UMinho | 2020

David Daniel Pinto Coelho Kramer

Crowd Sensing and Forecasting for Smart Cities

Universidade do Minho
Escola de Engenharia
Departamento de Informática

David Daniel Pinto Coelho Kramer

Crowd Sensing and Forecasting for Smart Cities

Dezembro de 2020

Universidade do Minho
Escola de Engenharia
Departamento de Informática

David Daniel Pinto Coelho Kramer

Crowd Sensing and Forecasting for Smart Cities

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação de
Professor Cesar Analide

dezembro de 2020

COPYRIGHT NOTICE

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisors Professor Cesar Analide and Professor Bruno Fernandes for their expertise, availability and dedication, which were key throughout the development of this dissertation.

I would also like to thank my everlasting friends João, Lages, Luís and Rui for being in my life for such a long time and yet still being the company that I always appreciate and need.

During this five year journey, I had the opportunity to meet a lot of great people. I am thankful to all of you. Also, a special thanks to Pedro, who was my partner in this journey, who grow up with me, who was a friend and with whom I have a lot of memories I deeply appreciate.

Last, but not least, I am forever grateful to my family, specially to my mother Maria. Thank you for your support, your core values and for being a role model in all aspects of life.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of Universidade do Minho.

Braga, dayth Month 2020

[illegible]

RESUMO

A utilização de inteligência sob forma de tecnologia no nosso dia-a-dia é uma realidade em crescimento e, portanto, devemos fazer uso da tecnologia disponível para melhorar várias áreas do nosso quotidiano. Por exemplo, a tecnologia atual permite a conceção de sensores inteligentes, mais especificamente sensores de multidão, para detetar passivamente dispositivos como *smartphones* ou *smartwatches* através de *probe requests* emitidos por estes dispositivos que, por sua vez, fazem parte de um processo de comunicação que ocorre sempre que o Wi-Fi dos dispositivos está ativado. Adicionalmente, *crowd sensing* - uma solução de *Ambient Intelligence (AmI)* - é estudada hoje em dia em várias áreas com bons resultados. Portanto, esta dissertação visa investigar e utilizar sensores de multidão para capturar passivamente dados acerca da densidade de multidões, explorar as capacidades do sensor escolhido, analisar e processar os dados para obter melhores estimativas, e conceber e desenvolver modelos de *Machine Learning (ML)* para prever a densidade nas áreas sensorizadas. Áreas nas quais o sensor de multidão está inserido - AmI, *Smart Cities*, Wi-Fi *Probing* - são estudadas, juntamente com a análise de diferentes abordagens ao *crowd sensing*, assim como paradigmas e algoritmos de ML. Em seguida, é explicado como os dados foram capturados e analisados, seguido por uma experiência feita às capacidades do sensor. Além disso, é apresentado como os modelos de ML foram concebidos e otimizados. Finalmente, os resultados dos vários testes de ML são discutidos e o modelo com melhor desempenho é apresentado. A investigação e os resultados práticos abrem perspetivas importantes para a implementação deste tipo de soluções na nossa vida diária.

Palavras-Chave: *Ambient Intelligence*, *Crowd Sensing*, *Machine Learning*, *Smart Cities*, Séries Temporais

ABSTRACT

Bringing intelligence to our everyday environments is a growing reality and therefore we should take advantage of the technology available to improve several areas of our daily life. For example, current technology allows the conception of smart scanners, more specifically crowd sensors, to passively detect devices such as smartphones or smartwatches through probe requests emitted by such devices, that, in turn, are part of a communication process that happens every time the devices' Wi-Fi is enabled. Additionally, crowd sensing - an Ambient Intelligence (AmI) solution - is being studied nowadays in several areas with good results. Therefore, this dissertation aims to research and use crowd sensors to passively collect crowd density data, explore the capabilities of the chosen sensor, analyse and process the data to get better estimations and conceive and develop Machine Learning (ML) models to forecast the density of the sensed areas. Areas in which crowd sensing is inserted - AmI, Smart Cities, Wi-Fi probing - are studied, along with the analysis of different crowd sensing approaches and ML paradigms and algorithms. Then, it's explained how the data was collected and analysed together with the insights obtained from it, followed by an experiment done on the crowd sensor capabilities. Moreover, it's presented how the ML models were conceived and tuned. Finally, the results from the ML several tests are discussed and the best performing model is found. The investigation, together with practical results, opens important perspectives for the implementation of these kinds of solutions in our daily lives.

Keywords: Ambient Intelligence, Crowd Sensing, Machine Learning, Smart Cities, Time Series Problems

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ACRONYMS

AI Artificial Intelligence.

AmI Ambient Intelligence.

ANN Artificial Neural Network.

AP Access Point.

API Application Programming Interface.

ARIMA Autoregressive Integrated Moving Average.

BPTT Backpropagation Through Time.

CNN Convolutional Neural Network.

CRISP-DM CrossIndustry Standard Process for Data Mining.

CV Cross-Validation.

DL Deep Learning.

ETS Exponential Smoothing State Space Model.

GPS Global Positioning System.

HD High Definition.

IDE Integrated Development Environment.

IoT Internet of Things.

IRM Infinite Relational Model.

ITU International Telecommunications Union.

LSTM Long Short-Term Memory.

MAE Mean Average Error.

MCS Mobile Crowdsensing.

ML Machine Learning.

MSE Mean Squared Error.

OS Operative System.

OUI Organizationally Unique Identifier.

PCA Principal Component Analysis.

RFID Radio-frequency identification.

RMSE Root Mean Squared Error.

RNN Recurrent Neural Network.

RSSI Received Signal Strength Indicator.

SOM Self Organizing Map.

U-matrix Unified Distance Matrix.

UN United Nations.

VRUs Vulnerable Road Users.

WPAN Wireless Personal Area Networks.

1 INTRODUCTION

In this section, the context and motivation of the problem will be discussed, along with the definition of the project's main objectives and the approach planned to achieve them. Furthermore, the research hypothesis together with the methodology used will be described, as well as the discussion of the problem and its challenges. Finally, it will give an overview of this dissertation's structure.

1.1 CONTEXT & MOTIVATION

Bringing intelligence to our everyday environments is a growing reality and therefore we should take advantage of the available technology to improve several areas of our daily life. The growing reality of technology, namely AmI and Smart Cities, respectively defined as *"a digital environment that proactively, but sensibly, supports people in their daily lives"* (Cook et al., 2009) and as a city that has the *"ability to reason upon the knowledge acquired through data gathered by sensorization, with focus on improving the quality of life at urban centres, considering sustainability and safety principles"* (Fernandes et al., 2018), enables the development of different solutions to improve areas of our lives, namely urban security, marketing or energy management. In fact, current technology allows the conception of smart scanners to passively detect devices such as smartphones or smartwatches through the sense of Wi-Fi probe requests and Bluetooth signals. The development of crowd control algorithms and the use of these crowd sensors will allow the gathering of important data, such as crowd density data, which can be used in distinct fields to positively affect our day-to-day. Moreover, ML, a subset of Artificial Intelligence (AI) that is a key part of AmI and Smart Cities, can be used to learn with these data and take an extra step in establishing the importance of collecting crowd density data (Fernandes et al., 2018; Cook et al., 2009; Jung and Muñoz, 2018).

In terms of motivation, the following vision by Weiser (1991) represents the author's motivation for this project: *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it"*. The main goal and motivation of this project is to prove that it is possible to develop a system capable of enhancing different areas of our lives, while not requiring an active user's role in the gathering of data, enabling the user to live his daily life in a normal way, that is, without making any kind of extra effort and still having its life improved significantly. The fact that this project also involves the use and theoretical knowledge of emerging technologies, like ML, and a possible contribution for the affirmation of Smart Cities, also plays a big role in the development of this dissertation.

