring functions (see section 3.1.5).

You can see in the pseudo-code in listing 1 the main frame of our algorithm. For each algorithm, for each scoring function, we randomly partition the data into training and testing sets (10% of data in training and 90% in test) 5 times. Then for each partition, we find the best model by grid search cross-validation and we compute and save the score of that model on the test set.

Thanks to the 5 different random partitions, we get different scores per technique and per scoring function allowing us to estimate the mean score and the variance of that score. A too small number of different partitions would result in a imprecise mean and variance while a too large one would take too much time so we took 5 partitions as a trade-off.

In the cross-validation phase, we first make a grid search on the parameters of the machine learning technique. And for each set of parameter, we make a n-fold (5-fold) cross-validation to know its mean score. Then we return the model that got the best score with its set of parameters. The largest grid search involves tuning 4 parameters over 3-5 possible values each. Thus requiring at most $5^4 = 625$ model cross-validations, each cross validation being 5 folds leading to a maximum of 3125 different model training only for the grid search. This must then be repeated 5 times while varying the training/test set.

```
for technique in techniques:
    for scorer in scorers:
        for iteration in range(5):
            (X_train, Y_train, X_test, Y_test) = random_partition(data)
            model = gridSearch_crossValidation(technique, X_train, Y_train, scorer)
            model.fit(X_train, Y_train)
            Y_pred = model.predict(X_test)
            score[tecnique][scorer][iteration] = scorer(Y_test, Y_pred)
```

Listing 1: Pseudo code of our algorithm to evaluate the accuracies of the machine learning techniques

In order to make this script work, we had to make some arrangements:

Some machine learning techniques need much time to execute on our dataset, especially for the grid-search part. We had to define a timeout for the search such that we do not spend too much time on a single method. This idea of timeout is practical but also necessary if we wish to use a model that could be trained in a reasonable amount of time on a larger dataset than the one considered. The timeout was set to 5 minutes on the entire grid search (not the training itself) knowing that it performs its training on several parameters that have a small amount of possible values and is based on 5-folds to permit a quicker tuning since the number of computations is quite large.

For the timeout to work correctly, we had two reasonable options. A post on the internet[4] gives some insights of why threads should not be used: either use signals (but this would then limit our program to Linux operating systems and since we performed part of the work on windows-only computers to have concrete results more quickly, by running the program in

[4] https://eli.thegreenplace.net/2011/08/22/how-not-to-set-a-timeout-on-a-computation-in-python

parallel on several computers, this was not a viable option) or sub-processes. We worked out on the latter, thus the grid-search is performed in a sub-process created by the *multiprocessing* library that is killed after 300 seconds if it has not terminated by itself. This way, we avoid wasting resources at each operation.

We also stored some logs for every different scoring metrics and models along with some other information useful for debugging. Whenever a timeout is hit, it is also logged as a warning. This logging system allows us to keep track of every results we had and associate them with the different choices we made. Also, because we might re-use those results and perform some stats on them later, they are also stored in a pickle file for easy retrieval.[5]

3.1.2 Regression vs Classification

Our first idea was naturally to use regression: even if the number of people is a discrete value, the idea would be to compute a floating value and round it to the closest integer to get the prediction. Since predicting 10 or 2 whenever the actual value is 9 are different mistakes, they should be penalized differently.
We thus originally considered all the regression techniques.

However, some regression techniques are originally classification ones, and we were wondering how these would perform on the same problem. Even though they should have a disadvantage since the classification is unaware of the fact that the value 2 is closer to 3 than 10. A comparison of these techniques will be done in the model selection (see section 3.1.6).

3.1.3 List of Machine Learning Models used

Here is the list of all the techniques[6] we managed to use (or at least we attempted to use). And, spoiler alert, they are numerous. Fortunately, some are really similar in practice.

For the **regression** first:

- Automatic Relevance Determination Regressor (ARD)
- BayesianRidge
- ElasticNet
- ElesticCV
- KNeighborsRegressor
- Gaussian Process Regressor
- Lars
- Lasso
- LassoCV
- LassoLars
- LassoLarsCV
- LassoLarsIC
- LinearRegression
- LinearSVR
- NuSVR
- Orthogonal Matching Pursuit
- Orthogonal Matching Pursuit CV
- Passive Aggresive Regressor
- Perceptron
- Ridge
- Stochastic Gradient Descent Regressor(SGDRegressor)
- Support Vector Regressor (SVR)
- Decision Tree regressor

[5] Although it is not the safest solution, it is the most direct one.

[6] The documentation of all those techniques can be found at `http://scikit-learn.org/stable/supervised_learning.html#supervised-learning`

- Random Forest Regressor
- Multi-layer Perceptron regressor (MLP regressor)
- KNeighbors Regressor

And featuring now the **classifiers**:

- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- Support Vector Machine Classifier
- BernouilliNB
- MultinomialNB
- GaussianNB
- Decision Tree Classifier
- Random Forest Classifier
- Multi-layer Perceptron classifier
- KNeighbors Classifier

Since we are considering so many models at first, our idea is not to hesitate to discard a technique that yield errors, warnings, convergence problems,... It is preferable to count on other methods (and method that we know in detail or at least how to choose their parameters) rather than insisting on tuning the parameters of so many algorithm extensively. Thus we discarded `ARD`, `Gaussian Process Regressor`, `QDA` and `MultinomialNB`.
We will detail some of the models that were the most useful, the objective not being to retranscript the entire *sci-kit* documentation.

3.1.4 General description of the Models

Here we will give some insights about the techniques we encountered, provided in the *sklearn* library. With each description, will be our first impression about them either based on the documentation or our previous experience. This means the following are the "before execution" ideas. This section and the following one : section 3.1.6 will help underline our expectations vs actual results for our task.

Linear Techniques

Before looking the many available models that fit this category, we assumed that those would have the following characteristics.

- Be fast to compute.
- Have a good accuracy if the relations between the input features and the output, *person count*, are linear.
- Have an average or low accuracy otherwise.
- Should not have large memory consumption.
- Being highly understandable. In this case, detecting linear correlation between features and output.

From the *sci-kit* documentation, we underlined the following techniques:

Linear Regression
This technique is the classical regression to a linear correlation between the features and the output.

- *"Linear Regression relies on the independence of the model terms"*[7] which is definitely not the case for some of our features. Indeed the *smoothed CO2* or *derived CO2* levels are not independent from the raw measurements as they are computed from them.
- "*Using the least-square errors as learning metric can lead to high sensitivity to noise.*"[7] Which will cause problems in this case because the person count measurements are mainly manual, human-made, so we should be careful with noisy data while using plain linear regression.

Ridge Regression
This technique adds a penalty on the size of coefficients, this helps to avoid some correlation issues that can be faced with the previous technique such as overfitting, sensibility to noisy data, etc.

The formal definition of overfitting (from [7]) is the following : Consider error of hypothesis h over training data: $error_{train}(h)$ and the entire distribution D of data: $error_D(h)$

Hypothesis $h \in H$ overfits training data if there is another hypothesis $h_0 \in H$ such that

$$error_{train}(h) < error_{train}(h_0)$$

and

$$error_D(h) > error_D(h_0)$$

This means that a model overfits a training set compared to another model. A model **A** overfits a model **B** when the first one has a better accuracy on the training set but a lower one on the validation set than the second. This means that although **B** has a lower accuracy on the training set (the learned model do not completely separate the training data) it actually generalize better to the task overall. **A** will outperform **B** only on examples already seen rather than on unseen ones (which is generally the objective of machine learning).

Lasso
"Lasso uses sparse coefficients, this will help to reduce the number of parameters to be optimized in the training phase"[7] and maybe allow faster computations.

As the `Lasso` regression yields sparse models, it can be used to perform feature selection. For high-dimensional datasets with many co-linear regressors, `LassoCV` is most often preferable. However, we thought that 7 dimensions, while being a "not so low number" is still quite reasonable and should not enter this category. Especially because the *CO2* features are related. The alpha parameter controls the degree of sparsity of the coefficients estimated.

Elastic-net
"Elastic-net is useful when there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both"[7]. This

[7]From *sci-kit* documentation, `http://scikit-learn.org/stable/modules/linear_model.html`

technique should be helpful with our first set of features, in which at least 2 are correlated. If both are significant in the classification/regression problem they would likely be selected by this model.

Orthogonal Matching Pursuit (OMP)

"Fits a linear model with constraints imposed on the number of non-zero coefficients"[7]. This technique might be useful with huge datasets in order to impose a limit on the complexity of the problem. Since our datasets are composed of 7 features and several thousands samples only, this model should not have a significant impact compared to other linear techniques. Furthermore, following the computation technique of this model, we think it might be sensitive to correlated features.

Bayesian Regression

"Include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand introducing uninformative priors over the hyper parameters.

The advantages of Bayesian Regression are:

- *It adapts to the data at hand.*
- *It can be used to include regularization parameters in the estimation procedure.*

The disadvantages of Bayesian regression include:

- *Inference of the model can be time consuming.*
- *Bayesian Ridge Regression is more robust to ill-posed problem."*[7]

This regression, as the name suggests, assume Bayesian priors over the model parameters.

Perceptron

"The Perceptron is another simple algorithm suitable for large scale learning. By default:

- *It does not require a learning rate.*
- *It is not regularized (penalized).*
- *It updates its model only on mistakes."*[7]

We chose to try this model because we already used it in other contexts where it performed quite well for simple classification tasks. We expect it to be fast and simple to configure but it could lack accuracy if the function to be predicted happens to be non-linear.

Passive Aggressive Algorithms

Those algorithms are variants of `Perceptron` with a regularization parameter. We also used it in order to see if this parameter will impact its accuracy compared to the regular `Perceptron`

Robustness regression

"Robust regression is interested in fitting a regression model in the presence of corrupt data: either outliers, or error in the model"[7]. This aspect of the technique might be useful for us because we are unsure of the validity of the entire dataset (although we made some measurements ourselves which can be trusted, for the other we rely on the willingness of other people using the rooms to give us accurate data).
However, we thought that because we relied on manual methods to measure the person count, and because we trust the sensors[8], we can have some confidence in the validity of our dataset. Even if our final model must be robust to noisy/erroneous data, you will see in our first experiments that, if the number of samples is large enough, the models are robust in general: i.e. the variance of their accuracy is low.
However, for smaller data set, this technique might be of interest, but we decided not to consider it for now.

Huber regression

Huber regression is a particular technique among the robustness regressions that seems more suitable to our problem. *"The HuberRegressor is different to Ridge because it applies a linear loss to samples that are classified as outliers. A sample is classified as an inlier if the absolute error of that sample is lesser than a certain threshold"*[7]. As stated earlier, we did not consider this technique at first but it might become interesting if the classic models are not robust enough.

Polynomial regression

To alleviate the problem of fitting non-linear models, a first improvement is to multiply the features among them, in order the build polynomial features from the coefficients. From *sci-kit*: *"In the standard linear regression case, you might have a model that looks like this for two-dimensional data:*

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2$$

If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

The (sometimes surprising) observation is that this is still a linear model: to see this, imagine creating a new variable

$$z = [w_0, x_2, x_1 x_2, x_1^2, x_2^2]$$

With this re-labeling of the data, our problem can be written:

$$\hat{y}(w, x) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5$$"[7]

So this is the (base) theory provided about polynomial regression. However, because we already have 7 features, and some are derived from the others, we thought about two issues:

- This will increase the correlation between all the features, ending up providing many (cor)related features that will inevitably cause problems for some linear techniques.
- In other techniques, adding those features will result in the *Curse of dimensionnality*.

The concept of *Curse of dimensionnality* will be explained in the `KNNeighbors` technique, where it can be more easily/graphically explained.

[8] We also inspected the output of the sensors in *Grafana* in order to detect anomalies, outliers, etc for the period of time we considered.

Discriminant Analysis techniques

Sci-kit comes with two classifiers falling in this category: `Linear` and `Quadratic Discriminant Analysis` (LDA & QDA) that may fit, as their names suggest, a linear and a quadratic decision surface, respectively.

"These classifiers are attractive because they have closed-form solutions that can be easily computed, are inherently multiclass, have proven to work well in practice and have no hyperparameters to tune.

They can also be used to perform supervised dimensionality reduction

If in the QDA model one assumes that the covariance matrices are diagonal, then the inputs are assumed to be conditionally independent in each class, and the resulting classifier is equivalent to the Gaussian Naive Bayes classifier"[9]. This might be an issue in our case where some features are correlated.

Also, shrinkage is a tool to improve estimation of covariance matrices in situations where the number of training samples is small compared to the number of features. If the `LDA` seems correct on a large number of samples but accuracy drops significantly with fewer examples, we might use this technique to increase accuracy.

One can consider [8] [9] and [10] for more details about `LDA`.

Support Vector Machines

"The advantages of support vector machines are:

- *Effective in high dimensional spaces.*
- *Still effective in cases where number of dimensions is greater than the number of samples.*
- *Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.*
- *Versatile different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.*

The disadvantages of support vector machines include:

- *If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.*
- *SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation."*[10]

The advantage of high dimensionnality support is negligeable in our case, as we have only 7 features in our dataset. We also believe that being memory efficient is secondary due to the reduced size of our final dataset. This technique has the advantage, although it does not provide probability estimates, to explain well which samples are determinant in the decision. Furthermore, it has been proven really effective in some classification problems: in a previous course project, it achieved 92% accuracy on a classification problem of digit recognition using Zernike moments as input features (where the number of features was quite high, around 40).

[9] From *sci-kit* documentation, `http://scikit-learn.org/stable/modules/lda_qda.html`
[10] From *sci-kit* documentation, `http://scikit-learn.org/stable/modules/svm.html`

Discussion over the mathematical model and its parameters

A resumé of those statements from *sci-kit* documentation is the following: *"SVM are built by choosing samples among the training set that are representative of the decision boundary of the problem. Those samples are called the support vectors and are given weigths to estimate to which class a new sample should belong"*[10]. Also, SVM's use the *kernel trick* we explained in the previous section.

Sci-kit provides 4 kernels:

- linear
- polynomial
- rbf
- sigmoid

The kernel is an important concept while tuning hyper parameters. *Sci-kit* has also many things to teach us about this concept. First, is the difference between stationary and non-stationary kernels:
"Stationary kernels depend only on the distance of two datapoints and not on their absolute values and are thus invariant to translations in the input space, while non-stationary kernels depend also on the specific values of the datapoints. Isotropic kernels are also invariant to rotations in the input space.
Hyperparameters can for instance control length-scales or periodicity of a kernel For each hyper-parameter, the initial value and the bounds need to be specified when creating an instance of the kernel".[11]

As a further explanation about kernel parameters, we will analyze a bit the formula of the RBF kernel given by the following expression:

$$k = exp(-\frac{||x - x'||^2}{\sigma^2})$$

First, we see that if the test value `x'` is equal to the example from the training set, the expression inside the exp will be equal to zero thus the kernel will yield the value 1.

If we decrease the value of σ_d drastically, the exponential argument will be a large negative number, thus we have

$$\lim_{\sigma_d \to 0} k = 0$$

for any value `x'` that is not exactly equal to the observation from the training set. Thus the weight of this slightly different observation `x'` will be close to 0. This behavior reminds that of a hash table, the learned function will only take into account the elements from the training set that exactly match the test example; thus greatly increasing the overfitting behavior of our model.

When the data is linearly separable in some appropriate feature space, the separating hyperplane is not unique. The *maximal margin hyperplane* separates the data with the largest margin. However, *"Due to outliers the samples may not be linearly separable in the feature space"* [7]. The parameter **C** allows for the dual problem of soft-margin optimisation. We introduce some tolerance in order to ease the classification problem. The objective is then two-fold:

- Minimize the error of the classifier against the dataset.
- Maximize the margin, that is, the confidence of the classifier in its prediction.

[11] From *sci-kit* documentation, `http://scikit-learn.org/stable/modules/gaussian_process.html`

"C is a positive constant balancing the objective of maximizing the margin and minimizing the margin error".[10] ***"C** is 1 by default and it's a reasonable default choice. If you have a lot of noisy observations you should decrease it. It corresponds to regularize more the estimation".*[10] In short, **C** is an important parameter that can be used to avoid overfitting.

"It is highly recommended to scale your data. Note that the same scaling must be applied to the test vector to obtain meaningful results"[10] *sci-kit* says.
Having seen this statement several times across the documentation, and to avoid discriminating some possibly interesting models, we decided to scale the data (the input features values) for all techniques presented.
Also, *sci-kit* warns about unbalanced problems, some classifiers might overfit or be biased towards some class/output number if it was overly represented in the training set. This was another reason for us (along with avoiding to train a classifier that would always yield the same number) to remove the extensive periods where the rooms were empty.
However, *sci-kit* also propose the following solution: *"Use the keyword class_weight in the fit method. It's a dictionary of the formclass_label: value, where value is a floating point number > 0 that sets the parameter C of class class_label to C * value".*[10] But since we already need to pre-process our data and scale it, we preferred to perform this operation only once at the beginning instead of applying different techniques for all the classifiers/regressors.

SVC vs SVM

Sci-kit can use `SVM` either for classification (`SVC`) or regression (`SVR`). *"The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction".*[10]

SVR's

There are three different implementations of `Support Vector Regression`: `SVR`, `NuSVR` and `LinearSVR`. `LinearSVR` provides a faster implementation than `SVR` but only considers linear kernels, while `NuSVR` implements a slightly different formulation than `SVR` and `LinearSVR`. The parameter nu in `NuSVC/OneClassSVM/NuSVR` approximates the fraction of training errors and support vectors.

As stated earlier, `SVM` are mature and well documented machine learning techniques. [11] and [12] provide the mathematical tools that are the basis for `SVM`. [13], [14] [15] and [16] explore more in details the formulation, parameters and kernels of these techniques.

Nearest Neighbors

`Nearest Neighbors` is a classic in machine learning. It follows this principle: if a new sample is close to some training examples with class **A** for the input features, then its labeling class should be the same. It follows the proximity principle and assumes that what is close in input is similar in output.
Mathematically, we need to give a function computing the distance between samples (e.g. euclidian distance) and a parameter K that will represent the number of nearest neighbors (NN) that will be involved in the classification of new samples. Then, we search the KNN of the new samples, see which class is dominant and label that example with this class. Note there are

several variations, usually, the closest examples might have a stronger weight in the vote.[12]

The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). Since many examples can be quite similar, a radius approach might be interesting. It is often successful in classification situations where the decision boundary is very irregular. The optimal choice of the value is highly data-dependent: in general a larger suppresses the effects of noise, but makes the classification boundaries less distinct.