1.2 MAIN OBJECTIVES

1.2.1 *Objectives*

The goal is to non-invasively sense the density of people through a crowd sensor located at a point of interest, where there are relevant variations in crowd density, to prove that these sensors can collect data relevant enough to bring positive changes in different areas in which these can be used. More specifically, this dissertation aims to firstly choose an adequate point of interest in which smart scanners will be used to passively detect smartphones and smartwatches through probe requests emitted by such devices, estimating the crowd density of that area. These estimations will then be made more valuable and accurate through the use of crowd control algorithms and the investigation on the use of MAC randomization. Then, besides the use of crowd detection and control algorithms, the objective is to study the capabilities of the chosen smart scanner by performing a Received Signal Strength Indicator (RSSI) experiment, where the sensing device's maximum range will be explored in an indoor environment. Lastly, the conception and development of various ML models will be done to forecast the density of the sensed areas.

1.2.2 *Proposed Approach*

This project can be divided in two main phases. The first is data acquisition. This sensing will be done with the use of a Crowd Sensing Smart Scanner, more specifically a second generation ESP8266 ESP-12E NodeMCU Amica board. To program this smart scanner, the Arduino Integrated Development Environment (IDE) will be used to develop software that allows the passive detection of the Wi-Fi signals from people's smart devices. In terms of data storage, the sensor will be storing data in the Firebase Realtime Database, a cloud-hosted NoSQL database that allows the storage and synchronization of the crowd sensor data in real time. To study the maximum RSSI values and the relation with how far a device is, a separate experiment will be conducted.

The main objective of the second phase is to know the density of people in a certain area as well as the use of algorithms for crowd detection and control. Having the data gathered, a complete data analysis will be conducted in order to clean and manipulate the data to make it richer and, in this way, getting precise and useful insights on the crowd density. Then, following the gathering of valuable information on the density of people provided by the collected data, different types of ML models and configurations will be tested, to find out which combination of these performs better at forecasting the density of the sensed areas.

1.3 RESEARCH HYPOTHESIS AND METHODOLOGY

In this dissertation, there are different hypothesis that will be addressed. On one hand, it will be addressed whether crowd sensing can be done through the non-invasive capturing of Wi-Fi or Bluetooth probe requests emitted by peoples' devices. Moreover, it will test if the data collected by this method offers relevant insights on crowd density and if it's valuable enough to produce good ML models in order to forecast crowd densities. Last, but not least, it is expected to prove that the used methods can be applied to real-life use cases (traffic forecasting and better crowd safety measures, for instance), and improving the quality of life in through these.

Concerning the working methodology, this project will follow the CrossIndustry Standard Process for Data Mining (CRISP-DM) which provides an overview of the life cycle of a data mining/ML project. The phases are Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and Deployment (shown in Figure 1). More importantly, the key aspect of this life cycle is that the sequence of the phases is not rigid, so this phases are parts of an ongoing cycle of analytics activity, having to go back forth among these phases frequently (Chapman et al., 2000).

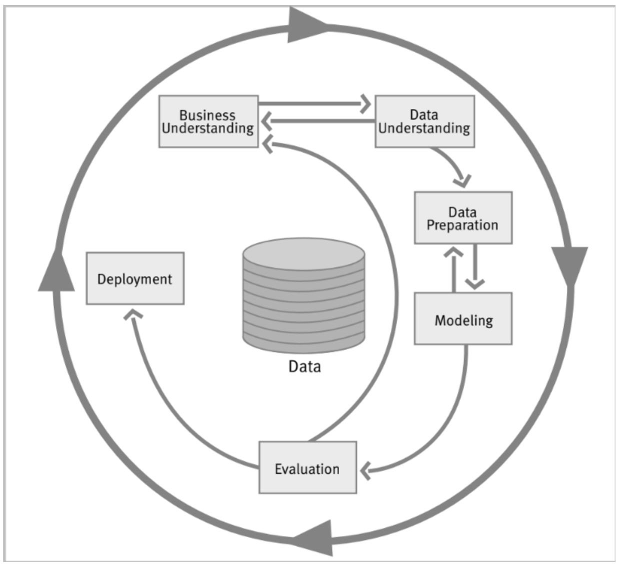


Figure 1: CRISP-DM Phases in Chapman et al. (2000).

1.4 THE PROBLEM AND ITS CHALLENGES

The purpose of this section is to explain in which areas the proposed approach can have a positive impact. Firstly, the problems that can be solved are going to be discussed. Secondly, the main expected challenges are going to be presented.

1.4.1 *Problem*

The problem can be divided in two main points. On one hand, there are areas of our life that can be improved through today's technology, namely the ones related to AmI and Smart Cities. On the other hand, the studied crowd sensing solutions have revealed some issues that could be fixed or improved.

AmI solutions have proved to be a great asset to improve several areas of our daily life. For example, in urban security, these kind of solutions can improve the safety of the Vulnerable Road Users (VRUs) (Fernandes et al., 2018). It is a known fact that the number of traffic deaths is increasing every year (World Health Organization, 2015) or that there could be more information available when wanting to plan a trip according to the traffic peaks in certain places. Another example is in the area of marketing. The decision making could be optimized if there was information about the group behavioural patterns in a certain place, making it possible to enhance each marketing decision. In either of these cases, an AmI solution, in this case crowd sensing, can be a solution to help improve these areas and several others. Ares et al. (2016) proved that crowd sensing provided insights about people's mobility in different places inside a disco, which could help enhance security, marketing or energetic efficiency issues.

Another category of the problem is related to the crowd sensing itself. The studied articles revealed that many of the approaches not always used the most efficient solution, either in terms of cost or accuracy, and also that there could be room for improvement in other aspects as well. The proposed solution will try to address these issues and develop a crowd sensing solution that takes into account factors like privacy, the type of sensing device, technologies used and the user role in the sensing phase, just to mention a few.

1.4.2 *Challenges*

The challenges can differ from AmI or Mobile and Pervasive Computing to more specific and technical one's. One major challenge is that even though a AmI system has clear benefits by continuously improving one's quality of life, the possibility of performing wrong actions demands caution when deploying such a system. Furthermore, the purpose of this dissertation is to develop a system that can bring advantages to one's life without any active participation from the user. Therefore, another challenge is being able to build technology capable of shortening the gap between humans and computing devices. Moreover, from a technical perspective, both sensing and reasoning phases have challenges that must be overcome in order to develop a viable system. Starting by the sensing phase, in the case of RSSI, a metric that can be useful for calculating how far a person is from the sensor, it might be difficult to rely solely on it because of its high sensibility to obstacles in the way of the signal. Additionally, the recently implemented MAC Address Randomization forces extra testing to conclude how much a certain MAC address is a viable identifier of a device or not. Besides that, choosing a point of interest where ground truth is available might be difficult to find, but would be helpful to prove if the system is viable or not. Also, after the sensing phase, it will be crucial to get the best out of the data available: creating new relevant features - a process called feature engineering, removing unnecessary data or choosing and testing suitable ML models are some steps to be taken.

1.5 DISSERTATION STRUCTURE

This document contains the following main sections: Introduction, State of Art, Experiments, Results and Discussion, and, finally, the Conclusion.

Firstly, the introductory chapter starts by establishing an overall view on this thesis theme context and motivation. Furthermore, the main objectives and the proposed approach to reach them are presented, along with the research hypothesis and methodology. Finally, the problem and its challenges are discussed.

In the State of Art chapter, the goal is to describe minutely the different kind of approaches on crowd sensing, specially the approaches that also involve the detection of Wi-Fi or Bluetooth probe requests and then reason upon its main advantages and disadvantages. Besides the description of the different solutions to help enhancing the numerous areas of a Smart City, this section contains information about the current state of ML, along with its different paradigms.