An important aspect of the `KNN` algorithm is its computationnal complexity: while many classifiers have a long training time, they are generally faster at predicting new samples. Nonetheless, `KNN` breaks this rule because : TRAINING TIME $= O(1)$, CLASSIFICATION TIME $= O(n)$. Indeed, the training consists only about retrieving the examples without further computations, which we assume is immediate because the data is already in memory.
However, in order to find the KNN, the naive algorithm involves computing the distance between the new sample and all training ones, resulting in an O(n) complexity, n being the number of training samples.
However, more advanced techniques from *sci-kit* includes:

- Algorithm $= 'kd\_tree'$ to reduce classification time to O(log(N)).
- Algorithm $= 'ball\_tree'$ To address the inefficiencies of KD Trees in higher dimensions.

For high-dimensional parameter spaces, the `KNN` method becomes less effective due to the so-called *"curse of dimensionality"*. This issue was one that prevented us from extracting many features from our original data. A simple illustration will be more helpful than a long explanation.

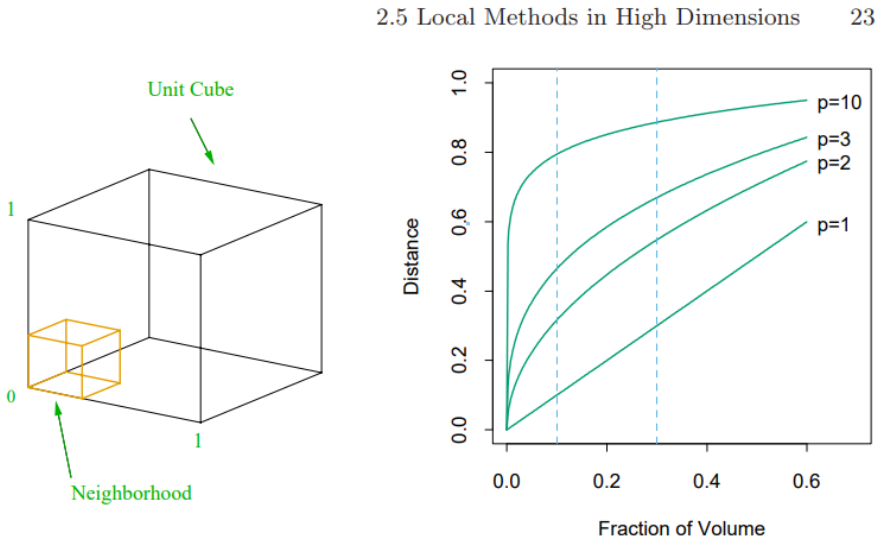


Figure 3.1: Illustration from The Elements of Statistical Learning [17]

"Assume the data is uniformly distributed in a d-dimensional unit hypercube. How far from a query point do we have to go to capture 1% (resp. 10%) of the data? In ten dimensions (d = 10), we need to cover 63% (resp. 80%) of the range of each coordinate to capture 1% (resp. 10%) of the data. So a 10% neighborhood contains a vanishlingly small fraction of the data as d increases" [7] citing [17].

[12]This is indicated in *sci-kit* by the `weigth` parameter by specifying the value `distance` instead of the default `uniform`.

Thus even with 7 features, we might need a very large number of samples in order to be representative of the entire input space. Yet, we think that some states in the space are unreachable under normal conditions, partly because some of our features are correlated: it will be rare to have a very high *CO2* level while the 5 minute mean remains low, otherwise, that is the derivative that will be high. Also, we think the *temperature* has some link with the *CO2* level and its diffusion. There are many reasons why we should not have samples in the entire mathematical , 7 dimensions, input hyperspace.[13]
Even if we should take care about this issue, we decided to test this technique, because we already have used it and it is simple to understand and therefore to manipulate its parameters.

Naive Bayes

The Naive Bayes (NB) technique is an "early" popular form of machine learning, it was historically applied to sort e-mails between legitimate ones and spam. This technique applies Bayes' theorem with the "naive" assumption of independence between every pair of features (which as another reminder is not the case in our training sample).
While Decision trees, SVM, ... directly estimate the decision boundary and Gaussian processes assumes the data follows some parametric distribution, Naive Bayes assumes conditional independence between the features. The different `Naive Bayes classifiers` differ mainly by the assumptions they make regarding the distribution of $P(x_y|y)$ that is the probability to observe the input features knowing the output class.
This algorithm is also known to be very fast, both in its training and its testing phase. However, it is generally applied to classification problems, not regression. Its "simplicity" might hamper its accuracy in our case.

Nonetheless, we decided to apply `MultinomialNB`, `BernoulliNB`, and `GaussianNB` to our dataset to see if we would have some good surprise about the performance of these classifiers.

Decision Trees

"Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The decision trees have the advantage to be well studied/understood and clearly explainable models (even to non experts). Thus their advantages are well known:

- *Simple to understand and to interpret. Trees can be visualized if they comprise a reasonable amount of understandable features.*
- *Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.*
- *The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.*
- *Able to handle both numerical and categorical data (input features). Other techniques are usually specialised in analyzing datasets that have only one type of variable.*
- *Able to handle multi-output problems.*

[13] As we have scaled our data to have a mean around 0 and a variance of one, the input space is closer to the unit hypercube than with the raw data.

- *Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.*
- *Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.*
- *Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.*

The disadvantages of decision trees include:

- *Decision-tree learners can create over-complex trees that do not generalize on unseen observations. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. Often, this problem can be avoided with post-pruning of the tree.*
- *Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble (Random Forests).*
- *The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. 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. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.*
- *There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.*
- *Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree."*[14]

We see that Decision Trees are well understood but might easily conduct to overfitting, bias, unstability,... However, those behaviors are well known and documented so these flaws can be countered by some pre processing techniques (class balancing, feature scaling, removing outliers,...) and feature extraction (computing means over minutes, derivatives,...) while keeping a good accuracy and still being able to understand clearly the boundaries of the trained model.

Basic mathematical formulation

Here we will explain the basic formulation of the Decision Tree learning algorithm. Note that this will be the baseline of the technique and that many improvements have been done in order to improve its efficiency, effectiveness, robustness,... The decision tree building uses a kind of greedy algorithm that creates splits based on the values of the input features. The greediness comes from the way the split are computed, the algorithm compute the **entropy loss** (or information gain) from splits on different features, and select the one that yields the highest improvement. This search pattern comes from the fact that computing the best tree (i.e. the shortest tree with the best classification results) is an intractable problem and has an exponential complexity. Thus

[14] From *sci-kit* documentation, `http://scikit-learn.org/stable/modules/tree.html`

attempting to find such optimal solution is time-consuming and useless. Furthermore, choosing the optimal tree can often lead to overfitting and thus, the model would not generalize well[15].

There exists different measures of "entropy" in the literature. Let p_{mk} be the proportion of class k observations in node m.

$$p_{mk} = 1/N_m \sum_{x_i \in R_m} I(y_i = k)$$

Common measures of impurity are Gini, Cross-Entropy and Misclassification.

$$Gini(X_m) = \sum_k p_{mk}(1 - p_{mk})$$

$$CrossEntropy(X_m) = -\sum_k p_{mk} \log(p_{mk})$$

$$Misclassification(X_m) = 1 - \max(p_{mk})$$

There are also different common techniques used to build decision trees: ID3, C4.5, C5.0 and CART. *Sci-kit-learn* uses an optimized version of the CART algorithm.

"ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data. C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan's latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node."[14]

Parameters discussion

The model parameters should be chosen carefully in order to avoid the disadvantages of the Decision Trees. As we stated earlier, they might be accurate and yet simple models but they must be used with care.
We present the first parameters we tuned

- 'splitter': either best or random, the strategy used to choose the split at each node.
- 'max_depth': The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- 'min_samples_split': The minimum number of samples required to split an internal node. We focused on using integers to define the minimum number of samples.

[15] As a side note, we first thought generalization is really important in our case, as rooms might have really different features we ignored: volume, windows, air conditioning and heating,... and many more parameters that could be considered but we decided to exclude in our analysis.

Many more parameters exist but the early machine learning step need to be fast considering the number of techniques involved.

Important remarks

The decision tree model also impacted us in our way of pre processing the data, *sci-kit* yields a warning about this:

"Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant. Class balancing can be done by sampling an equal number of samples from each class, or preferably by normalizing the sum of the sample weights (sample_weight) for each class to the same value. Also note that weight-based pre-pruning criteria, such as min_weight_fraction_leaf, will then be less biased toward dominant classes than criteria that are not aware of the sample weights, like min_samples_leaf".[14]

Since we wanted to build a dataset that could be analyzed by as many machine learning techniques as possible, we decided to perform the "class-balancing" within our pre processing task, by removing the extensive zeroes of the Paul Otlet's room dataset[16].

Furthermore, it is impossible to talk about Decision Trees without giving a word about `Random Forests`. Although *sci-kit* classifies them as ensemble methods, this technique is intrinsically related to Decision Trees and should be detailed in this section.
In fact, Random Forests **are** an ensemble Decision Trees. The Random Forest is built upon a collection of decision trees with random parameters variations.

"In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

The sci-kit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class."[17]

Also, the Random Forest has a **feature_importances** attribute that will be useful in order to determine which feature is good at predicting the number of people in a room and which one is actually useless. This will be important in our retro-active approach, to improve our feature extraction and pre-processing task after we have performed some machine learning experiments.

For more information about the Decision Trees in general, [18], [19], [20] and [21] explore those along with more general principles of pattern classification. [22] expose the C4.5 algorithm more precisely while Breiman in [23] and [24] explore the Bagging Predictors and Random Forests.

Ensemble methods

This part aims at combining several models in order to build a more accurate and reliable one. We will often refer to this method as making a `Voting model`, as we will use the fittest models for more in-depth study before combining them.
Note that we will focus on `Bagging Methods/Voting procedures` rather than boosting. We

[16]Remember that the nature of the data acquisition of the Parnas room prevented us from this problem at first hand.

[17]From *sci-kit* documentation, `http://scikit-learn.org/stable/modules/ensemble.html`

believe that the models we will use are already complex and do not need to be improved this way. Furthermore, it could increase the overfitting trait of some techniques.

The `Bagging Methods` are defined as followed in *sci-kit*:
`Bagging Methods` uses several models and *"aggregate their individual predictions to form a final prediction"*.[17]
This forms a *"way to reduce the variance of a base estimator"*.[17]
"bagging methods work best with strong and complex models (e.g., fully developed decision trees), in contrast with boosting methods which usually work best with weak models (e.g., shallow decision trees)."[17]

The last statement explains why we omitted to use the boosting methods. Since we have a kind small dataset (compared to others), our objective is to build several complex models, find the optimal ones, and group them in order to build a super-classifier that would be accurate for our task.

Voting classifiers are interesting, the idea is (in the context of classification) to group several techniques and have them vote for the class that should be assigned to a new, unseen sample. *Sci-kit* makes a distinction between two categories:

- Hard voting classifiers: every model in the list has one vote, and the class that has the greatest number of approvals is selected to label the sample.
- Soft voting classifiers: every model has votes according to its accuracy on the training set, the models that performed better are given more votes than the others.

The Soft Voting technique seems more reasonable at first glance. However, we, once again, must be careful about overfitting: models that perform well on the training set might not generalize correctly. For instance, one model could have a really high accuracy on the samples of a given room but drop drastically on another one. A possible method to adapt this algorithm to the regressors would be to round the numbers yielded by them and assign 1 class per possible integer.

Multi-layer Perceptron (MLP)

The last model we want to introduce is a kind a what is called "neural network". In fact, it is "only" an extension of the Perceptron algorithm we explained above. The magic behind this approach is that applying several linear transformations one after another can actually result in a function that is non-linear, thus yielding a model that can fit a non-linear decision boundary. As *sci-kit* states: *"MLP can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers"*.[18]

"The advantages of Multi-layer Perceptron are:

- *Capability to learn non-linear models.*
- *Capability to learn models in real-time (on-line learning) using partial_fit.*

The disadvantages of Multi-layer Perceptron (MLP) include:

[18]From *sci-kit* documentation, `http://scikit-learn.org/stable/modules/neural_networks_supervised.html`

- *MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.*
- *MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.*
- *MLP is sensitive to feature scaling".*[18]

Since we have scaled the features, we do not have to consider the last disadvantage. The tuning of hyper-parameters can lead to an increased grid-search complexity, we decided to impose a tough constraint on the size of the parameters. As we know that even if `MLP` are often considered as good models, they are slow to train. This fact explain our reluctance to perform a wide tuning at first. We also like the fact that this model can actually learn in real-time; This is a good argument to counteract its slow training time. However, we do not see this aspect as important in our case. It might be useful in the future if we were to create a system that automatically adapt to a given customer habits. Yet, in the context of our master thesis, we do not expect our system to evolve with time but to generalize well.

Sci-kit gives some insights about the complexity of `MLP`: *"Suppose there are* n *training samples,* m *features,* k *hidden layers, each containing* h *neurons - for simplicity, and* o *output neurons. The time complexity of backpropagation is* $O(n \cdot m \cdot m \cdot h^k \cdot o \cdot i)$ *, where* i *is the number of iterations."* This formula gives a more precise information about the complexity of "neural networks".

This ends the overview of the machine learning methods. The *sci-kit* documentation gave some insights about how we should pre-process our data and what we could expect for each technique.

As a conclusion of this section, we noted these important aspects to keep in mind (or to correct in the pre-processing and feature engineering steps) while searching a good machine learning model for our task:

- Class balancing
- Feature scaling
- Avoiding correlated input features
- Avoiding high-dimensionnal input space
- Avoid overfitting through parameter tuning, cross validation and using models with care
- Being aware of the complexity of some models
- Knowing the hypotheses behind every model (the `inductive bias`)

3.1.5 Evaluation metrics used

In this section, we describe how we evaluate the performance of the models. Although classifiers and regressors are evaluated differently in the literature, it is hard to tell which method has an advantage/is more accurate than the other if different metrics are used. For this reason we applied some (custom) classifier metrics to regressors and vice-versa. In the metrics we defined ourselves, predictions are rounded to the nearest integer value. Here are the metrics and a brief explanation if required:

- Strict accuracy: if the exact number of people is predicted the sample is considered as correctly classified, incorrect if not. The accuracy is given by $\frac{\#\text{of correct predictions}}{\#\text{of samples}}$.
- Threshold accuracy: follow the same principle as strict accuracy but here we allow a difference of a few persons: the sample is said correctly classified if $|prediction - actual| < 3$ The value 3 is also arbitrary and follows what we found in [4]. We define the next metric to alleviate the "arbitrary aspect" of this one.
- Proportional threshold accuracy: is computed the same way as the classic threshold accuracy but this time, the value is a fraction of the actual number of people in the room: for instance, when there are 0 people, the threshold will be 0 but when there are 10 people, we could allow an error of 2-3 persons. Following this reasoning we set the threshold to 35% of the actual number of people.
- Mean square error $\text{MSE}(\hat{\theta}) \stackrel{\text{def}}{=} \mathbb{E}\left[(\hat{\theta} - \theta)^2\right]$.
- Median absolute error $\text{MAD} = \text{median}\left(|X_i - \text{median}(X)| \right)$.
- R2 score which is 1 minus the ratio between the residual sum of squares and the total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

Where $e_i = y_i - f_i$ that is, the prediction error:

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

The paper [4] gives the following information about the metrics they used (related to strict and threshold accuracy): *"For all the four estimators, the 0-tolerance accuracy, i.e. the (strict) accuracy, is no more than 50%, which is mainly contributed by the correct occupancy detection when the room is empty, i.e. early morning and around midnight. In the day time, to estimate the exact number of indoor occupants is still a great challenge. However, for such a large office with a maximum of 35 occupants, three to four mis-estimated occupants has insignificant influence on decision-making of air-conditioning 10 and lighting systems. We can see from Fig. 3.2 that three and four-tolerance accuracy is up to 89% and 94%, respectively"*[19]

Note that the 3 first metrics are **accuracy metrics** whereas the 3 last are **error metrics** thus their interpretation differ and, obviously, we want the **accuracy metrics** to be high and close to 100% whereas the **error metrics** should be low (in absolute value) and close to 0 (1 for the r2 score).

3.1.6 First Results and observations

Some techniques caused memory errors on the first attempt with the (small) Parnas dataset with 7 features:
`ARDRegression` and `GaussianProcessRegressor`. Although we found strange to have that kind of error with "so little" data, we did not investigate the problem much because we decided to spend our time on other promising techniques.

[19] `https://arxiv.org/pdf/1607.05962.pdf`

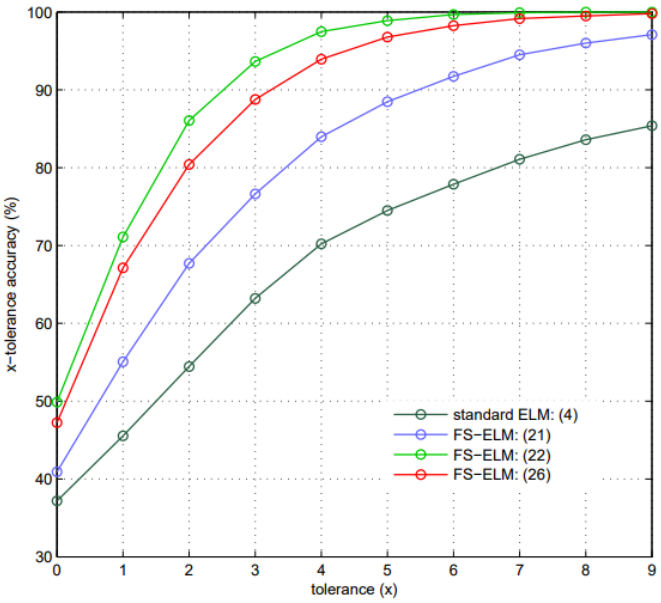


Fig. 7: The *x-tolerance accuracy* of the estimated five days' occupancy level.

Figure 3.2: Image taken from [4] showing the x-tolerance accuracy, the authors are using different models of X-treme machine learning.

The `OrthogonalMatchingPursuit` is also particular because it assumes the input features are not correlated. Which is not our case since we added a *smoothed and derived CO2* along with the original values. The technique thus possibly ends before convergence/required precision is met.

The `LDA` and `QDA` showed similar problems, saying it ended before convergence was met.

The `SVM regressor and classifier` along with the `LinearSVR, NuSVR` showed a quite slow training time, ranging in the 3-5 seconds for the training of **one model**. Thus the grid search is quite slow and sometime, we have to perform cross-validation only without parameter tuning for this reason.

The `MLP regressor and classifier` are also quite slow (as expected about "neural networks"[20]), thus we avoided using grid-search and used the default parameters.

The `BayesianRidge` and `SGDRegressor` also appeared to be slow. A limited parameter tuning has been performed for them.

Up to now, we have only discussed about the performance of the algorithms, their efficiency. Now we will look to their effectiveness (we could allow more time for a model to train if it outperforms significantly all the others). The program output a boxplot for each technique/metric possible pair, these are grouped into a single figure per algorithm for practical reasons. Those boxplots allow us to see the mean score and variance of each technique (the computation of those values are detailed in the section 3.1.1).