In the experiments chapter, a detailed overview of the development done along with the materials used in the various project phases are given. More specifically, in the data collection and data analysis sections, not only explain how these were performed, but also discuss the conclusions and insights taken from the data. Finally, through the collected data, the model conception and tuning section enunciates and explains the steps taken to achieve the best performing ML model.

Additionally, the results and discussion chapter presents the results from the various configurations tested in the previous section, along with the discussion of these.

Finally, the conclusion chapter enunciates the overall results of all the work done, whether they met or not the initially proposed goals, and, additionally, suggestions as a way to improve the current work.

2

STATE OF THE ART

In this section, it will be explained how Smart Cities and AmI are being broadly used and studied nowadays, and more specifically, how different authors approach crowd sensing and how they make the sensed data useful for different case studies. Also, an overview of ML paradigms and algorithms will be given.

2.1 SMART CITIES

In the cities' history, the importance of technology to improve quality of life has been always recognized and explored. Over the humankind's history, there are several examples of how technology can be useful for cities. Indeed, 3000 years ago Persian engineers dug a Qanat, a long tunnel connecting a well to its outflow many miles away. This ancient example of open and forward thinking, made possible the water supply for a city of one million inhabitants until very recently (Harrison et al., 2010).

In this modern times, with all the technology available we should strive for making better use of all the public resources available in order to improve numerous areas of a city and ultimately increase the quality of life of each citizen along with the reducement of operational costs of the public administrations (Zanella et al., 2014). Along side with Smart Cities stands Internet of Things (IoT) (Fernandes et al., 2018). IoT is related with technologies such as sensors, actuators, Global Positioning System (GPS) devices, and mobile devices (Xu et al., 2014). The use of these resources can improve the quality of areas such as agriculture, urban security, environmental monitoring, security surveillance, and others (Xu et al., 2014; Fernandes et al., 2018; Zanella et al., 2014). In fact, the availability of different types of data collect by IoT sensors, may be used to enhance the awareness of people about the status of their city or to improve other urban issues (Zanella et al., 2014). For example, urban IoT could enable the route planning in advance to reach the office or to discover the time of day in which one could do a shopping trip to the city centre without encountering traffic. The sensing capabilities and GPS installed on modern vehicles along with the sensing of different conditions in a given road, would allow these type of situations to happen (Zanella et al., 2014).

One important aspect to mention is that despite the fact that these kind of technologies are still in a early stage, there are real-life examples of already implemented systems. Recently, BMW developed an intelligent informatics system (iDrive system) that provides intelligent driving directions based on environment data (vehicle location and road condition) provided by various sensors and tags (Qin et al., 2013).

Finally, Smart Cities and IoT have proven to be a great asset to improve different areas of a city and a citizen's daily life. However, with great power comes great responsibility. Given the fact that, for example, every IoT technology will rely largely on the collection of personal and private information, protecting this data is a crucial and important for the evolution of Smart Cities. In fact, studies have revealed that the number of attacks on IoT entities, when compared to the traditional ones, is appearing to be much higher (Roman et al., 2011; Li, 2013; Ting and Ip, 2013). Therefore, IoT standardization, such as the definition of privacy and legal interpretation, which are still not clearly defined, should be worked upon to prevent privacy issues (Xu et al., 2014).

2.2 AMBIENT INTELLIGENCE

AmI should be given importance for the promotion of Smart Cities (Fernandes et al., 2018). AmI is a digital environment that brings intelligence and improvement to our daily lives by continously acting and reasoning upon the sensed data (Cook et al., 2009). In fact, a Smart City should embed AmI by creating several distinct sensorization levels in cities, as shown in Figure 2. At a first level, there are Application Programming Interfaces (APIs) that offer relevant information such as road conditions and the weather. On a second level people can wear devices that makes them a citizen sensor, contributing for the extraction of relevant data. The third and final level comprises the city in itself and the methods for extracting actionable data from the environment in where one lives and stands (Fernandes et al., 2018).

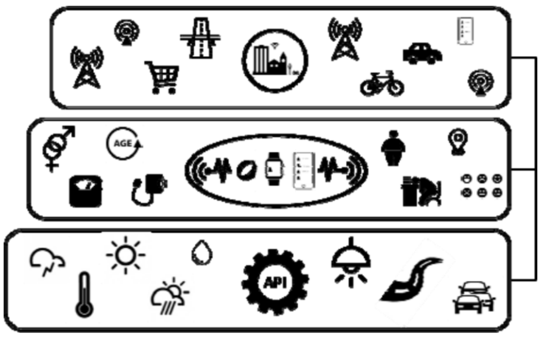


Figure 2: The scale levels for data collection and AmI in Fernandes et al. (2018).

When it comes to the AmI evolution, the evolution of technology played, naturally, a very important role. Initially computers were very expensive and difficult to understand and a single computer would typically be used by numerous individuals. This paradigm changed in the eighties with the PC revolution, allowing each individual to possess its own computer. Nowadays, as industry progressed and costs dropped, one individual often has access to more than one computer - naturally more complex and capable than the previous ones. Moreover, in current times the access is not limited to only just computers. This means that since the miniaturization of microprocessors, computing power is embedded in our daily lives. Whether present in mobile phones or cars, these devices are often something that we take with to travel outside our homes (like mobile phones or smartwatches) or to help us finding the best route to our destination (like cars and their GPS Navigation). All this evolution, namely in faster computation with reduced power along with an increased availability, made the realization of AmI possible (Cook et al., 2009).

AmI algorithms normally follow the same logic. Firstly, the objective is to gather data by sensing the environment's and users' state with the use of sensors. Following up, the use of AI techniques allows the reasoning upon the sensed data. Finally, acting has the objective to make AmI algorithm reach its goal, by executing actions that affect the end users (Cook et al., 2009). Shortly, AmI has three phases: **sensing**, **reasoning** and **acting**.

Sensing

Sensing is where AmI starts. In order to have some sort of effect on an individual's daily life, it is necessary to firstly collect data about this individual. This way, sensing is the process of perceiving the environment by collecting data effectively so that the reasoning and acting algorithms have practical use (Cook et al., 2009). Effectively means not only to sense with the most accuracy possible, but also to have the adequate data processing.

Sensors can be used for different purposes and in different ways. Some sensors have been designed for the detection of chemicals and humidity sensing (G.Delapierre et al., 1983) or to determine readings for light, radiation, temperature, sound, strain, pressure, position, velocity, and direction, and physiological sensing to support health monitoring (Ermes et al., 2008; Stanford, 2004). Others were built to detect Bluetooth-enabled mobile phones carried by the participants in a festival (Larsen et al., 2013). Alternatively, a way of detecting individuals, can also be by the use of wearable devices, such as Radio-frequency identification (RFID) tags that along with RFID readers enable the monitorization of the tagged objects. Also, the use of video-based techniques has been used for sensing purposes (Yuan et al., 2011).

Reasoning

Reasoning must operate with the data gathered by the sensing. In order to make this data useful, reasoning algorithms must be responsive, adaptive and beneficial to the problem that is being addressed. Therefore, these algorithms include different types of reasoning (Cook et al., 2009). They can vary from Modeling or Decision Making, to, in a more specific way, algorithms that enable crowd density estimation.

In terms of Modeling, the main goal is to build a model that is adaptable to a certain problem, by detecting anomalies and changes in patterns. As for Decision Making, the goal is to have automated decisions. For example, according to Mozer (2004), one of the few fully-implemented AmI systems, has the goal to determine ideal settings for lights and fans in a home, by using neural network and a reinforcement learner. Typically most of these algorithms use ML techniques, as they need to be responsive to changes and perfom prediction tasks.