As the parameters are computed several times for each model and metrics, it allows us to check whether a model changes its parameter drastically in function of the metric and thus see whether we should be worried about overfitting.

[20]One of our machine learning professor do not really agree with this term since it involves only applying several linear transformation in series.

First, we can observe some general trends in many techniques before going into more details:

- As expected and by definition, for every model, the `threshold accuracy` is always higher or equal than the `strict accuracy`.
- Overall, the models do not perform that well with the `strict accuracy` metric with accuracy generally lying around 30-50%.
- However, the models generally perform quite well when using the `threshold accuracy` metric. With accuracy generally ranging around 70-90%.
- The `proportionnal accuracy` lies between the two previous metrics, this metric is generally around 60-70%.
- The other metrics (that are classically used for regression problems) are a bit less interpretable. However, we can deduce that in general, the models have their error metrics quite low. We also see a good correlation between the `strict accuracy` and the `r2 score` in general.
- Some classifiers outperforms their regression counterpart (such as the `Random Forest` and the `Decision tree`).
- Some (rare) models have a really poor accuracy (the best example being `Lasso`).
- A few models have a quite good `strict accuracy` such as the `decision tree` and `random forest classifiers`.
- Many of the models have a low variance in their results, even using only 5 folds in this case. Some are impressively constant in their accuracy such as the `Random Forests`.
- Although our objective is to maximize the `strict accuracy`, it is interesting to see that some models that performed badly for this metric actually have nice or less observable results with regression metrics.

Note that the scales for the regression metrics are large in order to spot model instability at this point. We refined the evaluation of those metrics on more accurate models.

Here we provide boxplots of some of the techniques. Those were trained on 90% of the data of the Parnas room only (around 7500 training samples) and tested on a testing set composed of the 10% remaining (chosen randomly). The training phase follows the protocol we established earlier, as a reminder, we choose the best parameters with a grid search and a cross-validation, we then build a model with these parameters on the training set and evaluate its accuracy on the test set. We perform this operation for each scoring technique to see if we note some discrepancies between those.
Note that if some parameters are not mentionned, it means we did not tuned them and let the default value from *sci-kit.* either because we wanted to avoid using unexpected/irrelevant values or to reduce the size of the grid search.

In figure 3.3, we can see the performance of the *Lasso* technique. It performs kind of badly on the Parnas dataset for the strict accuracy metric. We discarded many models yielding similar first results. The best parameters we found for that technique are: {'alpha': 1, 'max_iter': 100, 'tol': 0.001}. In figure 3.4, we can see its leaning curve. It seems that increasing the number of training examples has no significant effect on the accuracy.

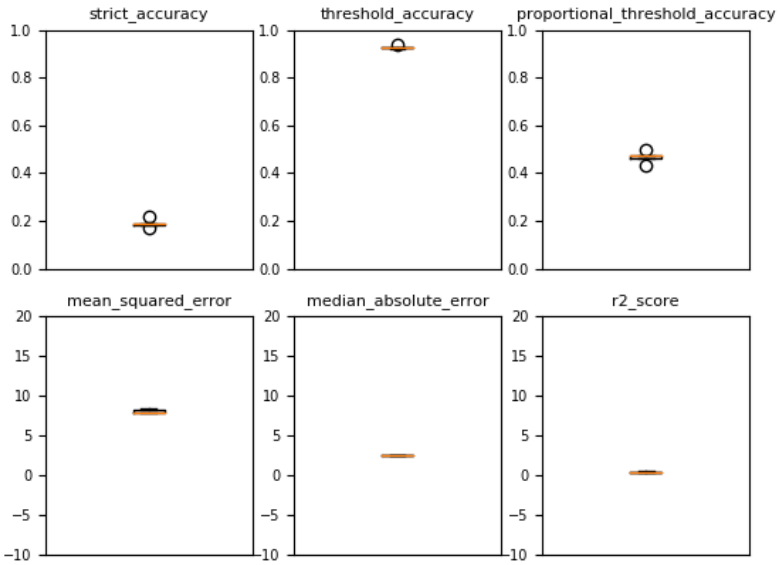


Figure 3.3: Boxplot of the different scores of the `Lasso` technique on the Parnas dataset.

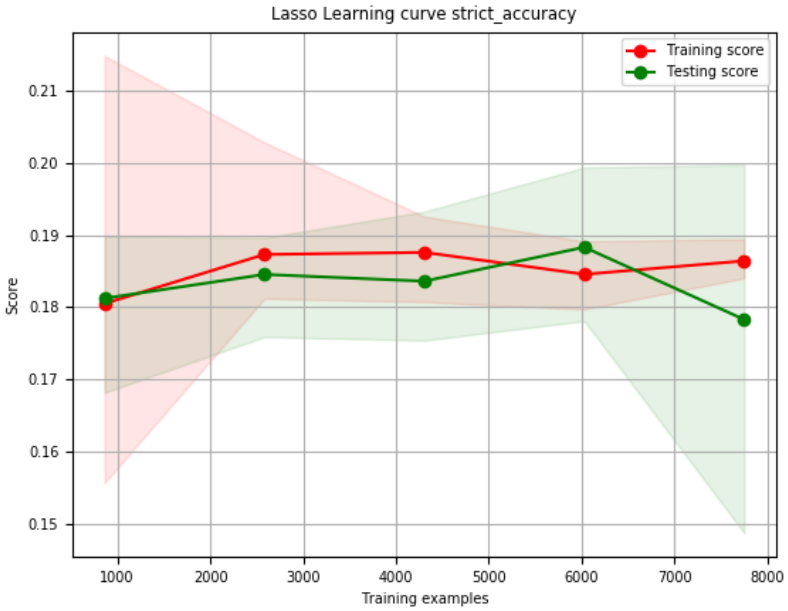


Figure 3.4: The Learning curve for the `Lasso` technique on the Parnas dataset with the strict accuracy metric.

In figure 3.5, we can see the performance of the `Bayesian Ridge` technique. It has a pretty bad (although not terrible) strict accuracy but, given some tolerance (as in the threshold accuracy), perform quite well. We also note good scores for regression metrics where lower (absolute) value is quite good.The best parameters we found for that technique are: {'alpha_1': 1e-07, 'alpha_2': 1e-07, 'lambda_1': 1e-07, 'lambda_2': 1e-07, 'n_iter': 100, 'tol': 0.001}. Although when trained with regressing scores some parameters were different ('alpha_1': 1e-05 and 'lambda_2': 1e-05).

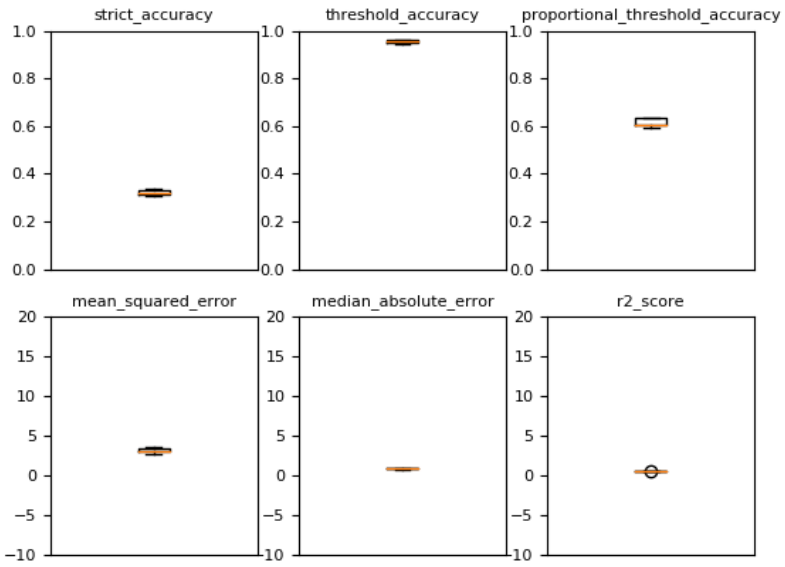


Figure 3.5: Boxplot of the different scores of the `Bayesian Ridge` technique on the Parnas dataset.

In figure 3.6, we can see the performance of the *Random Forest Regressor* technique. It yield good results compared to many other techniques. Concerning the used parameters, we observe some discrepancies for the n_estimators (100, 50, 10, 10 and 200 for respectively strict, threshold accuracy, mean square, median absolute error, r2 score). At first, we only tuned this one parameter to reduce the computation time of the grid search.

In figure 3.7, we can see the performance of the *Random Forest Classifier* technique. This is our good surprise. The Random Forest Classifier outperforms slightly its regressor counterpart (for the strict accuracy), even though we are facing a regression problem. This can be due to the fact that we are considering small rooms that generally have a limited number of occupants. Here also, we find differences in the best parameters for the n_estimators. The `Random Forest Classifier` yields an increasing learning curve (see figure 3.8). Even with a small portion of the training set, the model seems to have a quite nice accuracy ($\pm 70\%$). And the accuracy continues to rise as the training set grows. It even looks like the performance could be improved with a bigger dataset.

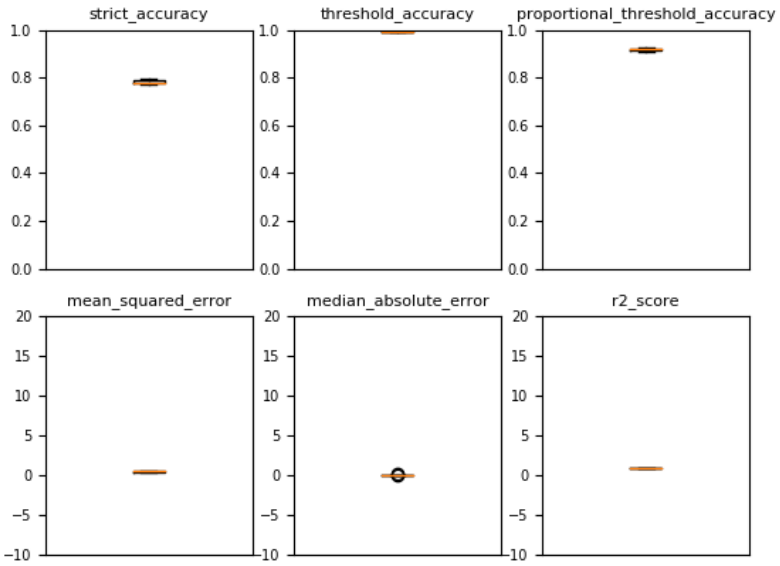


Figure 3.6: Boxplot of the different scores of the `Random Forest Regressor` technique on the Parnas dataset.

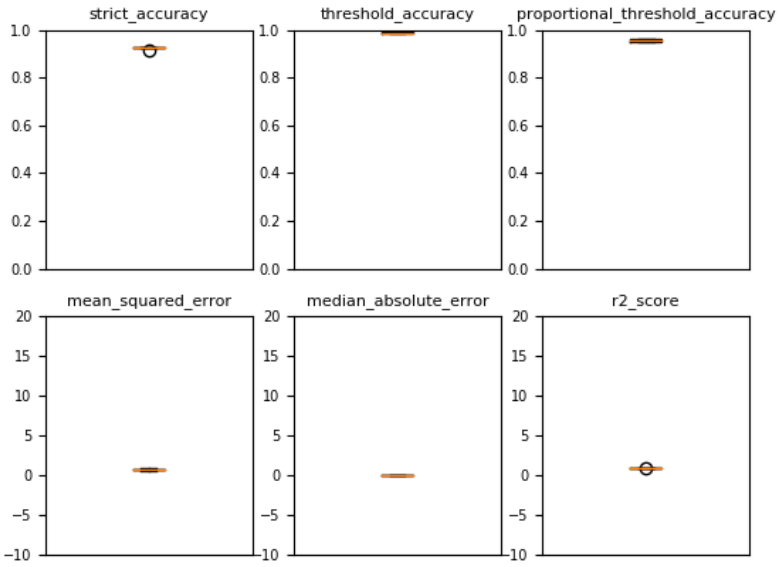


Figure 3.7: Boxplot of the different scores of the `Random Forest Classifier` technique on the Parnas dataset.

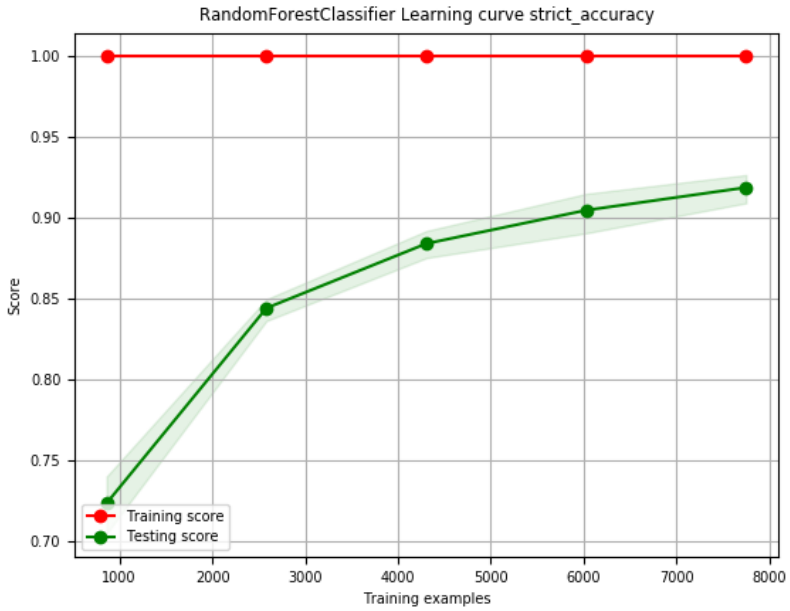


Figure 3.8: The Learning curve for the `Random Forest Classifier` technique on the Parnas dataset with the strict accuracy metric.

As a side remark, we observed similar results than the Random Forests with the `Decision Tree Regressor` and the `Decision Tree Classifier` but with higher variance and a slightly lower accuracy.

In figure 3.9, we can see the performance of the `KNNeighbors regressor` technique. It is quite efficient, it has a good accuracy and a small variance. Concerning the used parameters, the regressor chooses: {'n_neighbors': 1, 'weights': 'uniform'} for *strict accuracy* and *proportional threshold accuracy*, {'n_neighbors': 7, 'weights': 'distance'} for *threshold accuracy* and *r2 score* while giving {'n_neighbors': 19, 'weights': 'uniform'} for *mean square* and *median absolute error*. The number of neighbors in the two last cases comfort us in our idea that this technique will avoid overfitting in our application. However, the *strict accuracy* and *proportional threshold accuracy* chose a n_neighbors': 1 which might indicate overfitting for those two metrics. Nonetheless, we think some parameter tuning in the next section might lead to more interesting results. In figure 3.10, we can see its leaning curve. The accuracy is quite good even for small training set size ($\pm 60\%$). And it rises progressively as the training set size grows. We can also expect the accuracy to rise even further for bigger training sets.

As another side note, we observed very similar results for the `KNeighbors classifier`. By seeing the pictures, it is even hard to tell which one belong to which model. However, the classifier uses as best parameters: {'n_neighbors': 1, 'weights': 'uniform'} for all metrics except the mean squared and median absolute error where it uses 19 neighbors such as the regressor. Thus, the regressor might be more suitable to avoid overfitting.

Now, we will analyze more in depth the `Multi Layer Perceptron`, because it yields some interesting results and we will especially compare the regressor to the classifier.

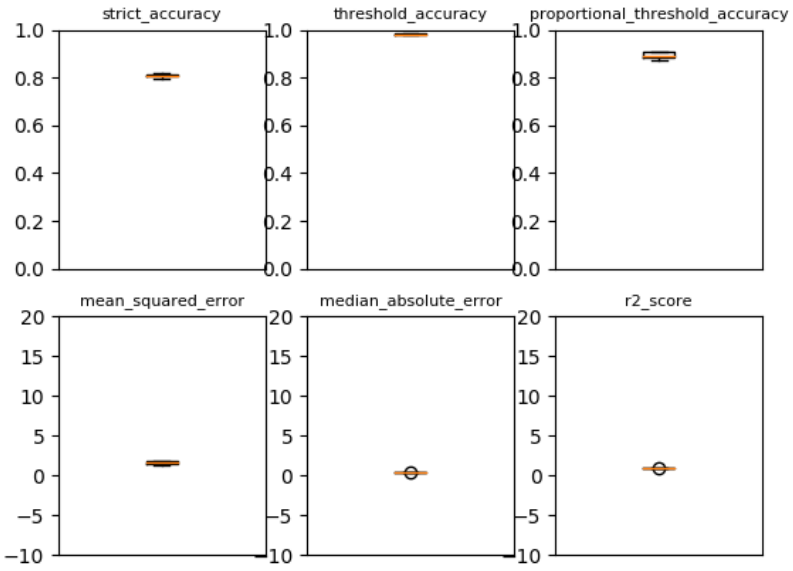


Figure 3.9: Boxplot of the different scores of the `KNNeighbors regressor` technique on the Parnas dataset.

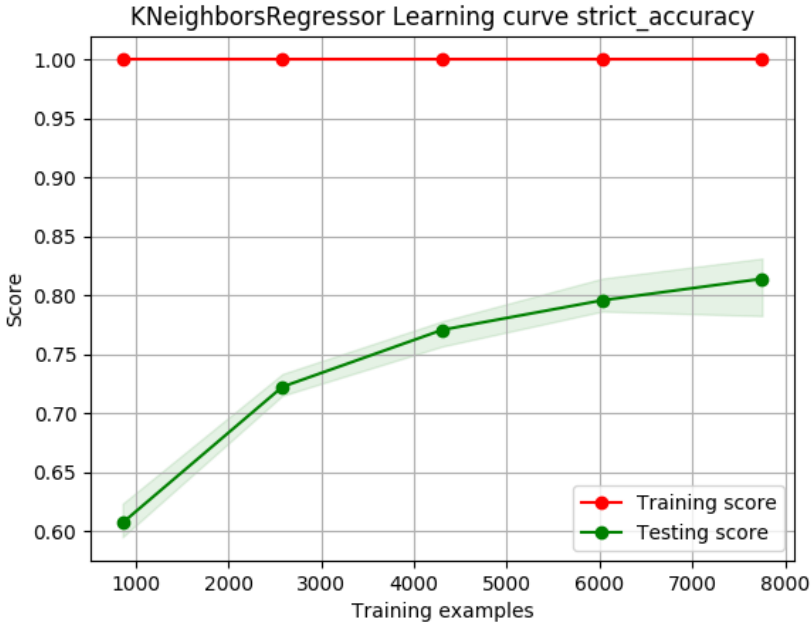


Figure 3.10: The Learning curve for the `KNNeighbors regressor` technique on the Parnas dataset with the strict accuracy metric.

In figure 3.11, we can see the performance of the `Multi-layer Perceptron regressor` technique. It does not have a significant advantage over most of the other techniques. However, in this case we did not tune any parameter, we just kept the *sci-kit* default values, to yield this picture. We believe its performance might increase by giving some freedom about its structure. In figure 3.12, we can see its leaning curve. It is interesting in the fact that both the training and testing accuracy improve with the number of samples. This might be a good indicator that this technique generalize well to our problem. We computed in figure 3.13 the accuracy to predict zero values and non-zeros values for the `Multi-layer Perceptron regressor`. What we see was expectable, the algorithm is better at predicting correctly that there is nobody in a room than predicting the correct number of people while there are some. We did not showed the zero vs non-zero for other algorithm because the deductions we made from them are the same as in this figure.

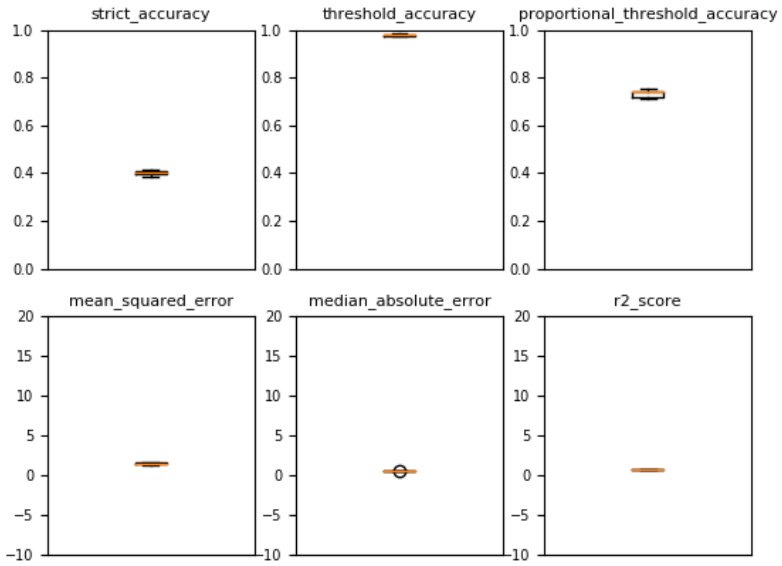


Figure 3.11: Boxplot of the different scores of the `Multi-layer Perceptron regressor` technique on the Parnas dataset.

In figure 3.14, we can see the performance of the `Multi-layer Perceptron classifier` technique. We did not set up a parameter grid search to build this model (for the same reasons than the `MLP regressor`). We also see that the classifier outperforms its regressor counterpart. In figure 3.15, we can analyze its leaning curve. We clearly see that with a larger dataset, we might further improve the `MLP classifier` accuracy. We computed in figure 3.16 the accuracy to predict zero values and non-zeros values for the `Multi-layer Perceptron classifier`. We observe the same phenomena as for the regressor, but with a higher overall acccuracy and lower variance. Interesting fact, the difference between the zero and non-zero accuracy remains at around 20%.

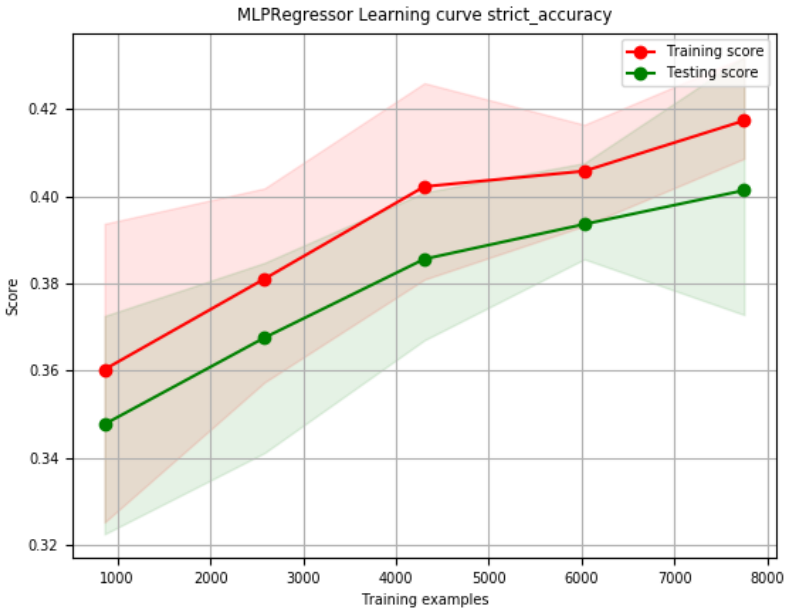


Figure 3.12: The Learning curve for the `Multi-layer Perceptron regressor` technique on the Parnas dataset with the strict accuracy metric.

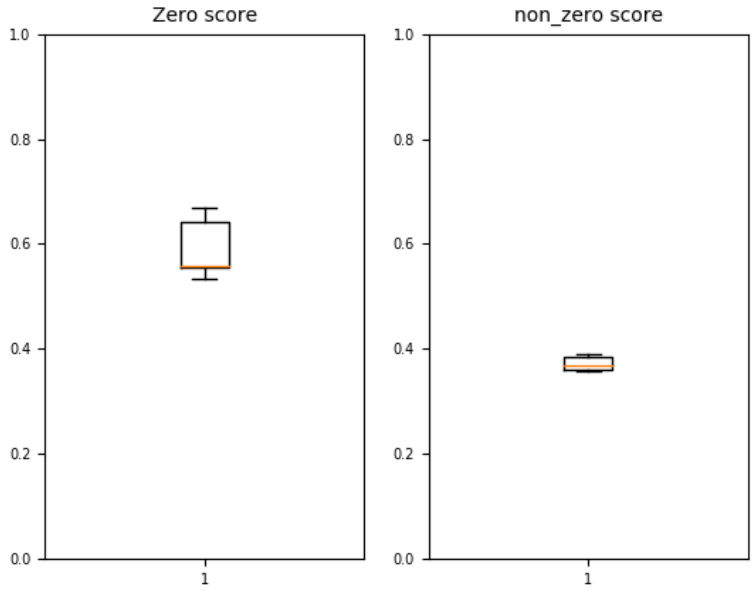


Figure 3.13: Zero vs non zero scores for the `Multi-layer Perceptron regressor` technique on the Parnas dataset with the strict accuracy metric.

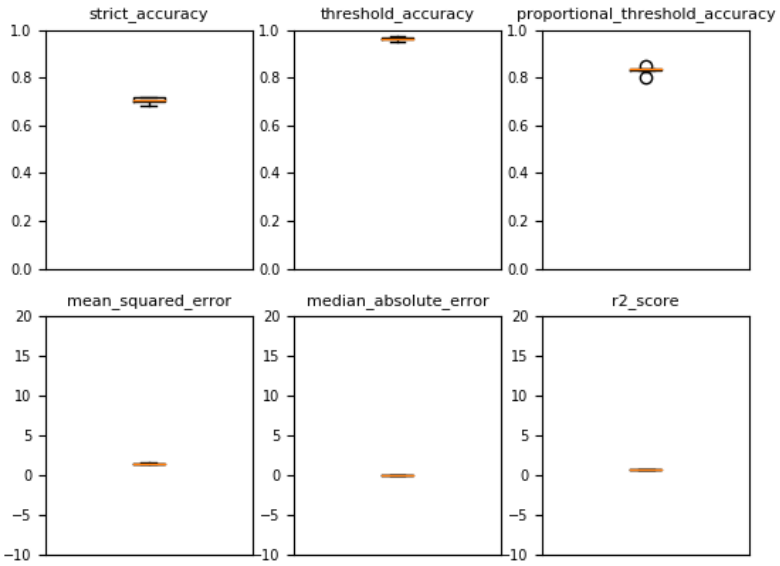


Figure 3.14: Boxplot of the different scores of the `Multi-layer Perceptron classifier` technique on the Parnas dataset.

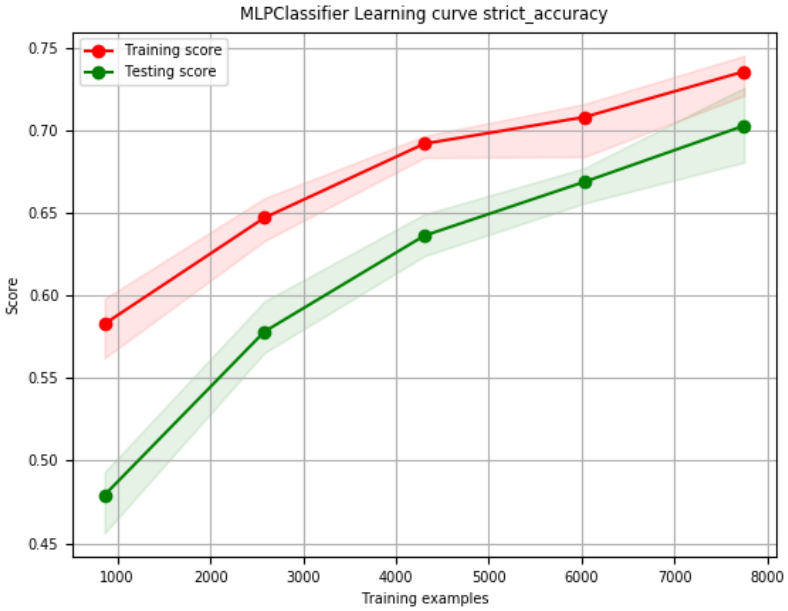


Figure 3.15: The Learning curve for the `Multi-layer Perceptron classifier` technique on the Parnas dataset with the strict accuracy metric.

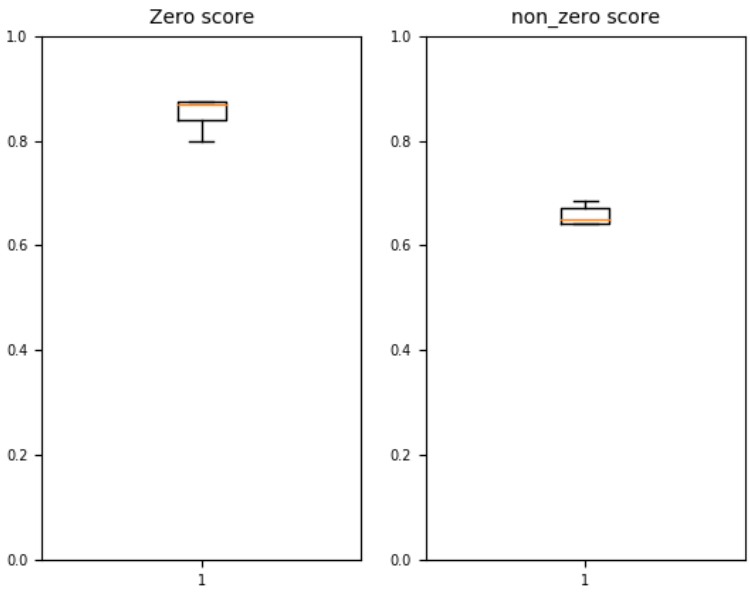


Figure 3.16: Zero vs non zero scores for the `Multi-layer Perceptron classifier` technique on the Parnas dataset with the strict accuracy metric.

These results are interesting because we did not expect a classifier to significantly outperform a regressor in our problem, that we consider more as a regression (giving a rough estimate of the number of people) than classification which cannot distinguish beforehand which class is closer to another. In the case of the KNeighbors, the results were surprisingly similar but one should not reasonnably choose the `KNN classifier` over the `KNN regressor` because of the hyper-parameters. We will conduct more in depth parameter tuning for both MLP models to see if one still outperforms the other.

Since we decided to analyze so many models and our objective in this part is to have a first glance on those, a deep analysis is outside the scope of this work.

The next step is to select the seemingly most promising models, deepen the range of hyper parameters and combine them to create a single super-model that would ideally have the following properties:

1. High *strict accuracy*.
2. Low error rates for the regression metrics (and around 1 for the r2 score).
3. Low variance in the results (for us, telling the accuracy is between 75-85% is better than between 70-90% even if the mean is the same, meaning we also consider the worst case scenario).
4. Our model should generalize well, possibly be usable for different rooms and not overfit the training data.
5. Low computational cost or at least where the grid search for parameter tuning is done in a reasonable amount of time.
6. Ideally, our model should be understandable and explainable so that we can actually see what is happening and which feature is more related to the person count.

These properties are sorted by priority according to our experience and needs. Of course, we assume that it will be impossible to meet all these conditions thus the ranking is important to guide our best-first search approach. The next step is to finally select the models we `feel, think, see,...` are the more suitable for the task.

3.1.7 Feature importance

Another interesting metric to help us understand how helpful the environmental data was to predict the number of people in a room is the feature importance. For each machine learning model, we can compute how helpful one feature is to the prediction by removing it and observing the effect on the testing score.

The algorithm we used to compute the feature importance is the following (see listing 2). For each feature we train the model on the training set in which the feature was removed. Then we compute the prediction on the test set and the score of the prediction. We do that computation several times (we chose to do it 5 times, like for the grid-search cross-validation) with different random partitions between training and testing sets to have a good approximation of the mean score for each feature.

The importance of each feature can then be computed as the difference between the mean score of the prediction with all the features and the mean score of the prediction with one feature removed divided by the mean score of the prediction with all the features. So the importance of a feature is the part of the score that is lost when the feature is not used for the prediction.

Because *co2* and *co2_smoothed* features are strongly correlated (see section 2.5), we removed the two features together when computing their importance because when removing only one, the model can still have the information the feature is carrying in the correlated feature.

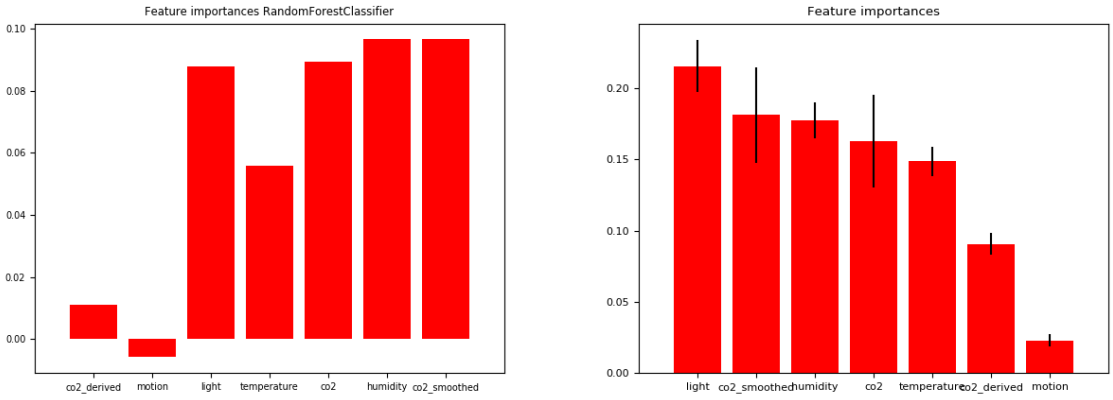
```
for iteration in range(5):
    (X_train, Y_train, X_test, Y_test) = random_partition(data)
    for feature in features:
        remove_feature(feature, X_train, X_test)
        model.fit(X_train, Y_train)
        Y_pred = model.predict(X_test)
        score[feature][iteration] = scorer(Y_test, Y_pred)
for feature in features:
    importance[feature] = (base_mean_score - avg(score[feature])) / base_mean_score
```

Listing 2: Pseudo code of our algorithm to compute the importance of a features for one machine learning model.

For the `Random forest` model, the importance of each feature can be retrieved from the model itself. In figure 3.17 you can see the two feature importance (the one computed with our algorithm on figure 3.17a and the one of the `Random Forest` model attribute on figure 3.17b). The two importance scores are quite similar as the four most important features highlighted for the `Random Forest classifier` are the *light*, *humidity*, *co2* and *co2_smoothed* and the two less important feature are also the same for the two importance scores (*motion* and *co2_derived*).

In figure 3.18, you can see the feature importance of two *random forest classifier* fitted with different scoring metrics, the *threshold accuracy* (figure 3.18b) and the *proportional threshold accuracy* (figure 3.18a) (see section 3.1.5 for details about those metrics). You can see that the feature importance score change with the considered scoring metric but the comparative influence seems to be conserved[21] (important/unimportant features with one metric are also

[21] The same observation was made with most of the other machine learning technique we used.

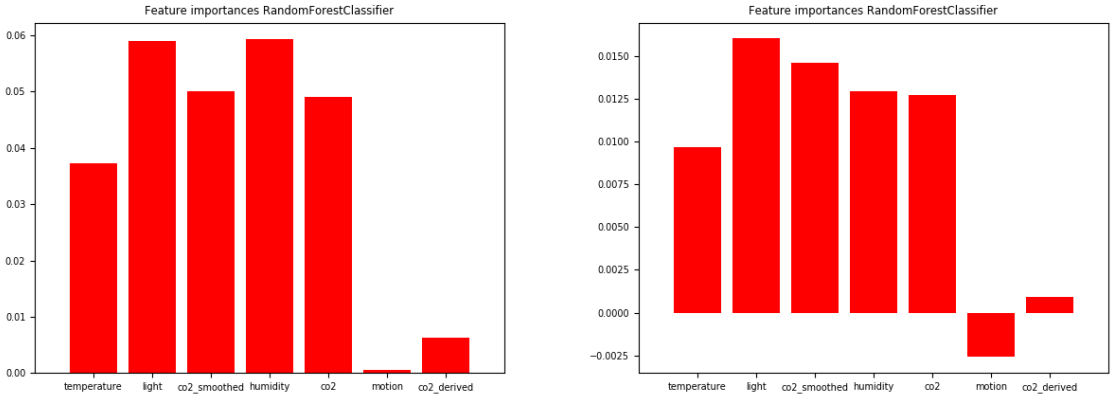


(a) Our computed feature importance.

(b) The feature importance of the model.

Figure 3.17: Feature importance of the `Random Forest classifier` with *strict accuracy* scorer.

important/unimportant with other metrics). It means that the scoring metric does not influence that much which features are used by the model.



(a) Computed with proportional threshold scorer.