Concerning crowd density estimation, there are several approaches that one can take. In the case of Schauer et al. (2014), in order to reduce the number of false-positives when sensing the crowd, a hybrid approach is used, considering both the RSSI value and the time when a MAC address was captured. By using two sensors, and comparing the time delay of a certain MAC address between the two sensors along with a certain threshold defined for RSSI, the authors prove that the possibility to reduce the false-positive rate is increased.

Acting

The last typical phase of a AmI system, is Acting. It allows the connection between the real-world and all the work done by the previous phases through the execution of actions that affect the system users. It can be done by robots or simply by notifications or interactions (Cook et al., 2009; Ramos et al., 2008).

Conclusion

One main goal of AmI is to make pervasive computing, ubiquitous computing and context-aware computing a reality. Indeed, shortening the gap between humans and computing devices, would allow the wide spread of this technologies through a area or a group of people, making possible the development of devices without explicit operator control (Cook et al., 2009; C.Augusto et al., 2010). In the context of this dissertation, it is crucial to reduce the human-computer interaction, so that the system can infer situations without the active role of any person.

Finally, AmI is a paradigm that can bring multiple benefits to our daily lives and in different ways while still being adaptive, as shown above. Every phase has its own purpose and are mutually dependent. Nevertheless, similarly to Smart Cities, there are privacy and security challenges that must be acknowledged. On one hand, the fact that every AmI system requires some sort of sensing on a user's environment or daily life, even with potential benefits, brings up privacy issues. Moreover, in 2003 a survey showed that privacy protection was more important to the participants than any potential benefits provided AmI technologies. Nonetheless, there is also a city whose evolution was enhanced by the open acceptance to loss of privacy (O'Connell, 2005). On the other hand, issues like the AmI systems performing the wrong actions and forcing humans to extra work - having the opposite intended effect - are an aspect that makes the deployment of these systems a process that requires caution.

2.3 WI-FI PROBING

These days, the majority of smart devices are equipped with Wi-Fi communication interfaces. This type of interface is normally used for enabling internet access and has been widely adopted by any kind of smart device. In recent years, smart devices usage has suffered a huge increase (Zhou, 2017/09; E.Longo et al., 2018). For example, in 2016, the shipment of China's smartphones represented 95.7% of China's total mobile phones shipment in the same period (Zhou, 2017/09). Other valuable facts are that the number of cellphones is now bigger than the actual number of people in the world, according to United Nations (UN)'s International Telecommunications Union (ITU), the World Bank, and the UN (M.Murphy, 2019), or that the number of smartphones users worldwide is increasing every year, as show in Figure 3. Therefore, Wi-Fi has been widely used and its usage is increasing at a fast pace.

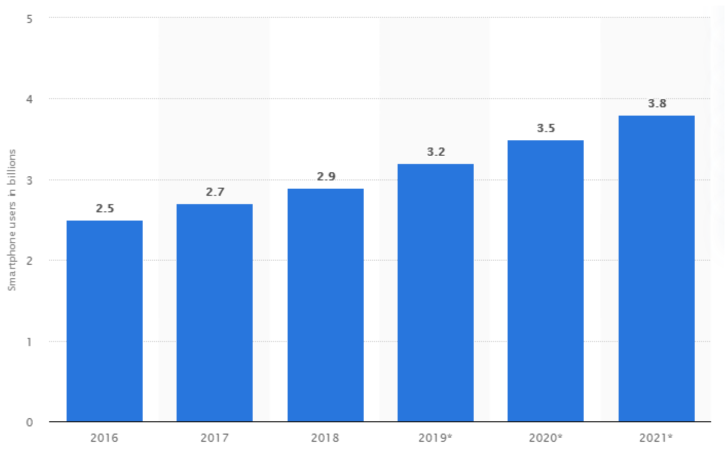


Figure 3: Number of smartphone users worldwide from 2016 to 2021 in Holst (2019).

Smart devices that have a Wi-Fi interface periodically send Wi-Fi probe requests in order to discover which wireless networks are available for connection (L.Oliveira et al., 2019). This mechanism is defined by the IEEE 802.11 standard as an active mechanism or an active scan (Zhou, 2017/09; Freudiger, 2015). These probe requests are constantly being sent whenever the smart device's Wi-Fi is enabled, sending information containing for example the device's MAC address. It should be noted that the rate at which a device sends this packets depends on the Operative System (OS) used on it (Freudiger, 2015). To sum up, this communication process that happens every time the device's Wi-Fi is enabled, allows the sniffing of the probe requests, allowing the detection of devices nearby a certain sensor (Zhou, 2017/09). This opens the possibility to use smart devices' probe requests as a way to get insights into numerous areas like crowd density

(Schauer et al., 2014), shopping habits (Barbera et al., 2013) or traffic forecasting (Ares et al., 2016).

Over the past few years, the MAC address, which is sent along a probe request, has suffered changes due to privacy issues. At first, the MAC address of each smart device was unique and immutable, which lead to a lot of contest regarding privacy dangers. Nowadays, the main strategy adopted by manufactures to fight this issue is the randomization of the MAC addresses (Freudiger, 2015; L.Oliveira et al., 2019). There were other proposed approaches, which did not have continuity: Beresford and Stajano (2004) suggested changing MAC addresses over time, whereas Greenstein et al. (2008) proposed removing link-layer identifiers all-together. In L.Oliveira et al. (2019) the randomization of MAC addresses is defined as *"a strategy that is intended to prevent potential observers from identifying which mobile devices are within reach of a sensor"*. Moreover, details like device, manufacturer, and operating system version influence how the MAC address is or not randomized (L.Oliveira et al., 2019). Martin et al. (2017) identified that most devices, especially those with an Android OS, do not implement randomization of MAC addresses in any way. There are also studies that show that the MAC randomization, in the case the iOS 8.1.3 randomization mechanism, can be defeated (Freudiger, 2015). Besides that, others have showed that is possible to estimate the number of mobile devices present at a certain place and time through a solution that is immune to MAC address randomization strategies (L.Oliveira et al., 2019).

To conclude, Wi-Fi is a growing technology that is increasingly more integrated in ubiquitous computing. Its usage in most of today's devices, opens new possibilities on how to implement crowd sensing. Even though MAC address randomization is feature that does protect user privacy, it is still an on-going process, as many devices still don't have it implemented, or because there are methods to get around it. Therefore, finding a way to implement crowd sensing without compromising one's privacy while still getting the most of the fact that smart devices are constantly sending probe requests, is one of the objectives of this dissertation.

2.4 CROWD SENSING

One big goal of this dissertation is to sense the crowd, gathering data either for calculating density of people at certain points of interest, or even forecast these values. Crowd sensing can be done in multiple different ways, and consequently, the objective of this section is to review different approaches to this problem. There are some key aspects which will be taken into account in order to have some comparison parameters:

1. Sensing
   - The kind of device used for the sensing;
   - Technologies involved;
   - The collected data structure;
   - The role of the user in the sensing (an inactive role or opportunistic and participatory sensing) (Sun et al., 2016; Petkovics et al., 2015; Ganti et al., 2011; Guo et al., 2015);
   - The scale on which the approach is being tested;
   - The storage method used;
   - What kind of privacy measures are used;
   - Pre-processing.
2. Reasoning
   - Data Modeling;
   - ML Models.

2.4.1 *Sensing through Bluetooth*

The paper produced by Larsen et al. (2013) presents an approach to Crowd Sensing at a large-scale music festival with over 130 000 participants for obtaining spatio-temporal data. In order to do so, the authors use several Blutetooth scanners to discover Bluetooth-enabled mobile phones carried by the participants. In a second phase, they reveal what kind of algorithms used for analysis purposes.