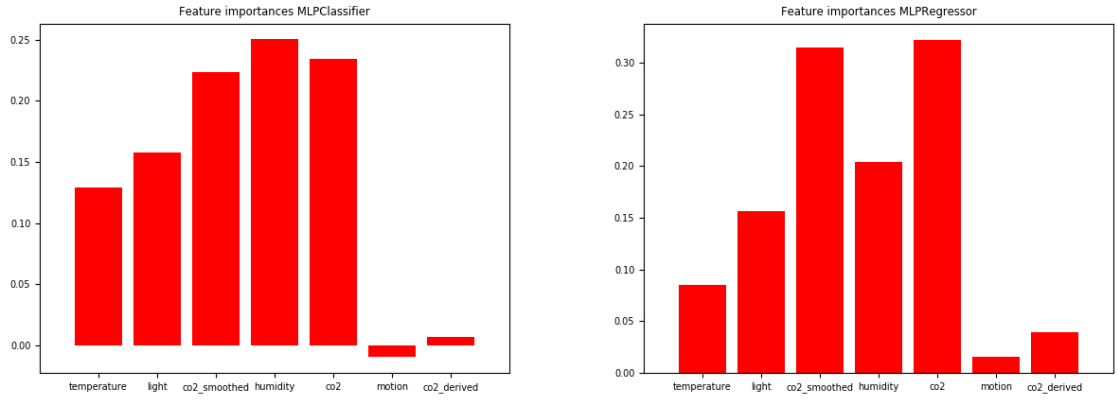
(b) Computed with fixed threshold scorer.

Figure 3.18: Feature importance of the `random forest classifier`, *fixed threshold* vs *proportional threshold*.

In figure 3.19 you can see the feature importance of two `MLP (Multi-layer Perceptron)` models, one classifier (figure 3.19a) and one regressor (figure 3.19b). You can see that the feature importance of the two models are quite similar[22] which means that the models use the same features to make their predictions. So in term of features used there seems to exists no major differences between the classifier and the regressor while in term of accuracy, the `MLP classifier` performs much better than its regressor counterpart ($\pm 70\%$ vs $\pm 40\%$).

With the feature importance we can also try to see if there are some features that have to be used in order to have a good accuracy. In figure 3.20, you can see the feature importance of two models that performed well on the parnas dataset ($\pm 80\%$ strict accuracy) and on figure 3.21, you can see the feature importance of two models that performed badly on the parnas dataset ($\pm 20\%$ strict accuracy). We can observe that the models that performed well use the same features to make their prediction (*co2*, *co2_smoothed*, *light*, *temperature* and *humidity*) while avoiding to use

[22]The same observation can be done on other machine learning models having a classifier and a regressor like the `K nearest neighbour` or the `random forest`.

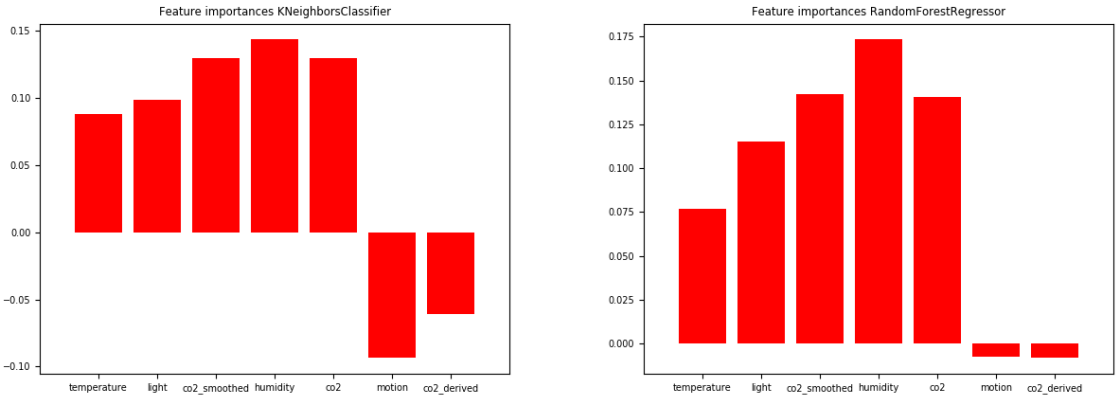


(a) The classifier.

(b) The regressor.

Figure 3.19: Feature importance of `MLP` models fitted with *strict accuracy* metrics, classifier vs regressor.

the two last features (*motion* and *co2_derived*). The models that performed badly on the other hand do not seem to use the same features, one uses only the *co2* and *co2_derived* features while the other poor model uses all features but *light* and *humidity*. So it seems with those examples that for a model to perform well, it has to use all but the *motion* and *co2_derived* features.



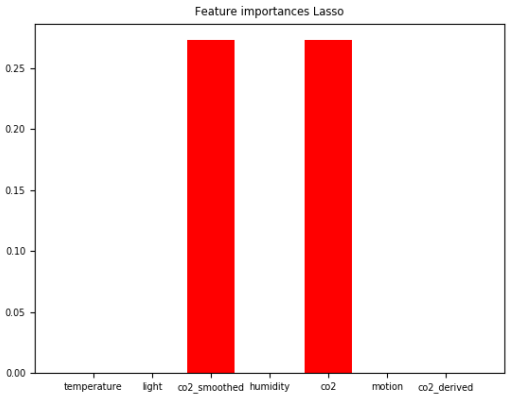
(a) The `K nearest neighbour classifier`.

(b) The `random forest regressor`.

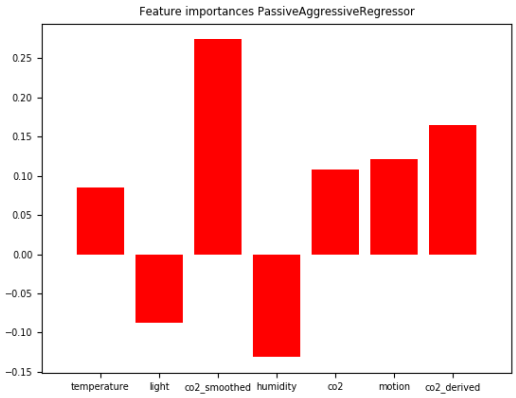
Figure 3.20: Feature importance of models that performed well (computed with *strict accuracy* metric).

So it appears that the feature we added to the dataset in the feature engineering part (*co2_derived*) does not give useful information to the models that perform well. But it could also be due to the limits of the feature importance metric we defined. Indeed the metric only tells us what happens to the accuracy when the feature is removed but does not tell us if the model uses it when it is present.

Let's introduce another feature importance metric, that we called singe feature importance, to know whether *co2_derived* carry useful information. This second feature importance metric is quite similar to the first as it just take one feature instead of removing only one. The feature importance score is then the accuracy of the model trained on the feature alone divided by the accuracy of the model trained on all the features.



(a) The `lasso` linear model.

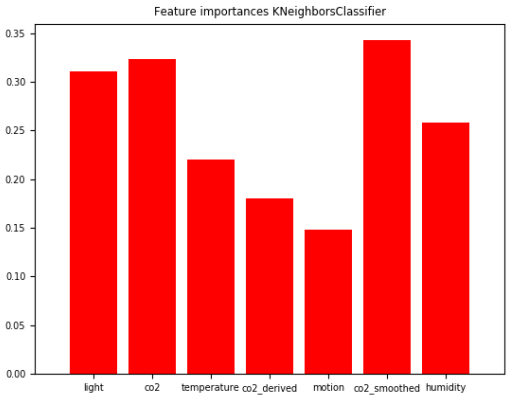


(b) The `passive aggressive regressor`.

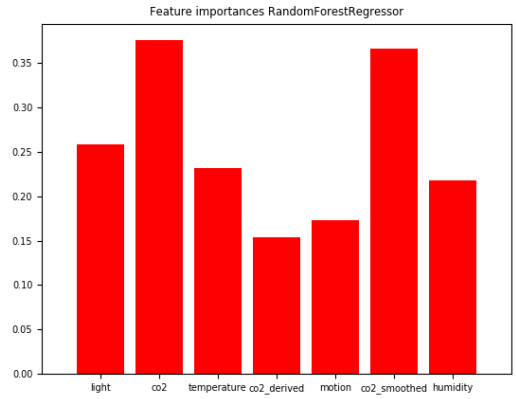
Figure 3.21: Feature importance of models that did not perform well (computed with *strict accuracy* metric).

You can see in figure 3.22 the single feature importance of the two models that performed well that we analyzed earlier. With that single feature importance metric, we can confirm that the two bad features (*co2_derived* and *motion*) do carry useful information as they allow the two models to predict the person count with a strict accuracy of ±15% (±30% with proportional accuracy and ±60% with threshold accuracy).

So we can say that all the features seems to carry useful information about the person count to predict and could be useful to different machine learning models depending on how they work.



(a) The K nearest neighbour classifier.



(b) The random forest regressor.

Figure 3.22: Single feature importance of models that performed well (computed with strict accuracy metric).

3.1.8 Retro-active approach

Even if it has been hidden a bit throughout this session, we used a retro-active approach to tackle this problem: such as the agile approach, we iteratively performed the following:

1. Pre-process the data and extract some features.
2. Define some scoring metrics.
3. Train the models and record their performance.
4. Assess their overall performance, detect outliers, strange behaviors, warnings during computation,...
5. Design new techniques for all the previous steps.
6. Repeat.

It is following this scheme that we progressively adapted our data filtering and scoring metrics, adding the proportional threshold metric for instance.

Now that we have some results, it is time to tune and combine the most promising models.

3.2 Playing with Machine Learning and Democracy

This section is dedicated to the search refinement among the models. We will work iteratively, performing the following:

1. Choose some (3-5) promising machine learning techniques.
2. Verify their best parameters (also to avoid overfitting e.g. a KNN with N=1 is overfitting by definition).
3. Adapt/Increase/Deepen the range of parameters to analyze if necessary.
4. Build the models with their best parameters and assess their performance.
5. Combine the models using ensemble methods and evaluate whether the model fulfills the properties given earlier.
6. Repeat.

3.2.1 Selected models and parameter tuning

Here to deepen our analysis of the "best" models (we double the allowed time for the grid search), we will also analyze in detail those new elements[23]:

- Learning curve for different metrics.
- Performance on the zero prediction, i.e. is the algorithm good at predicting there is nobody in the room.
- Performance on the non-zero prediction, i.e. is the algorithm good at predicting the strict correct number of people whenever there is at least one person. We expect this metric to be harder than the previous one.

We will also analyze later (see section 3.2.3) the performance of our final, Voting, algorithm between rooms: is an algorithm trained on the Parnas room effective for the Paul Otlet one or not? What about an algorithm trained on both rooms?

We decided, following the insights from last section, to choose the following machine learning techniques:

- `Random Forest Classifier`, because of its high stability and accuracy.
- `Random Forest Regressor`, to see if using a regressor and a classifier of the same type can actually increase precision.
- `KNNeighbors`, because its performance is high and it is an algorithm we know quite well (classifier and regressor).
- `MLP`, more by curiosity rather than evidence of effectiveness, and because we believe some more parameter tuning can highly increase performance. Also, we pick this technique because some variations has been correctly executed in [4] (classifier and regressor).

[23] We already analyzed these for some models before but we now will do it systematically.

3.2.2 In-depth parameter tuning

In this part, we will show the results of tuning the parameters of models, to produce the following graphs, we re-defined manually the set of parameters to consider. Furthermore, we discarded some set of parameters that would automatically lead to overfitting: such as setting the number of neighbors to 1 in `KNN`, or the min split to 2 for the `Random Forest`. Here are the parameter sets we considered for the different models[24]:

Random Forest

- 'max depth' of the trees: [20, 30, 40].
- 'min samples split', the minimum number of sample required to make a split in a tree: [10, 20, 30, 50].
- 'n estimators', the number of trees in the forest: [50, 75, 150, 200].

K-Neighbors

- 'n neighbors', the number of nearest neighbors involved in the decision process: range(3, 25, 3) that is [3 6 9 12 15 18 21 24].
- 'weights', the method used to give weights to the nearest neighbors: uniform or distance weighted.
- 'algorithm', the way used to compute which training sample is closest to the new one: [auto, ball tree, kd tree, brute].

MLP

- 'hidden layer sizes', the structure of the neural network[25]: [(100,), (200,), (50,), (25,), (10, 10), (25, 25), (50, 50), (10, 15, 10)][26].

The set of parameters seems limited for each technique, but we wanted the grid search for the best parameters to be under 10 minutes also because we perform some cross-validation outside the grid search. As an indicator, the entire cross validation for the 3 classifiers and the 3 regressors took around 12 hours on a laptop[27].

The parameters were chosen among the best selected by the grid-search. However, those are changing following the metrics and even changing only with using random sampling to build the testing set. Thus, we chose the set based on accuracy and our experience about those models. It also explains why we considered the techniques we know, we were lucky in our work because the algorithms we knew were the most accurate ones.

The following parameters were retained for the voting model:

[24] Again, most of the numbers are arbitrary and based on our knowledge and experience, especially from [7].

[25] Note that there exist other machine learning techniques that allow the structure of the node to evolve to better match the data called NEAT aka NeuroEvolution of Augmenting Topologies see `https://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies` for more information. Also, [2] used this technique successfully for CO2 room occupancy detection.

[26] Each number indicates the number of neurons inside a hidden layer. tuples with several numbers meaning there are several hidden layer one after the other.

[27] Even though there were some breaks between the computations, the logs can attest that they were approximately that long. The tests ran on a Intel(R) Core i5-4210H CPU (2.9GHz), without parallel computations.

- `RandomForestClassifier`: 'max_depth': 40, 'min_samples_split': 10, 'n_estimators': 200.
- `RandomForestRegressor`: 'max_depth': 30, 'min_samples_split': 10, 'n_estimators': 200.
- `MLPClassifier`: 'hidden_layer_sizes': (50, 50).
- `MLPRegressor`: 'hidden_layer_sizes': (50, 50).
- `KNeighborsRegressor`: 'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'distance'.
- `KNeighborsClassifier`: 'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'distance'.

Some parameters seemed to be not really important, such as the max depth and the n estimators of the random forest, because they vary greatly over slightly different data sets (a cross validation sometimes builds a model with three different values). We decided to let a high value for both, this might lead to overfitting and make our model not generalizable to other rooms but at this point, we felt there are so many differences from one room to another that it would be really surprising to have such a good model.

Although we forced some parameters to avoid overfitting, some algorithms still decide (aka have a higher accuracy on the testing set) to take the values that are most prone to this behavior: the best example is the KNeighbors, we ruled out the value of 1 for the number of neighbors, but the technique still chooses the lowest value for this parameter. We believe that being close to the training data is not a problem in our case, as we have seen that even with **really high** (90-100%) accuracy on the training set, it performed quite well on the testing set.

The next step involves combining those (already pretty good) algorithms and see if the accuracy and variance change. Also we want to see how they perform against data from another room, whether it has some reasonable accuracy or not.

3.2.3 Grouping the models

Now, we will group the models we selected in the previous section and assess the Voting classifier accuracy and its variance. As we stated earlier, we ruled out the boosting methods because the techniques we have chosen are already complex enough and have a reasonable accuracy. Thus, we preferred to rely on the voting ones. We also decided to use the hard voting procedure at first because the algorithms have quite similar results and weighting their vote might rather lead to give much power to the overfitting one over the others, which is exactly the behavior we want to avoid by using voting procedures at first.

We considered 2 approaches: first, we rely only on the classifiers and build a `Voting classifier` from the *sci-kit* library. This way we can re-use exactly our framework and apply the techniques we employed for the early machine learning experiments. Note that at this point, we will *change the scale of the metrics in the graphs* in order to analyze the differences of our choices in details

Voting classifier

This model is thus composed of the `Random Forest classifier`, the `KNeighbors classifier` and the `MLP classifier` with the parameters mentioned above.

As we see in 3.23, the different accuracies are pretty good and especially, the gap between the `strict accuracy` and the `threshold accuracy` is narrowed (+-10% of difference). Also we see

a slight decrease in variance of the results. The learning curve in figure 3.24 confirm the trend that more data will improve the classification and the zero vs non-zero scores in figure 3.25 also confirms our expectations. Thus, even if the gain is marginal, it might be interesting to perform such model agglomeration to increase the chances of generalization.

The next step is to add the regressors to the model.

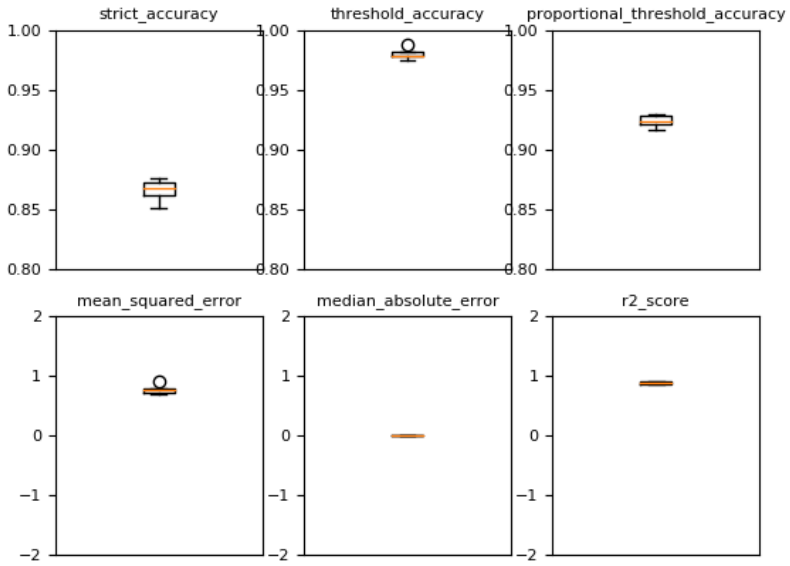


Figure 3.23: General accuracy of the mean-based `Voting classifier`.

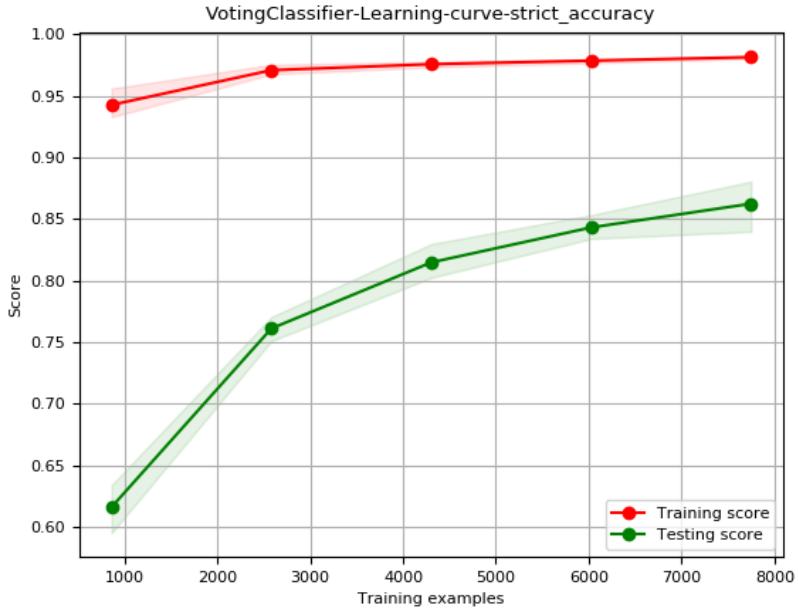


Figure 3.24: Learning curve of the mean-based `Voting classifier` with the strict accuracy metric.