Methodology

In the referred study, the authors' method to obtain data is to use Nokia N900 smartphones with custom software for detecting Bluetooth-enabled devices in proximity, as crowd sensors. As Larsen et al. (2013) states, this methods has some benefits and drawbacks. On one hand, the Bluetooth scanners, functioning as master devices, scan passively continuously for devices without any active participation on the user side. On the other hand, since Bluetooth is a short-range low-power protocol for implementing Wireless Personal Area Networks (WPAN), the range in which devices can be discovered is very limited. Besides that, the discoverability of a device can depend on its operating system and additionally on the fact that normally it has to be set manually by the user.

Regarding the scanner itself, besides having the Bluetooth module in it, it offered 3G communication, data storage, battery power and GPS tracking. This allowed the gather of data in real time whilst having power in case of a event's power outage, or the tracking of the device in case the it got lost.

Data

In terms of data storage, the scanned data was stored locally using the SQLite database on the device and also uploaded to server, always depending on the network availability. In order to maintain availability and robustness of the system, the authors present a not so clear approach. Approximately there were two scans per minute, and if the devices did not upload data to the server during a certain amount of a time, the device would be rebooted either by issuing a command via Bluetooth or by manually turning them on and off. It is not clear if the command via Bluetooth was triggered automatically by the software itself. Additionally, having any kind of manual work in this case is not the best solution when the objective is to maintain availability and to implement an AmI system. Nevertheless, an periodical reboot would occur every 24 hours to minimize the effect of any device not working properly.

As for the data structure, the collected data is a time series of events, where each entry is characterized by the time, scanner ID and the Bluetooth MAC address of a discovered device. The authors argue that the RSSI was not registered, since using that measure for distance calculation has an accuracy which can depend on the type of environment (Larsen et al., 2013).

Regarding privacy issues, even that according to Larsen et al. (2013) the Danish Data Inspectorate considered that that information didn't enable the linkage of the device to a person, the authors still made sure to hash the MAC addresses after extracting information about the vendor. Also, the human-readable identifiers on each device were not recorded for a faster scanning and for anonymity issues.

Data collection and Analysis

In order to provide sufficient coverage of all the relevant areas, the authors used 33 Bluetooth scanners placed in the vicinity of stages like shops, bee booths and mixing areas of the stages. This allowed the coverage of rich spots and also the placement in spots where a power source was available. The scanned data was uploaded in real time via a 3G network. Naturally, in events where high number of mobile phones are present problems with the mobile network tend to occur. So some sensors only uploaded data when they got their connection reestablished- normally in the early morning hours. With this being said, only 7 of 33 sensors were able to run without the need of being maintained one or more times during the 8 days of the festival. With this strategy, the authors were able to collect 1 204 725 observations during the 8 days of the festival. Taking into account the number of unique devices discovered (8534), 6.5% of the overall population was actually observed. Therefore, 8534 entries in the dataset does provide some window to discover patterns in participants mobility.

Micro Groups Modelling

Taking into account that the authors claim that given the fact that the radius of Bluetooth is limited to about 10 meters for the transmitters used in most of the mobile phones and that the case study in Larsen et al. (2013) is to get insights on the internal structure of a crowd, the authors analyse the data at two levels. One of them is Micro Groups Modelling.

In this level, the objective is to discover if people move alone or in groups and how groups are different in Larsen et al. (2013), micro groups are sets of people frequently co-occurring in the same area and time frame (spatio-temporal bin). After dividing the timeline of the entire festival into 1076 x 10-minute temporal bins and with each scanner creating a spatial bin, the authors started by out of the 8534 unique devices discovered, removing the ones that were seen in less than 10 temporal bins or less than 3 spatial bins. Either because being in less than 10 temporal bins didn't provide enough data or because being in few spatial bins meant that the scanned device was a stationary one (such as crew laptops). For all common occurrences, a directed graph was constructed having the weight of each link estimated by the number of co-occurrences of the participant A with participant B, divided by total number of occurrences of the participant A.

This method had many constraints: from 130 000 participants, participants had to be seen in the same 10 meter radius within the same 10 minutes at least half of the times they were observed in total and in at least 3 different locations, to ensure meaningful data for the purpose. Regardless, the authors were still able to detect micro groups, having in the end 500 people moving around while belonging to a particular structure.

Macro Modelling

In this case, the objective was to combine the spatio-temporal traces with the bands schedule, in order to find out which concert each of the participants attended (Larsen et al., 2013).

After assigning meta information to each show (genre, playcount, etc), the authors first approach was to calculate the pearson correlation between the number of unique devices found during each concert and the logarithm of the playcount of the band. A strong correlation was expected, as the number of people at a concert seems to be correlated with how much the band is listened to (playcount). In the end, for Larsen et al. (2013) the Pearson's correlation proved that people's choices regarding concerts are not really correlated to the band's popularity. So a more complex model was used.

In terms of pre-processing, the authors transformed the time series data into a binary attendancy table, having a matrix that maps each participant to the concerts attended by the person. After that, they transformed it into a matrix that indicated how many times each participant was scanned at a certain concert. Finally, the table is again transformed to a binary matrix by filtering all the entries which had a lower value than a certain threshold. To remove outliers, some similar processing was done as in the Micro Modelling. Participants who participated in less than three concerts and Bluetooth devices recorded in the same location throughout the festival were removed, having remained 5127 attendees for further analysis.

Regarding the model, the data was fitted in an Infinite Relational Model (IRM) to reveal the pattern's of people behaviour at the festival. The model's stability and generalizability are proven by different number of measures used to evaluate it as well as the usage of non-complete datasets and the insertion of randomness in the testing phase. Some interesting insights were obtained in this process. For example, in Figure 4 it's possible to see that the user cluster 5 is highly associated with the concert cluster 15 or 20, meaning that people in cluster 5 attended concerts from these clusters.

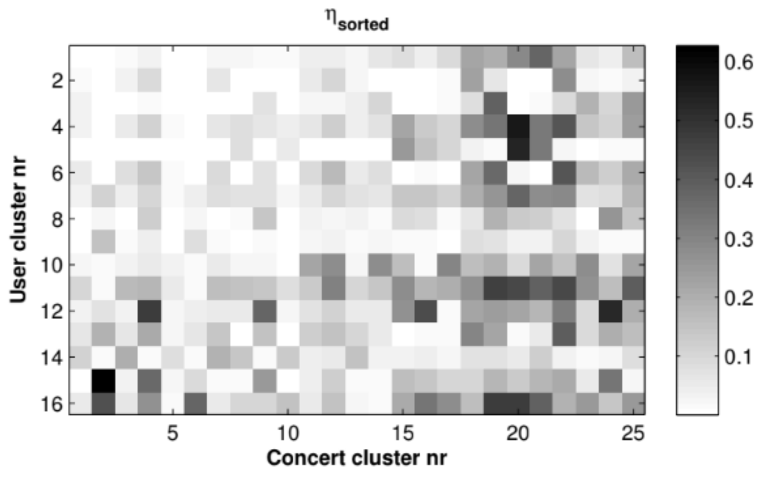


Figure 4: Figure that demonstrates the relation between user and concert clusters in Larsen et al. (2013).