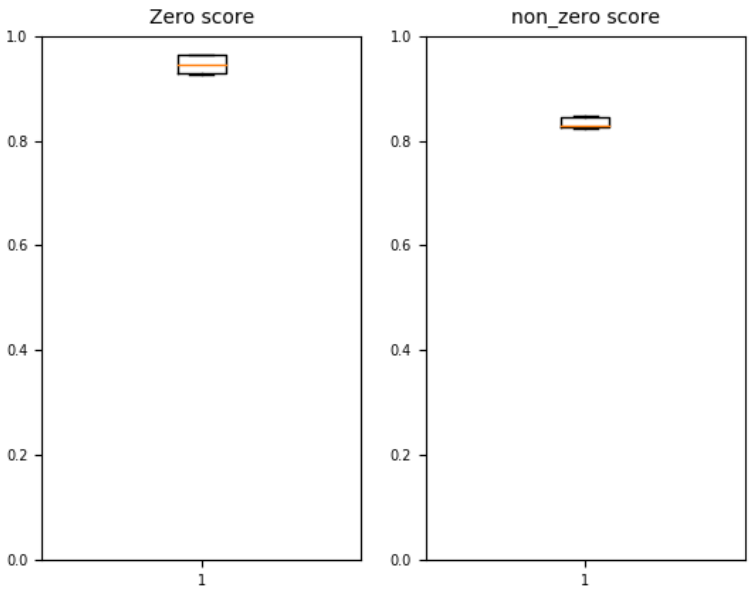


Figure 3.25: Zero vs non zero scores of the mean-based `Voting classifier` with the strict accuracy metric.

Home-made Voting algorithm

Now we will build a model comprising the 6 techniques we have selected. It will then be composed of the same classifiers as the Voting plus their regressor counterpart. However, *sci-kit* does not allow to group several regressors along with classifiers, one must define the classes that should be assigned to a given value in the continuous space. Thus we build our own model along with its `predict` method that we called `predict_voting`. The functionality is similar to the method *predict* of *sci-kit* models, to ease the integration of this new model in our program[28]. The method is just computing the mean of the different models output for each sample to predict.

Here, the results (in figure 3.26) are less expectable and need to be discussed a bit. We see a slight decrease in strict accuracy, more around 75-78% Although it still performs well with the other metrics. This is more visible in the learning curve in figure 3.27. We believe this is due to the fact we added the same regressors as the classifiers with very similar parameters and more importantly to the *way* we included the regressors in the voting procedure. This can actually increase the confidence of the algorithm while it is actually overfitting the data. This is especially true with the regressors, when 2 regressors attempt to predict on the testing data, they might be unsure about the exact number of people and yield values close to x.5 . However, if two algorithm tend very slightly towards a value, it might make the entire decision different and thus lead to a decrease in `strict accuracy`. On the other hand, a regressor might strongly disagree with all the others. In a voting procedure taking the mean of the prediction (the first voting algorithm we made), this can lead one technique to completely change the result and make it false for the entire model.

We thus decided to re-define our mean-based procedure and opt for a more classical voting one. For this, we round the results of the regressors to the nearest integer, then return the value that has to most votes. This new approach should yield similar results than the voting classifier.

[28]Nonetheless, we had to distinguish the two cases several times inside the program because the home-made model is of a different type of object than the sci-kit ones.

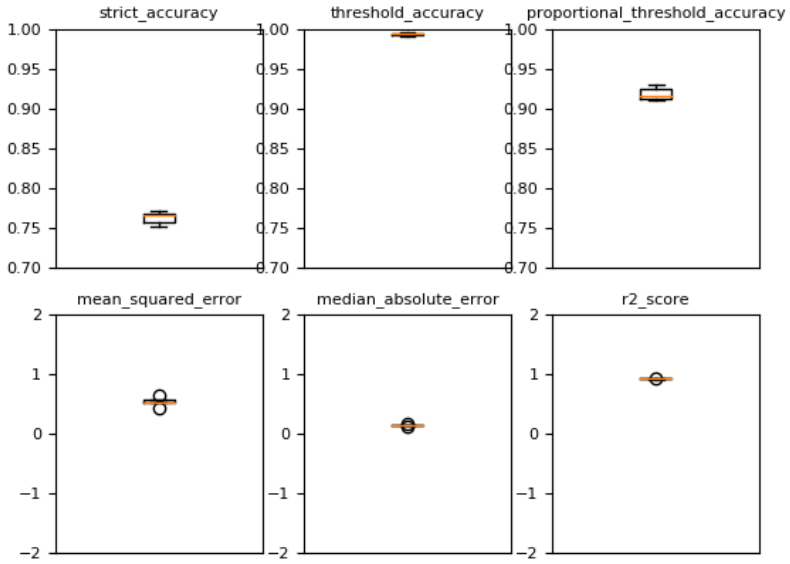


Figure 3.26: General accuracy of the `Voting algorithm`

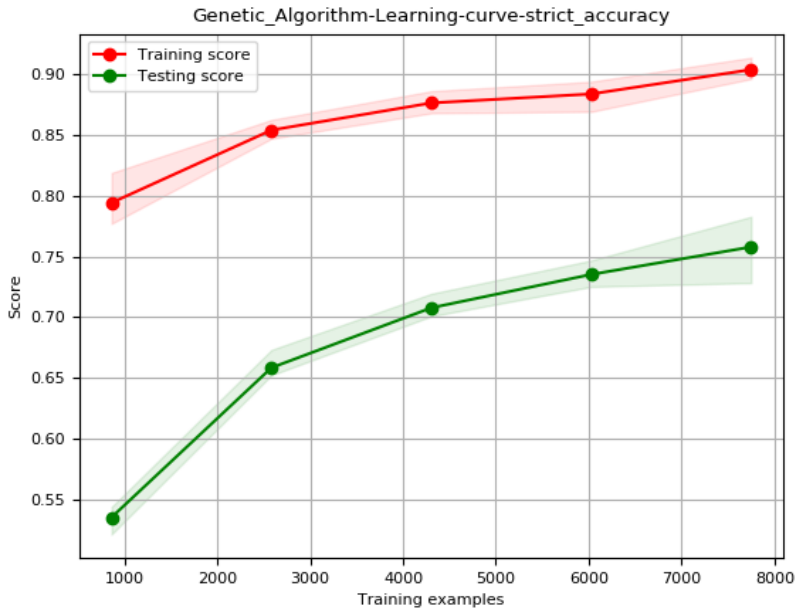


Figure 3.27: Learning curve of the `Voting algorithm` with the strict accuracy metric. Notice the scale for the accuracies has changed in order to show the results correctly

Indeed, we see in figure 3.29 an increase in strict accuracy (85-87%) and very similar results for the other metrics along with their variances. We decided to keep this last algorithm for the remaining of this section to check whether it generalizes well or not.

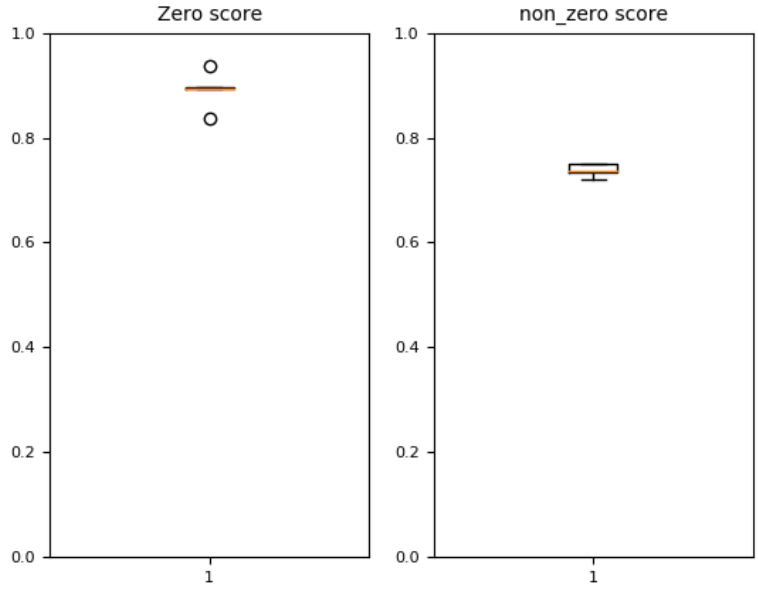


Figure 3.28: zero vs non zero scores of the `Voting algorithm` with the strict accuracy metric.

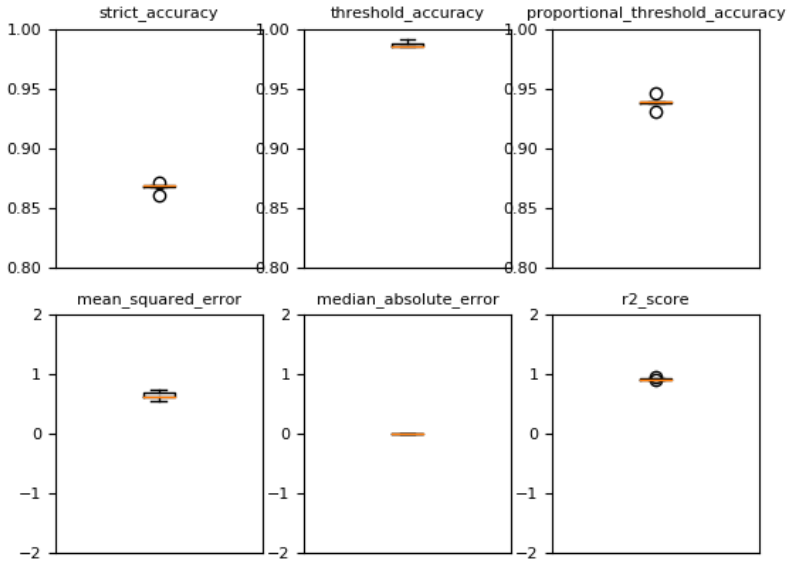


Figure 3.29: Different scoring metrics of the Voting algorithm with the new vote

3.2.4 Between-rooms accuracy

Now, we will see how well our model generalizes to other rooms. First, we will keep the data of the different rooms separated and use one as training set and the other as testing set. Next, we will assess the performance of the algorithm while using both dataset as training and testing (using cross validation like described previously in section 3.2.5).

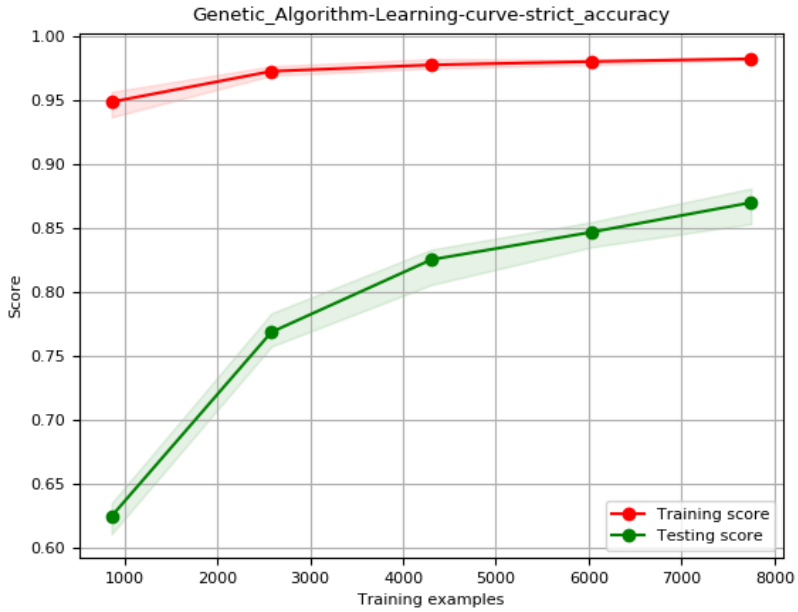


Figure 3.30: Learning curve of the Voting algorithm with the new vote with the strict accuracy metric

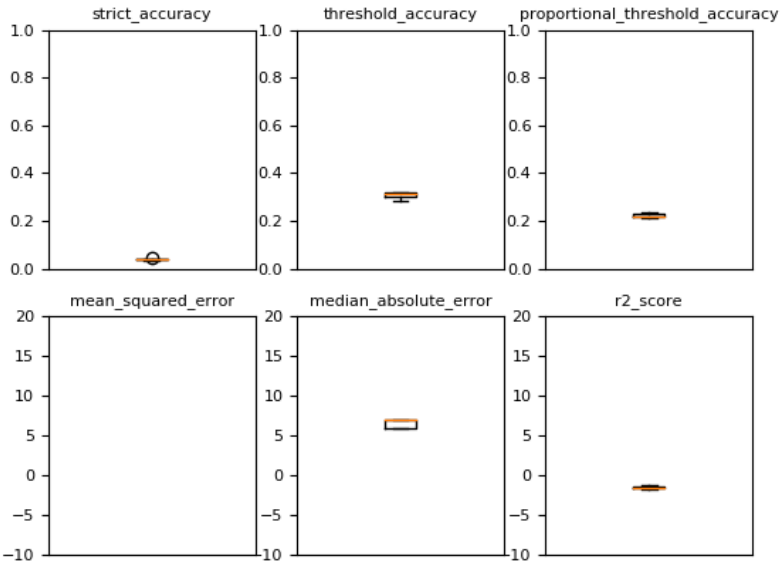


Figure 3.31: Accuracy of the Voting model when trained on Parnas room and tested on Otlet.

As expected, when trained with another room, the model performs badly. The accuracy in figure 3.31 is close to random as we may quickly estimate: while observing our data (in section 2.6), we noticed we had a maximum number of people of 14; thus a classifier, for instance, will face at most 15 possible classes during its training phase. Then, if we assume a model gives a uniformly distributed random guess, the probability to give a correct answer is given by

$$P(\text{Correct guess}) = P(C1)P(sayC1) + P(C2)P(sayC2) + P(C3)P(sayC3) + ...$$

Where $P(C1)$ is the actual probability that a sample belongs to class C1 and $P(sayC1)$ is the overall probability that the model yield a class C1 for a sample to label. If the guess is uniformly distributed, it means $P(sayC_i) = P(sayC_j)\forall i, j$. Therefore the previous equation is equal to

$$P(\text{Correct guess}) = P(sayC1) \cdot (P(C1) + P(C2) + P(C3) + ...)$$

Note that $P(sayC1) = \frac{1}{n\_classes}$ and $P(C1)+P(C2)+P(C3)+... = 1$ because we have considered all the possible classes and that the model is making uniformly distributed random guess.[29] This yields:

$$P(\text{Correct guess}) = \frac{1}{n\_classes}$$

Regardless of the initial class balance. In our case we obtain

$$P(\text{Correct guess}) = \frac{1}{15} \simeq 0.066667$$

Which is approximately what we see in this case, a strict accuracy of around 5-6%. The logs showed the exact values lie between 0.051 and 0.064.

An interesting result is shown in figure 3.32: in this configuration, the accuracy on non-zero samples is higher than its complement. This can be explained by the following:

First, the model already has low accuracy, and the difference is only by a few percents.

Second, while the **global** accuracy of a model that makes uniformly distributed random guess is class priors independent, the accuracy for a given class will change. The Paul Otlet data has a much higher percentage of zeroes than the Parnas one (due to how we gathered the data and how it has been pre-processed)[30]. Thus, because the guess are independents and we only consider the samples that are actually labelled as 0 :

$$P(\text{correct on zeroes}) = P(say0|0) = \frac{P(0 \cap say0)}{P(0)} = P(say0)$$

which *should* indicate the class accuracy would be the same as the global one. However, the training and testing set have far different priors, to be exact we have the following result:

$$P(\text{correct on zeroes}) = P(say0\text{Parnas}) = \frac{\#0 \text{ in Parnas}}{\#\text{samples in Parnas}} < \frac{\#0 \text{ in Otlet}}{\#\text{samples in Otlet}}$$

[29] One might argue that if a class has not been seen, this sum does not equal to 1. However, our first equation would remain unchanged because, in absence of zero-shot learning, a classifier can never predict an unseen label. Thus $P(sayC\_unseen) = 0$ and our first equation is still correct.

[30] As a remainder, 15.58% for the Parnas room while 60% for the Paul Otlet! This is a major reason why we preferred to consider the Parnas data set for much of our experiments.

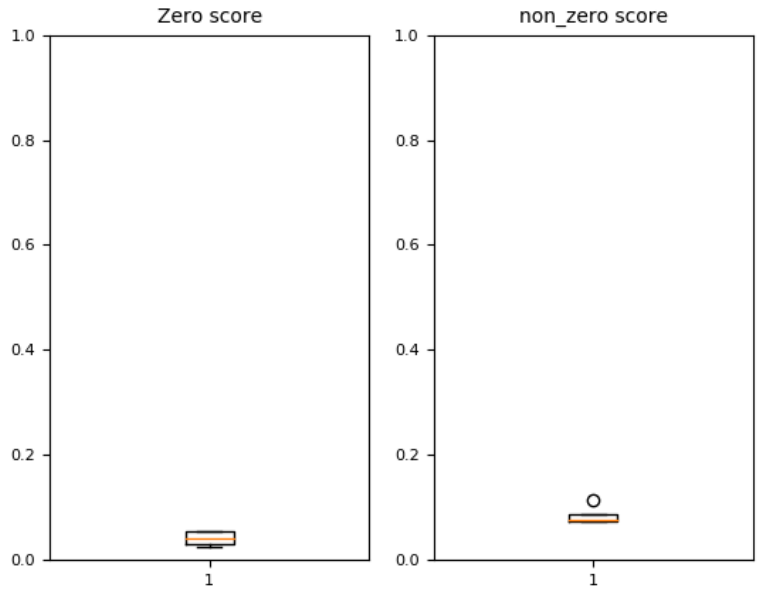


Figure 3.32: Accuracy of the Voting model when trained on Parnas room and tested on Otlet on zeroes vs non-zeroes samples with the strict accuracy metric.

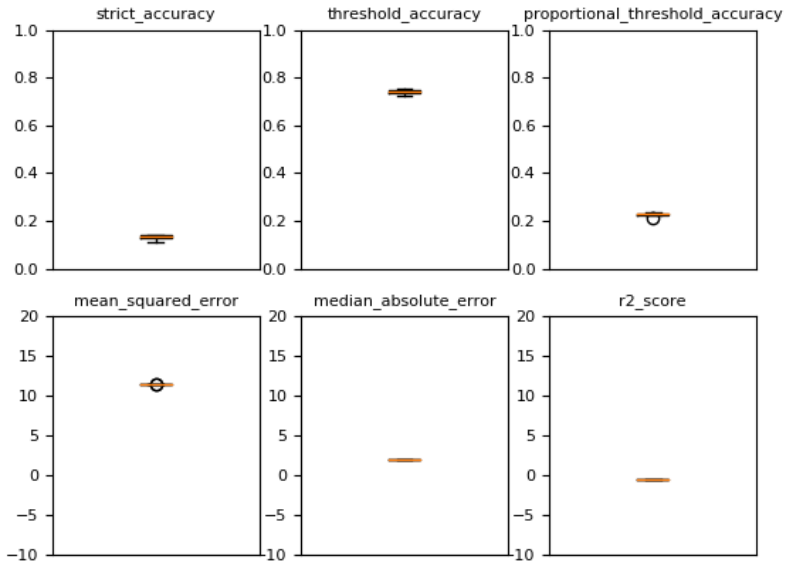


Figure 3.33: Accuracy of the genetic model when trained on Otlet room and tested on Parnas. The r2 score is slightly negative

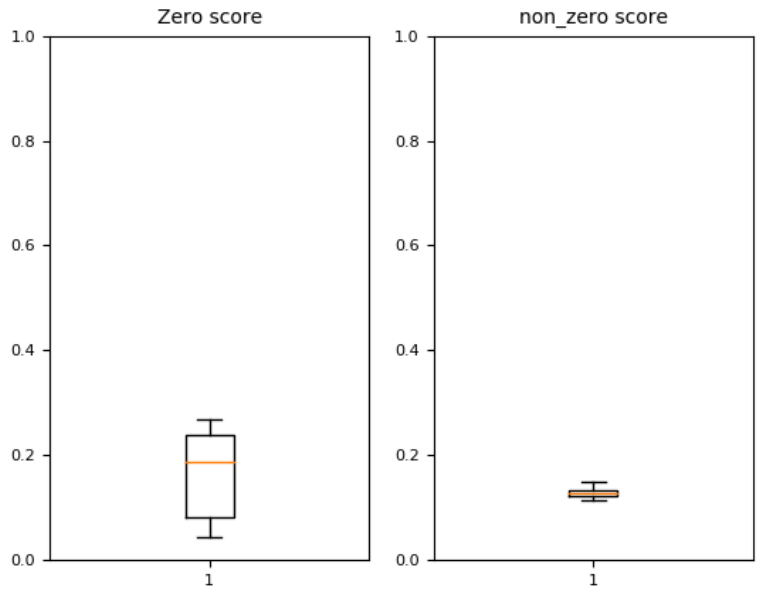


Figure 3.34: Accuracy of the genetic model when trained on Otlet room and tested on Parnas on zeroes vs non-zeroes samples with the strict accuracy metric.

We see in figures 3.33 and 3.34 that the global results are similar while inverting the training and testing set. A notable result though is the variance in the zero score. But this is certainly due to the large number of zero samples inside the Otlet dataset that implies much flexibility in the trained model.

3.2.5 Both rooms accuracy

Next, we will try to apply the model to both datasets as training and testing set, using the decomposition and cross-validation we used earlier in listing 1.

We see in figures 3.35, 3.36 and 3.37 that even while combining data from different rooms, the algorithm has sufficient knowledge in order to make pretty good predictions. However, this dataset is mainly composed of data from the Parnas room. For the sake of completeness, we computed the same figures (see figures 3.38 3.39 and 3.40) with balanced dataset: that is, we select randomly X samples in the Parnas data where X is the number of samples of the room Paul Otlet (because the latter is smaller).

Here we see a slight, but barely noticeable decrease in accuracy (around 5%). We noticed in some experiments that the accuracy is generally a bit lower in the Otlet dataset (when trained and tested on this set only) in general. First, because it is smaller and secondly, because it is more biased towards the 0 class, thus it makes it difficult to apprehend for some of the techniques we used in the combination. We thus feel that this decrease is normal and follows our expectations. For the rest of our work, we will continue to use both entire datasets without balancing.

Next, we will check again the importance of features for this Voting algorithm and evaluate its accuracy while removing the noisy or unnecessary features.

The figure 3.41 shows the feature importance for the Voting algorithm. It reveals that *temperature*, *humidity*, *light* and *CO2* are useful, while *motion* and *CO2_derived* are (very) slightly degrading performance. This was predictable because we use a combination of techniques that had similar features marked as important. Figure 3.42 shows the accuracy of the model

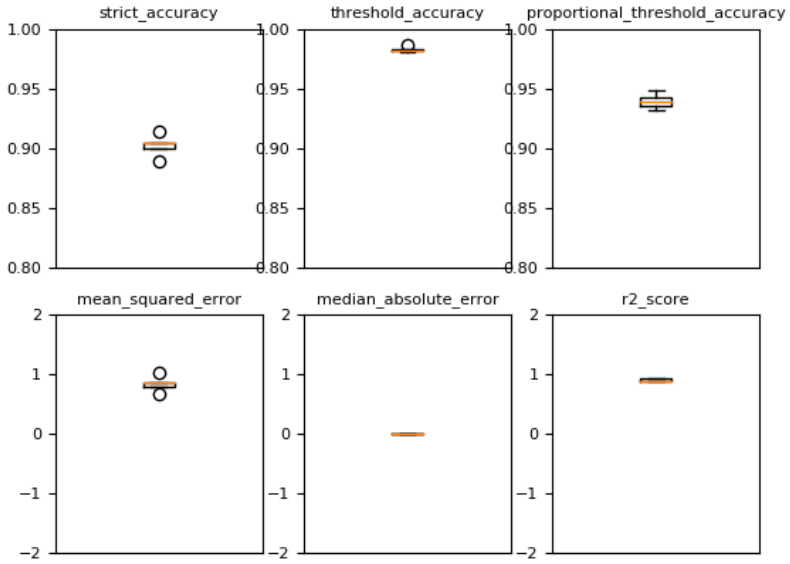


Figure 3.35: Accuracy of the genetic model trained and tested on data from both Parnas and Paul Otlet rooms.

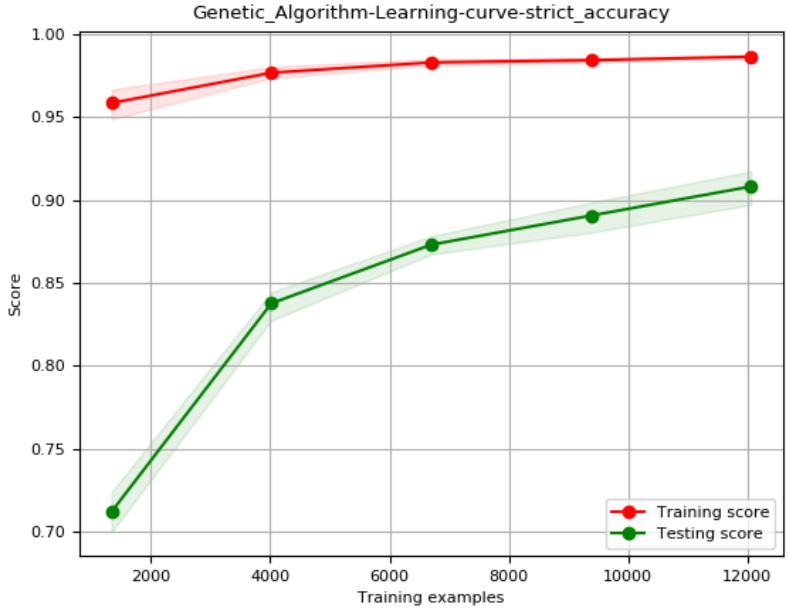


Figure 3.36: Learning curve of the genetic model trained and tested on data from both Parnas and Paul Otlet rooms with the strict accuracy metric.

while having removed *CO2_smoothed*, *CO2_derived* and *motion* features. We decided to also delete the *CO2_smoothed* since it is highly correlated to the *CO2*. The effectiveness of the model seems to be the same for every of the 6 metrics. The differences are only at the order of the percent: before feature removal, accuracy was between 89-91% and after it lies in 87-90% . We also note that the variance of the model slightly increase compared to figure 3.35.

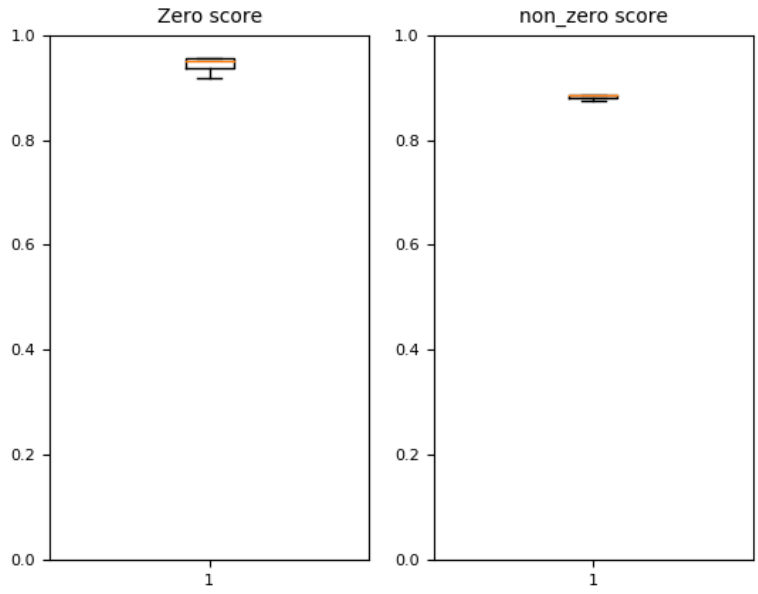


Figure 3.37: Zero vs non-zero scores of the genetic model trained and tested on data from both Parnas and Paul Otlet rooms with the strict accuracy metric.

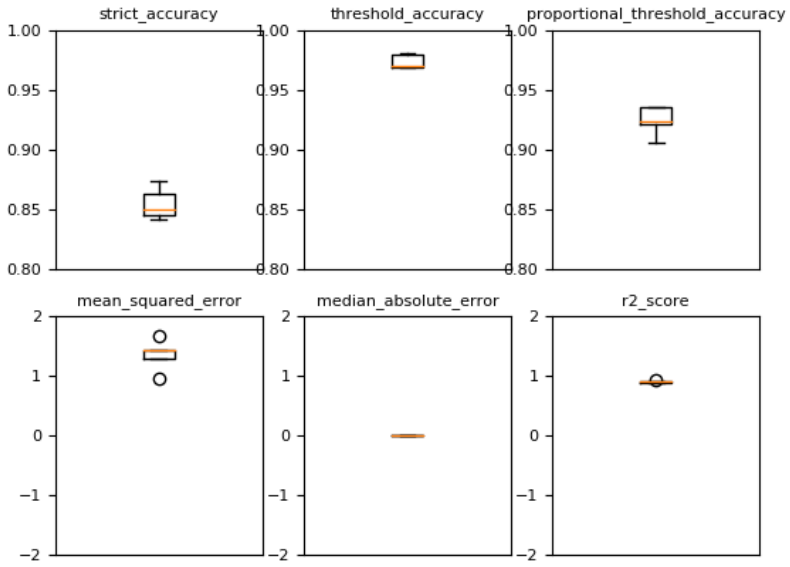


Figure 3.38: Accuracy of the genetic model trained and tested on data from both Parnas and Paul Otlet rooms with balanced data set.

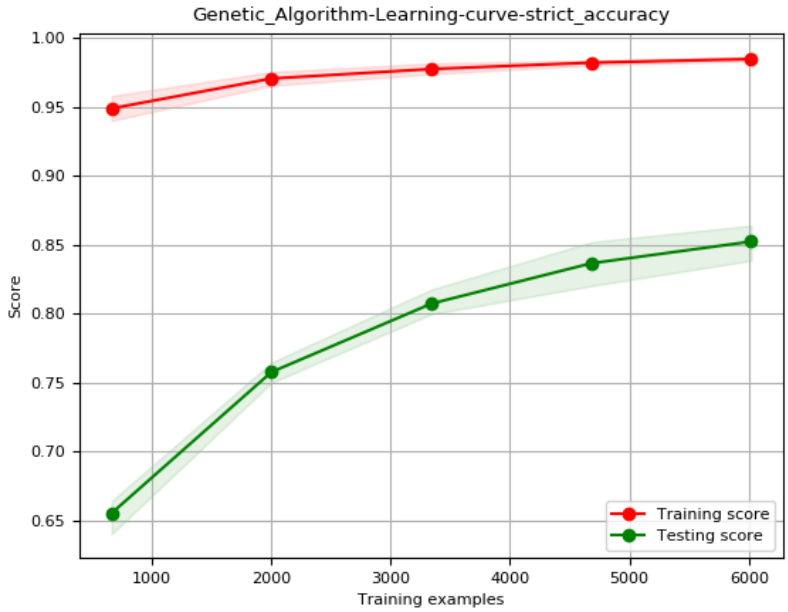


Figure 3.39: Learning curve of the genetic model trained and tested on data from both Parnas and Paul Otlet rooms with balanced data set with the strict accuracy metric.

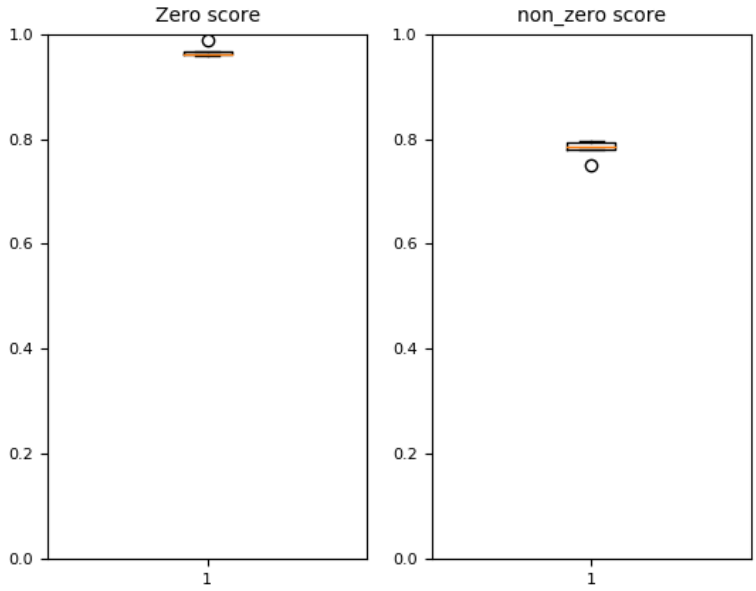


Figure 3.40: Zero vs non-zero scores of the genetic model trained and tested on data from both Parnas and Paul Otlet rooms with balanced dataset with the strict accuracy metric.

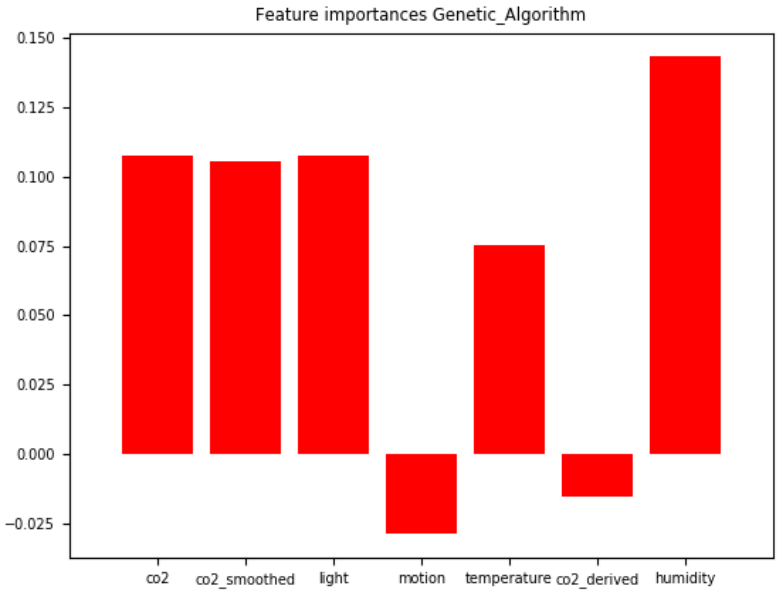


Figure 3.41: Feature importance of the Voting model trained and tested on data from both Parnas and Paul Otlet rooms with strict accuracy metric.

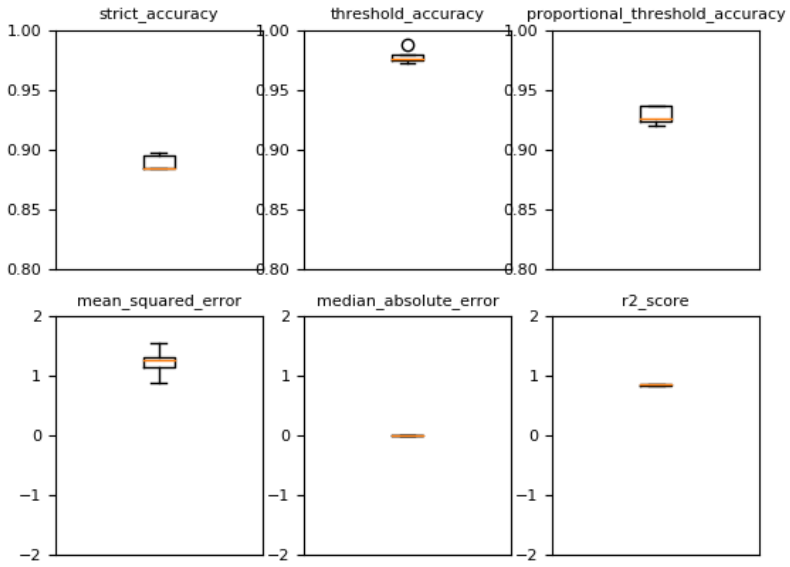


Figure 3.42: Scoring metrics of the Voting model trained and tested on data from both Parnas and Paul Otlet rooms with features CO2_smoothed CO2_derived and motion removed

3.2.6 General conclusion on the accuracy

We will now summarize what we discovered throughout this section and briefly report the observations we made :

- Some techniques performs badly overall and do not improve with the number of samples
- Most of the techniques have an average strict accuracy
- Some techniques have a high to very high strict accuracy on their own (e.g. Random Forest)
- Some classifiers outperform their regressor counterpart, most yield similar results
- Most of the good models select the same features as important
- The Otlet dataset is more biased towards the 0 class and thus is harder to predict
- Models are good at making predictions for the same room but are close to random for the other one
- However, the prediction is still good when combining the data of different rooms
- Grouping regressors by applying the mean of the values decreases performance
- Grouping models can reduce variance and may increase accuracy. Although it was barely noticeable in our case

The end results of our work are thus in figure 3.35. As a remainder, a `Voting model` composed of:

- `RandomForestClassifier`: 'max_depth': 40, 'min_samples_split': 10, 'n_estimators': 200.
- `RandomForestRegressor`: 'max_depth': 30, 'min_samples_split': 10, 'n_estimators': 200.
- `MLPClassifier`: 'hidden_layer_sizes': (50, 50).
- `MLPRegressor`: 'hidden_layer_sizes': (50, 50).
- `KNeighborsRegressor`: 'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'distance'.
- `KNeighborsClassifier`: 'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'distance'.