Discussion and Conclusions

To sum up, this paper proved that even though Bluetooth based approaches to crowd sensing can have some drawbacks (like having to be in discoverable mode for example), it has proven in real-life its feasibility. Using Bluetooth signals as a way to detect persons might not be the best solution, since these signals have a really short range of detection. Additionally, the analysis of the spatio-temporal data proved to reveal interesting insights when wanting to discoverer relations between each user that was sensed and also between the user and a concert. The issue is that nowadays some smart devices have the MAC randomization enabled, so the present approach wouldn't work so well nowadays. Nevertheless, the authors made sure to protect one's privacy by hashing the MAC address of each device. The device used was Nokia N900, which offered 3G communication, data storage, battery power and GPS tracking. Having a smartphone to essentially sense Bluetooth signals and be responsible for the storage of data is not worth the price of the device. Finally, the authors say that all the data and its insights were being displayed in a 46 inch monitor during the festival and that the participants were attracted by it, which can reflect on how people are becoming more open-minded towards these kind of technologies (Larsen et al., 2013).

2.4.2 *Sensing through Bluetooth and Wi-Fi*

In Schauer et al. (2014), the main objective is to present pratical approaches of crowd sensing, such as crowd density and pedestrian flows estimation. The methods are tested in a real-life scenario. In fact, all the sensing occurs in a major German airport, which is not only beneficial because of being a real scenario, but also because the authors use ground truth information provided by the security check to test the feasibility of their approach.

Methodology

During the period of 16 days, the authors obtain data by using 2 time-synchronized laptops to discover devices with Bluetooth or Wi-Fi enabled. These two laptops were placed in two different locations: one located in the public area, and the other inside the area after the boarding pass scans - the security area. As in Larsen et al. (2013), this papers draws attention to some of the disadvantages of using Bluetooth as a way of crowd sensing. On the other hand, the wireless local area network technology, know as Wi-Fi, does present some advantages in crowd sensing when compared to Bluetooth. Firstly, instead of having a communication range of 10 meters like Bluetooth, it can go from 35 meters in indoor environments to 100 meters for outdoor environments, which facilitates the identification of Wi-Fi enabled devices. Besides that, the authors claim that in different mobile devices the active scan - which is the process that allows that detection of the Wi-Fi probes emitted by such devices - occurs at least once within two minutes.

Data

Regarding data structure, the data collected on these experiments had fields like the RSSI value, the MAC address and the entry's time stamp. In the 16 days, there were over 11 million probe requests captured in the public area and about 8.5 million in the private one. Daily, the authors were able to capture 9995 unique Wi-Fi MAC address and 357 unique Bluetooth addresses. With this data, the Schauer et al. (2014) concluded that Apple devices send out probe requests more frequently when compared to other Android devices.

Crowd Density Estimation

The Schauer et al. (2014) define crowd density estimation as: *amount of people per unit of area within a certain time interval*. Therefore, estimating this measure in the node's coverage is done by counting the amount of unique devices during a certain period of time. In order to evaluate the results, the authors expected that whenever there was a higher frequency of board pass readings, the crowd density should be higher too. In Figure 5, it is possible to see that the correlation exists and that there is a higher density of people in the public area, which was expected.

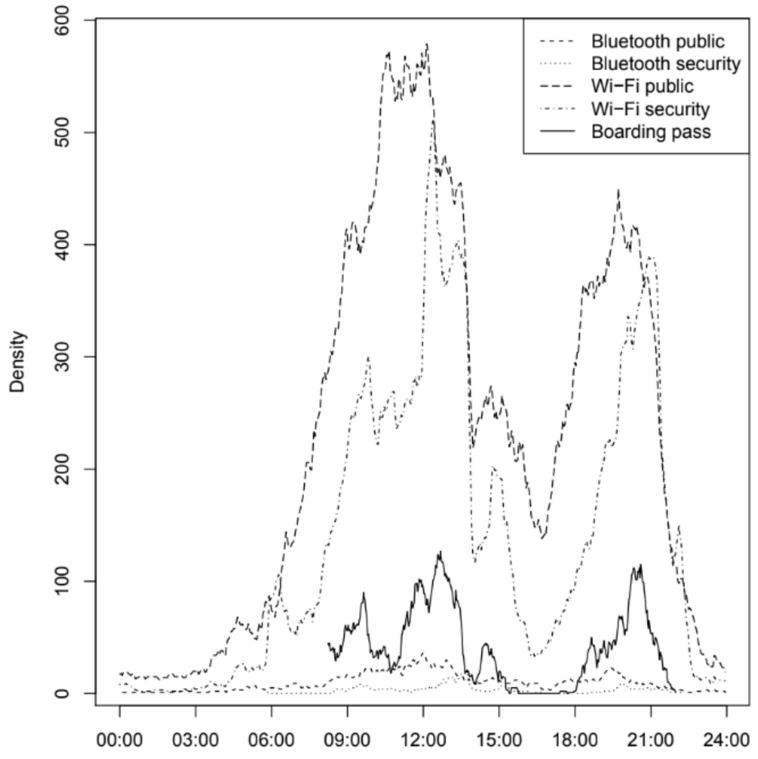


Figure 5: Daily data of Wi-Fi, Bluetooth, and boarding pass readings in Schauer et al. (2014).

Although there was no data representing the ground truth, it is possible to conclude that the density of the crowd had some relation with the actual number of people that were on the boarding pass readings.

Pedestrain Flow Estimation

The Schauer et al. (2014) define pedestrian flow as *"amount of people moving one way through an area of interest within a certain time interval"*. In order to estimate this measure, the paper presented four different approaches: Naive Approach, Time-based Approach, RSSI-based Approach and, finally, an Hybrid Approach.

The simplest way, the Naive Approach, was to count the number of unique MAC addresses which were captured at the both sensors (s1 and s2) within a specific time interval. Two big problems arised from this method. One of them was that with this method it wasn't possible to determinate the direction of a person. The other, was that a device present in the overlapping zones of each sensor would be accounted as a person moving, increasing the rate of false-positives. The first problem was solved using a Time-based Approach, in which the specific time interval had to be positive (between the first and last node). In the second case, the authors presented a RSSI-based approach in which the pedestrian flow is calculated as the number of unique MAC addresses in a certain period of time that have the RSSI value over a certain threshold for both nodes. Similar to Larsen et al. (2013), Schauer et al. (2014) refer that the RSSI can be influenced by numerous factors, such as the environment itself or the device characteristics. Therefore, defining a reasonable threshold was the key here, but the Schauer et al. (2014) concluded that it was necessary a more of a Hybrid Approach. Finally, this method uses both the RSSI value and the time of when a MAC address was captured. This lead to the following method to calculate pedestrian flow: number of unique MAC addresses captured containing a positive time delay between the sensors and at least one capture with the RSSI value over a certain threshold for both nodes. This way, this method ensures that the sensed pedestrian is moving from one point to another, while also reducing the possibility of false-positives in case the pedestrian was in the overlapping detection zones of the nodes.

This final approach offered some good results, but in order to improve them Schauer et al. (2014) decided to only capture data when the security gate was open. This ensured that the captured data could be compared with the ground truth having less false-positives and therefore a focused estimation. Figure 6 presents the pearson correlation (which indicates how much one observation is correlated to another, from a scale to 0 to 1 (Schauer et al., 2014), of each approach and also with the optimal time shift for sensoring being implemented.

	Bluetooth	Wi-Fi naive	Wi-Fi RSSI	Wi-Fi time	Wi-Fi hybrid
max	0.73	0.82	0.93	0.93	0.93
average	0.44	0.41	0.56	0.47	0.57

	Bluetooth	Wi-Fi naive	Wi-Fi RSSI	Wi-Fi time	Wi-Fi hybrid
max	0.79	0.86	0.91	0.91	0.91
average	0.53	0.61	0.74	0.63	0.75

Figure 6: Correlation coefficients for each approach and with different focused estimations in Schauer et al. (2014).

Note that Schauer et al. (2014) performed different times shifts for each estimation and then calculated the correlation coefficient for each time shift.