That takes samples from two different rooms can predict the exact number of people with more than 87% accuracy (90% on average) and with 93% if it is allowed a threshold of 0.35 times the number of people present (e.g. if there are 7 people in the room, we allow a maximum error of 2 persons).

This concludes our analysis about the performance of machine learning technique to predict the number of people present in a room knowing the *co2, co2_smoothed, co2_derived, humidity, light, temperature* and *motion*.

Chapter 4

Reproducibility and extensions

4.1 Reproducibility

Throughout this work, a particular care was given to the reproducibility of the different steps we followed. The data sets and source code are available to who would like to deepen the analysis. Especially, the machine learning models, parameters and seeds for random operations are given. The most difficult part being the data gathering, it is possible that the information measured is inaccurate and even erroneous. However, no data set is perfect and machine learning even require the data to contain some noise.

The source-code for the machine learning part along with the dataset is available at

`https://bitbucket.org/masterthesisiot/masterthesisiot/`

4.2 Possible Extensions and Improvements

Before concluding this report, we will present some further improvements that can be done in order to extract more knowledge and enable more uses of the techniques showed.

Better grid search for the camera We designed a grid-search in order to tune the parameters of the camera counting the people entering and leaving the Otlet room. However, we only based the accuracy in the grid search on the number of frames with the correct person count using small videos. This technique is really prone to overfitting and choosing another scoring method might improve the final result. One can also avoid this search thanks to solid knowledge in image processing and motion detection.

Find other automatic ways to count people We have explored only a few methods to count the people in a room to generate the training set for the supervised learning. One might explore ideas that were presented in this report or new ones. Later in the production, we discovered that some IoT companies used WiFi and Bluetooth device detection to estimate the number of people in large events. However, this solution appeared to be "overkill" for small indoor rooms (the hardware being much more expensive first). If the technique is not much intrusive or costly, it might even replace the need of CO2 sensors and learning since it is the knowledge targeted for possible real life useful applications.

Extend parameter tuning for some ML techniques Some machine learning techniques have not been tested at all, while some others were not extensively analyzed. Some of them because of the lack of knowledge about the parameters or their intrinsic functioning. While it is prohibitive to test every existing algorithm, it might be interesting to try the `Gaussian Processes` for instance.

Extract other features from available data The original dataset has been extended with the *CO2 smoothed* and *CO2 derived* features. Although we discovered that the *CO2 derived* was not useful for the final model, one can extract other features, combination of feature that could increase the accuracy. There are many ways to compute the *CO2 derived*, by changing the formula, the duration on which it is applied, build it on *CO2* instead of *CO2 smoothed*,...

Change to person count timeout from 1 hour to a different value Like many other values used throughout this paper, the validity period for the (manual) person count value was set to 1 hour. But the dataset would have been much bigger and maybe harder to predict than the current one. Tuning those parameters can improve the value of the gathered data and make use of many more measurements.

Distinguish two types of non-zero errors We analyzed the zero and non-zero scores of the best models chosen. The number of 0 samples (real value) that were correctly classified vs the non-0 samples accurately predicted. A nice idea would be to also distinguish the type of errors within the non-zero examples. Whether the mistake is : the estimator concludes there are people in the room but not the correct number, or the estimator predict there is nobody while there are some persons. This could give an indication of the accuracy of the model to predict the emptiness (or not) of a room. Which is still a nice information that could enable numerous applications.

Use custom kernel for some techniques As we detailed in the 3.1.4 section, *sci-kit* allows the users to define custom kernels for the different techniques that support this parameter. We said that kernels are to be handled with caution (especially their parameters) to avoid overfitting and unstable behavior. Nonetheless, one can build up a kernel made of already existing one by combining them (summing, multiplying,...). We actually explored this possibility while considering `Gaussian Processes` but still ended up with memory errors.

Soft voting For the `Voting classifier` from *sci-kit* and the one we defined, only the hard voting has been considered. It might however become useful for some cases to give weights to the different algorithms in order to improve accuracy and reduce variance. Our first guess is that giving more weight to the `Random Forest Classifier` should improve the final model performance.

Sampling from a sufficiently large number of rooms There are many ways to possibly improve the work that has been presented. But there is one result that could trigger much interest in many fields : we observed that our model do not generalize from one room to another but remember those are very different in volume, windows, lighting, average occupation, air conditioning,... Still, we observed that when trained on both rooms, the algorithm do not loose much of its accuracy (we believe the slight drop with both entire data set and the more noticeable one on size-balanced sets are due to the bias from the Otlet room measurements). Thus, it is possible that with enough measurements from a sufficiently large number of rooms (that have

some similarities of course, having a model predicting the number of people in a bureau or in an auditorium would be highly challenging if not impractical) one can build a super training set and train a super machine learning model from it that would then be able to generalize well. We acknowledge that using less overfitting machine learning techniques might also be a way to generalize but we think acquiring knowledge from rooms really different that the already seen ones is counter-intuitive. By acquiring much knowledge over various rooms, the technique could really be profitable and empower IoT infrastructure.

4.3 Related Work

4.3.1 CO2 based room occupancy detection

The first category of related work is directly connected to our final purpose : room occupancy detection. Either based on CO2 or on other means.

Chaoyang Jiang, Mustafa K. Masood, Yeng Chai Soh, and Hua Li in [4] present a room occupancy estimator. Predicting the number of people inside a room based on CO2 solely. There are using Feature Scaled Extreme Learning Machine (FS-ELM) algorithm using a pre-smoothing of the CO2 data in order to reduce variance and impact of spikes on the results. They also introduce the x-tolerance accuracy, from which came our idea of *threshold accuracy* and *proportional threshold accuracy.* However, they are testing their algorithm in an office room "*with 24 cubicles and 11 open seats*" and can achieve an "*accuracy is up to 94 percent with a tolerance of four occupants.*"

Chen Mao and Qian Huang in [1], instead of relying solely on CO2 measurements, are also measuring light intensity to predict room occupancy. They use "*a wireless CO2 sensor network is connected with HVAC systems to realize fine-grained, energy efficient smart building.*" Note that they are relying on light in front of the entrance door of a room in order to improve the accuracy of the model. This solution was applicable in their case because the room entrance was in front of a window, which was not our case. Note that their two systems are independent but complementary. On the contrary, in our work, we relied on sensors generating both data and applied machine learning techniques to the entire set.

M.S. Zuraimi, A. Pantazaras, K.A. Chaturvedi, J.J. Yang, K.W. Tham and S.E. Lee in [25] "*found that that the dynamic physical models and Support Vector Machines (SVM) and Artificial Neural Networks (ANN) models utilizing a combination of average and first order differential CO2 concentrations performed the best in terms of predicting occupancy counts with the ANN and SVM models showing higher predictive performance*" The idea to add the average and first differential was somehow applied in our work, but it showed no particular advantage to us however. Furthermore, even if the `MLP` revealed to be useful in our case, the `SVM` appeared slow to train and yielded a run-of-the-mill strict accuracy. Their "*Model average accuracies across all tolerance was between 70 and 76% demonstrating good performance for a large number of occupants*" A limitation of our work is the rooms we analyzed, in which the maximum number of occupant was 14 in our data set.

Andrzej Szczurek, Monika Maciejewska and Tomasz Pietrucha in [26] are combining CO2 concentration,temperature and relative humidity in order to predict room occupancy. "*Particularly useful is the combination of temperature and humidity sensors. The analysis revealed that the choice of appropriate classifier is very important. Nonparametric, nonlinear, minimum distance classifier (k-NN) was very effective, while parametric classifier, utilizing linear discriminant functions (LDA) was unsuccessful. The presented method may be also used for determining the duration of occupancy periods*". We are using the same set of data source (*temperature*, *CO2*

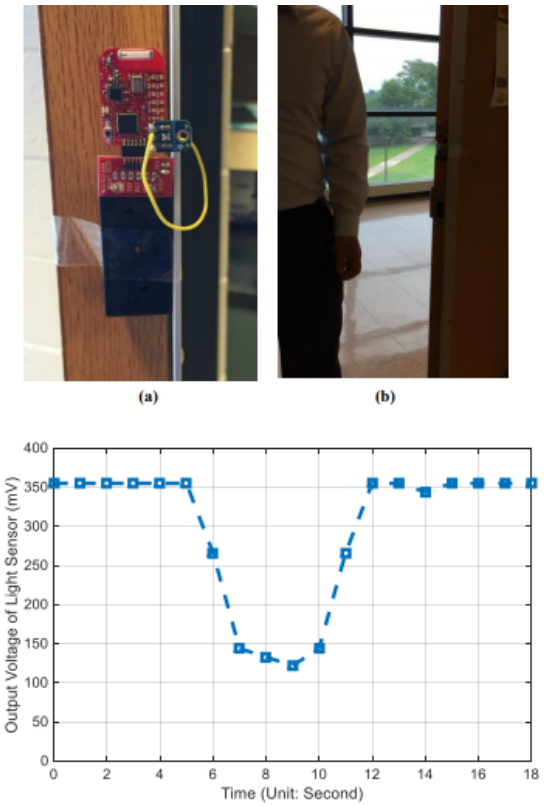


Figure 4.1: (a) Experimental setup of a light sensor on the door frame (b) A person walking and passing the door frame (c) Measurement response of a light sensor response to a passing event. Image taken from [1]

and relative *humidity*) but with the addition of extracted feature, light and motion (the latter revealed to be useless to our model). Retrospectively, the model choice they made comfort us with our final algorithm, using `KNeighborRegressor` and `KNeighborClassifier` along with other techniques.

Finally, H*wataik Han, Kyung-Jin Jang, Changho Han, and Junyong Lee* analyzed the occupancy estimation based on CO2 using Dynamic Neural Networks. More precisely, a TDNN (time-delayed neural network) by varying the number of tapped delay lines and the number of neurons. Note that "*Networks were trained using the first-day data and results were obtained for the rest of the days*" meaning the training set was quite small compared to the testing one. They are also introducing a time shift between the sample and the actual prediction : the predicted number of people in a room is based on samples made between 5 and 15 minutes earlier. We did not get far in using neural networks in this work. We relied solely on `Multi Layer Perceptron` which has a fixed, user-determined structure. Because some techniques showed good accuracy, and because we have little experience with neural networks, we did not explore that possibility.

4.3.2 Image processing and motion detection

A second category was more of help for the camera solution used for the person counting part. It mainly involves image processing techniques and motion detection.

P. KaewTraKulPong and R. Bowden in [27] analyze the background subtraction technique in motion detection that can also differentiate shadows from moving objects using a computational colour space. Their model also improves the speed of learning and accuracy compared to previous (although successful) work. This paper was referenced in the motion detection tutorial we

followed to become familiar with *picamera* and *OpenCV*. The paper is the paveway for the image processing techniques we used to implement the person counting system using the Raspberry pi with a camera.

Zoran Zivkovic in [28] explored an Improved Adaptive Gaussian Mixture Model for Background Subtraction. This paper explains the use of Gaussian mixture applied to background subtraction. It details equations and advanced techniques for the motion detection task we used with the Raspberry pi camera.

One year later, *Zoran Zivkovic* and Ferdinand van der Heijden in [29] researched an "*Efficient adaptive density estimation per image pixel*" for the background subtraction task. They give the recursive equations that are used in order to build a background subtraction model.

Andrew B. Godbehere, Akihiro Matsukawa and Ken Goldberg in [30] analyzed in detail a Visual Tracking of Human Visitors under Variable-Lighting Conditions. This idea was used in this work in order to adapt our background subtraction to different times of the day. We implemented a (basic) concept of computing the background using the means over the 100 (or more) previous images. We denote the authors are also using the *OpenCV* library.

Badhan Hemangi and K. Nikhita in [3] are detailling the setup of a "*people counting system using Raspberry pi with opencv*". This article was the closest to our task. Our program using a Raspberry pi camera is mostly based on it. They used it "*at the entrance of a building so that the total number of visitors can be recorded*". However, we think the particular location of the rooms we considered are tough compared to the examples we have seen in several posts and articles. The Parnas room is impractical because its entrance is located in a corridor, thus it requires to distinguish people passing-by vs actually entring/leaving the room. While the Otlet room has a dark entrance and an insufficient lighting. If one was to adjust the camera setting, then the sunlight coming from the room (the camera being outside it) would blind the camera and make detection impossible from the brightness shift (or it would need to be dynamically adjusted).

Conclusion

This work aims at predicting the number of people inside a room knowing *co2, humidity, light, temperature, motion* and having extracted the *co2_smoothed, co2_derived* features. The training data has been gathered with IoT sensors relying on the LoRaWAN network for TTN. The number of people for this data set has been (semi) manually retrieved although an attempt to collect it automatically with a camera mounted on a Raspberry pi, using image processing for motion detection, has been performed. The set has then been pre processed in order to retain only potentially useful and remove untrusted information. Some features were extracted and then many machine learning techniques, with the help of the *sci-kit* python library, have been applied to the datasets. Some of them showed pretty good results and they were combined into a `Voting algorithm` `Random Forest`, the `KNeighbors` and the `MLP`, one from *sci-kit* using only the classifiers and one made within this project using the classifiers and the regressors.

The final model can predict the exact number of people with a 85% accuracy. The model is not able to predict the *person count* from an unseen room where its performance is similar to a random guess. However, while combining the data sets from different places the method still performs well and keeps its accuracy. One should just remind that the Otlet dataset is biased, in particular, towards the empty room case. While there are many ways to possibly improve the presented technique, we think we followed reasonable guesses and assumptions throughout this task, based on our experience and knowledge. We also believe that with a more dense data set from many different (but similar) rooms, a machine learning model can generalize to unseen, similar, rooms and become handy for various purposes.

Abbreviations

Here we will recap the different abbreviations we have seen throughout this work. Some of them were used inside the provided program.

- **API** Application programming interface
- **ARD** Automatic Relevance Determination Regressor
- **CART** Classification and Regression Tree
- **clf** Classifier
- **GP** Gaussian Processes
- **GUI** Graphical User Interface
- **KNN** K-Nearest-Neighbor
- **KRR** Kernel Ridge Regression
- **LDA** Linear Discriminant Analysis
- **LC** Learning Curve
- **max iter** maximum number of iterations
- **MLP** Multi-layer Perceptron
- **n estimators** number of estimators (bagging procedures)
- **n iter** number of iterations
- **n neighbors** number of neighbors
- **NB** Naive Bayes
- **NN** Nearest Neighbors
- **OMP** Orthogonal Matching Pursuit
- **QDA** Quadratic Discriminant Analysis
- **RBF** kernel Radial Basis Function kernel
- **SGDRegressor** Stochastic Gradient Descent Regressor
- **SVM** Support Vector Machine
- **tol** tolerance
- **TTN** The Things Network

Bibliography

[1] C. Mao and Q. Huang, "Occupancy estimation in smart building using hybrid co2/light wireless sensor network," *Journal of Applied Sciences and Arts*, vol. 1 : Iss. 2 , Article 5, 2016.

[2] H. Han, K.-J. Jang, C. Han, and J. Lee, "Occupancy estimation based on co2 concentration using dynamic neural network model," *34th AIVC - 3rd TightVent - 2nd Cool Roofs' - 1st venticool Conference*, September 2013.

[3] B. Hemangi and K. Nikhita, "People counting system using raspberry pi with opencv," *International Journal for Research in Engineering Application & Management*, vol. 2, April 2016.

[4] C. Jiang, M. K. Masood, Y. C. Soh, and H. Li, "Indoor occupancy estimation from carbon dioxide concentration," *Energy and Buildings*, vol. 131, pp. 132–141, July 2016.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[7] P. Dupont, "Lecture notes of lingi2262 machine learning : classification and evaluation," June 2017.

[8] R. Duda, P. Hart, and D. Stork, "Pattern classification (decision trees)," *Wiley-Interscience, 2nd edition*, 2001.

[9] T. Mitchell, "Machine learning (linear discriminant analysis)," *McGraw Hill*, 1997.

[10] B. C.M., "Pattern recognition and machine learning," *Springer*, 2006.

[11] B. Boser, I. Guyon, and V. . Vapnik, "A training algorithm for optimal margin classifiers.," *In Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, 1992.

[12] Y.-C. Ho and R. Kashyap, "An algorithm for linear inequalities and its applications," *IEEE Transactions on Electronic Computers*, vol. 14, pp. 683–688, 1965.

[13] N. Cristianini and J. . Shawe-Taylor, "Support vector machines and other kernel-based learning methods.," *Cambridge University Press*, 2000.

[14] B. Scholkopf and A. Smola, "Learning with kernels: Support vector machines, regularization, optimization and beyond.," *MIT Press, Cambridge*, 2002.

[15] J. Shawe-Taylor and N. Cristianini, "Kernel methods for pattern analysis.," *Cambridge University Press.*, 2004.

[16] V. Vapnik, "The nature of statistical learning theory.," *Springer, 2nd*, 2000.

[17] R. T. e. T. H. Jerome H. Friedman, "The elements of statistical learning," *Springer, 2nd*, 2009.

[18] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1 n°1, pp. 81–106, 1986.

[19] L. Breiman, J. Friedman, R. Olshen, and P. Stone, "Classification and regression trees," *Wadsworth International Group*, 1984.

[20] R. Duda, P. Hart, and D. Stork, "Pattern classification (linear discriminant analysis)," *Wiley-Interscience, 2nd edition*, 2001.

[21] T. Mitchell, "Machine learning (decision trees)," *McGraw Hill*, 1997.

[22] J. Quinlan, "C4.5: Programs for machine learning.," *Morgan Kaufmann*, 1993.

[23] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24 n°2, pp. 123–140, 1996.

[24] L. Breiman, "Random forests," *Machine Learning*, vol. 45 n°1, pp. 4–32, 2001.

[25] M. Zuraimi, A. Pantazaras, K. Chaturvedi, J. Yang, K. Tham, and S. Lee, "Predicting occupancy counts using physical and statistical co2-based modeling methodologies," *Building and Environment*, vol. 123, pp. 517–528, October 2017.

[26] A. Szczurek, M. Maciejewska, and T. Pietrucha, "Occupancy determination based on time series of co2 concentration, temperature and relative humidity," *Energy and Buildings*, vol. 147, pp. 142–154, July 2017.

[27] P. KaewTraKulPong and R. Bowden, "An improved adaptive background mixture model for realtime tracking with shadow detection," *In Proc.2nd European Workshop on Advanced Video Based Surveillance Systems*, September 2001.

[28] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," *In Proc. ICPR*, 2004.

[29] Z. Zivkovic and F. van der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern Recognition Letters*, vol. 27, pp. 773–780, August 2005.

[30] A. B. Godbehere, A. Matsukawa, and K. Goldberg, "Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation," *n: American Control Conference (ACC)*, June 2012.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**