Discussion and Conclusions

The authors were able to prove that for crowd sensing the Wi-Fi is more reliable than Bluetooth not only because of its characteristics but also because more modern devices use it nowadays. Besides that, Schauer et al. (2014) proved that it is possible to estimate crowd density and pedestrian flow using the Wi-Fi and Bluetooth as a way sense to a certain area. Additionally, the correlation results between the sensed data and corresponding boarding pass readings, showed the feasibility of the method implemented, having an average of 0.75 in the Pearson's correlation factor with their best approach. The authors also show an interesting way to counter the overlapping coverage area of various sensors. Moreover, the RSSI value was used in the calculation of the Pedestrian Flow, but because of its volatile values the method used to counter this issue was to define a threshold as a way to filter results and combine this metric along with others. The way this threshold was defined is not explained explicitly. Regarding the sensing device used, using a cheaper option (e.g. a micro controller), even if with a higher number of sensors deployed, would have the same end result. Finally, either on the Crowd Density or Pedestrian Flow estimation, the authors claim that MAC randomization in 2014 didn't raise a problem when wanting to recognize a certain device. But, time has passed and Apple is not the only integrating MAC randomization mechanism in their devices now. Despite being proved that this mechanism is still on early stages, it should be a factor to take into account when wanting to do some sort of crowd sensing.

2.4.3 *Sensing in a SmartCity context*

In Ares et al. (2016), the main objective of crowd sensing is to address and improve issues that could be solved in a Smart City's context. In fact, a mobility monitoring system is presented as way to improve several areas of city, like traffic or security issues inside buildings. Whilst the sensing phase applied to the four different areas of study is the same, the authors then use different methods for analysis, processing and modelling algorithms for the different cases.

Methodology

The sensing device used in Ares et al. (2016) is a single-board computer, based on a Rasperry Pie board. As the objective is to sense each individual through the the possesion of digital devices, the board has a Bluetooth and Wi-Fi cards, allowing the board to detect Wi-Fi and Bluetooth probe requests.

The monitoring system developed by Ares et al. (2016) has an architecture of six modules. The first one is the software implemented in the sensor in order to detect Bluetooth and Wi-Fi devices along with extraction and encryption of the devices' MAC address. It is also responsible for publishing this data to the server. Secondly, there's a module to act as gateway between the sensors and the server, making possible the communication between the sensors themselves, and the sensors and external networks. Then, there are modules responsible for the control of the sensors from distance and for the storage either in a local server or in a cloud-based storage, which have services for data mining, ML, forecasting and visualization.

As in any other crowd sensing case, privacy was also one of the thematics discussed by Ares et al. (2016). In this case, the authors argue that as soon as a MAC address (either on a Wi-Fi or Bluetooth probe) is captured from a device, it is immediately encrypted using the SHA1 hash algorithm. Apart from that, the system proposed only publishes information about general data, meaning that neither individual data is shared nor data relating to a specific device.

Analysing people's mobility in a discotheque

One of the use cases that Ares et al. (2016) tested their approach in was a discotheque. A place where from common knowledge almost every one has a smartphone and where the change, in terms of mobility, is very fast. In order to collect the data, five devices were installed in a discotheque, one in each main room and the others on the main entrance and outdoor terrace. It is also mentioned - even though the authors don't expand on it - that given that fact that the eletromagnectic scenario was heavy, the testing of the device was more difficult.

In terms of amount of data, a total of 2200 different devices were detected in one of the busiest nights. After collecting the data, the main goals were to find group behavioural patterns in the data, which could optimize the decision making for marketing or security issues in the disco. To do this, Ares et al. (2016) first extracted several variables, shown in Table 1, from the data so that they could then apply clustering methods.

Table 1: Variables extracted from the data in Ares et al. (2016).

Variable name	Description	Type
entrance_time	First date/time the device was detected	Date
out_time	Last date/time the device was detected	Date
stay_time	Number of seconds the device has been in the disco	Integer
abs_time_node_X	Number of seconds the device has been detected in every node X (from 1 to 5)	Integer
relat_time_node_X	Percentage of the time the device has been detected in every node X (from 1 to 5)	Float
relat_night_time_node_X	Percentage of the time the device has been detected in every node X (from 1 to 5) regarding the whole time	Float

The clustering method used in Ares et al. (2016) was a Self Organizing Map (SOM). This method uses a feed-forward neural network that by using an unsupervised training algorithm and nonlinear regression techniques, is able to cluster the data and finally find interesting relations between the set of variables. By transforming the resulting data of the clustering method into a Unified Distance Matrix (U-matrix), the authors generated a graph that visually represented the multi-variable dataset in a two-dimensional display. Using this U-matrix + SOM approach, several graphs were produced. For instance, one of the graphs produced by SOM was able to show that in room of the sensor 112, the Main Room, the number of devices staying there a higher percentage of time was the highest between all rooms, followed by the sensor 122, as demonstrated in Figure 7.

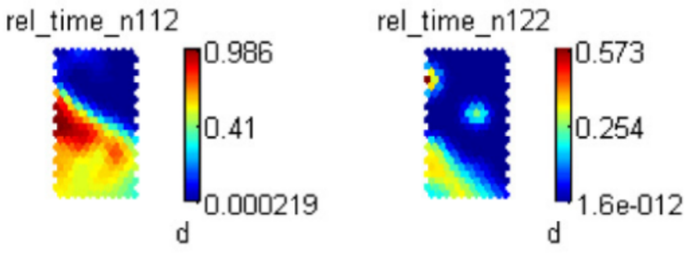


Figure 7: SOM plane analysis for the variable referring to the time a device spent in a nodes 112 and 122 in Ares et al. (2016).

Through this use case it was possible to gather and process data that can be useful for different purposes. The clustering method used showed interesting insights about people's mobility in different places inside the disco, making it possible to address issues on marketing or energetic efficiency in the context of Smart Cities (Ares et al., 2016).

Traffic Forecasting

One other interesting use case in Ares et al. (2016) is regarding Traffic Forecasting. Sensing is not only useful to get insights or relevant live information about different environments, but also to forecast relevant situations like whether traffic is going to be heavy or not at certain hours in certain places.

Compared to the previous use case, in this one there was real data provided by the Spanish traffic management agency, so even though detecting cars at high speed seemed to be an issue, the validation of the approach in Ares et al. (2016) was possible by comparing the real data with the data collected by their approach. In order to collect the data, six nodes were placed in six different positions along different roads (that had loop detectors installed by the traffic management agency), as illustrated in Figure 8.

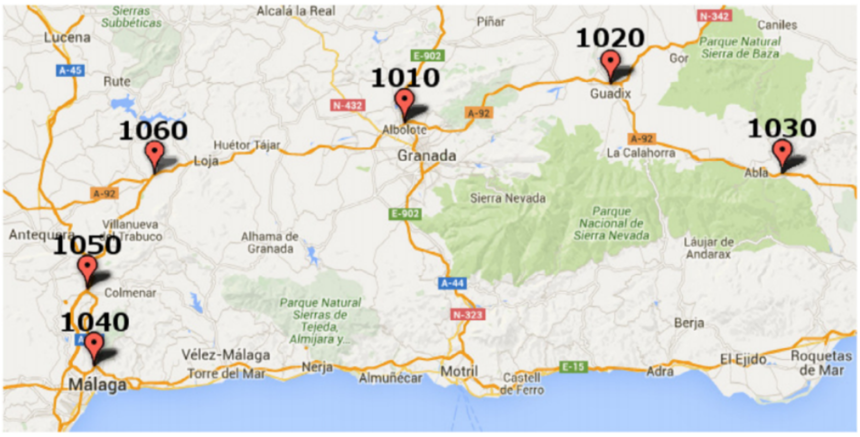


Figure 8: Location of the six sensors in Ares et al. (2016).

Since one of the goals in Ares et al. (2016) was to forecast the traffic, the sensed data had to be processed so that there was information about the number of cars that passed a given point in a certain period of time, organizing the data in a time series way.

Given this processing, the authors in Ares et al. (2016) tested four different methods (using the forecast package of the R language (Hyndman and Khandakar, 2008) for forecasting the traffic, enabling the prediction of the number of cars that pass in different points in certain periods.

As stated in Ares et al. (2016), the method that had the most accurate behaviour was the Exponential Smoothing State Space Model (ETS). For this experiment the data used was from the node 1010, having a total of 1920 values. Figure 9 demonstrates that the ETS method was able to forecast the values with good accuracy when compared to the real data.

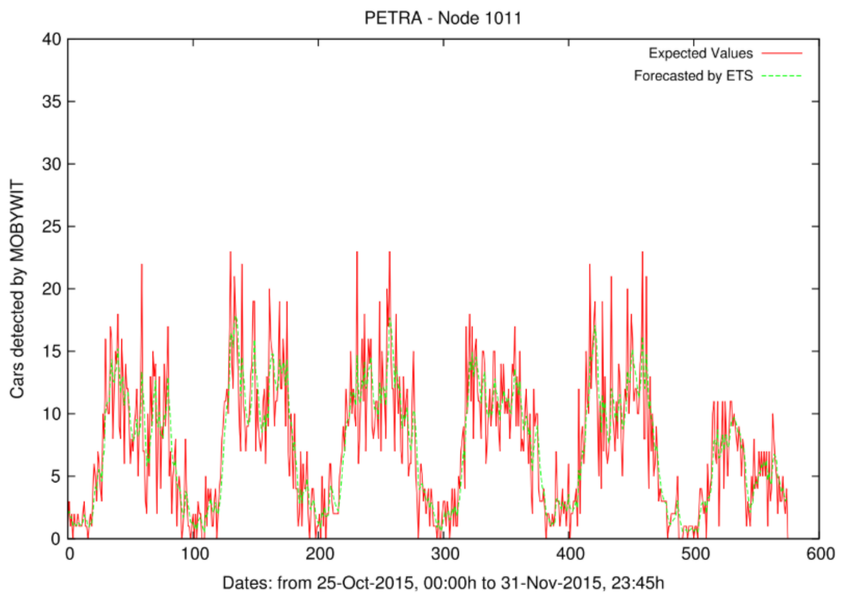


Figure 9: Expected vs forecasted number of cars in one of the highway points in Ares et al. (2016).

Finally, this use case showed that it is possible to sense data even from cars that are moving on a high speed and also suggest which forecast method might work best for time series problems. Having this embedded in a Smart City would improve allow the route planning or the improvement of traffic jams, for example.

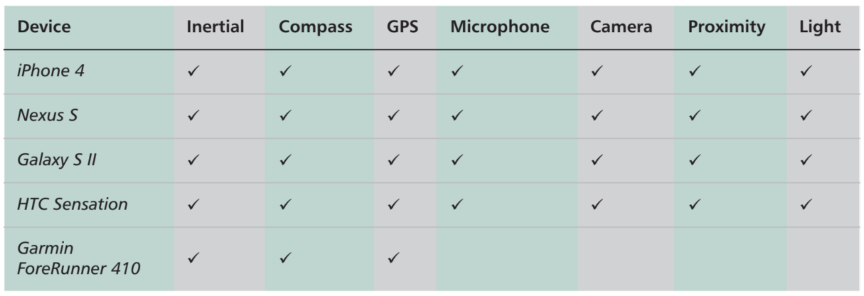
Discussion and Conclusions

The approach in Ares et al. (2016) used a singled-board computer to detect Wi-Fi and Bluetooth probe requests in order to do the crowd sensing. Privacy-wise, the authors used the SHA1 hash algorithm in order to encrypt the MAC address as soon as it was captured from a device, protecting any individual's data with this method. In the first use case, the use of clustering methods proved to be valuable when wanting to show group behavioural patterns in the extracted data. Nevertheless, in this case there were sensors distributed in various places of a discotheque, but it was not clear on whether there was overlapping of the sensors coverage area or not, and if it was something that was taken into account. A possible way to do this could be using the RSSI value to filter unwanted values. In the second use case, the proposed solution allowed the forecast of traffic related data. It was referred that vehicles travelling at high speed could be difficult to detect, but the results showed that when compared to the ground truth information provided by the loop detectors, the sensed data was accurate. Finally, MAC address randomization wasn't mentioned at all throughout the paper, which means the authors didn't consider the possible effects of this mechanism on their results.

2.4.4 *Mobile Crowd Sensing*

All the approaches referred before have one point in common: they use passive sensing techniques, allowing the detection of users without them playing an explicit role in the gathering of data. Mobile Crowdsensing (MCS) is a paradigm that offers a different approach, where individuals with sensing and computing devices collectively share and extract data for a common purpose (Ganti et al., 2011).

Instead of using typical IoT devices, such as RFID tags (Cook et al., 2009) or single board computers (Ganti et al., 2011), this approach defends that mobile devices can be more useful. Commonly equipped with various sensing capabilities and also with powerful computation, as Figure 10 shows, mobile devices can offer different possibilities when it comes to crowd sensing.



Device	Inertial	Compass	GPS	Microphone	Camera	Proximity	Light
iPhone 4	✓	✓	✓	✓	✓	✓	✓
Nexus S	✓	✓	✓	✓	✓	✓	✓
Galaxy S II	✓	✓	✓	✓	✓	✓	✓
HTC Sensation	✓	✓	✓	✓	✓	✓	✓
Garmin ForeRunner 410	✓	✓	✓				

Figure 10: Some of the sensing capabilities of different mobile devices in Ganti et al. (2011).

MCS has characteristics that offer not only new opportunities but also big challenges. When it comes to advantages of this paradigm, mobile devices have a bigger variety of sensing capabilities and more computing, storage and communication resources than a traditional IoT sensor. Besides that, the fact that mobiles devices are already deployed in one's life may lead to a easier usage of this devices as sensors to our everyday quotidian, improving it in several areas. Instead of using specific sensors for example to collect traffic data, a possible solution could be using smartphones carried by drivers to collect the traffic data (Ganti et al., 2011).

In terms of drawbacks, the main one is that fact that MCS, either on participatory sensing or opportunistic sensing, involves an active role of the user in order to extract any kind of data. On the one hand, participatory sensing can be defined as sensing that *"requires the active involvement of individuals to contribute sensor data"*. On the other hand, opportunistic is the sensing where *"user involvement is minimal"* (Ganti et al., 2011). In the end, either one of these approaches are inserted into the process know as Crowdsourcing, defined as *"The practice of obtaining needed services or content by soliciting contributions from a crowd of people, especially from an online community"* (Guo et al., 2015). More specifically, if users reported the available parking spots with text or images, that would be inserted into MCS (Ganti et al., 2011). The main issue is that the owners of mobiles devices commonly are not willing to contribute for the sensing, processing and communication of the data, unless they have an appropriate incentive mechanism to do so (Ganti et al., 2011). Besides this main disadvantage, MCS might have problems to deal with the different structure of data produced by different mobile devices (even for the same purpose), privacy related issues or the need of communication with a large number of devices (Ganti et al., 2011).

2.4.5 *Sensing in a Mobile Crowd Sensing paradigm*

Following up on the MCS section, this section will present an approach embedded in this paradigm, where multiple mobile phones are used collaboratively to estimate crowd density through the scan of the environment for Bluetooth devices (Weppner and Lukowicz, 2013).

Methodology

In Weppner and Lukowicz (2013), the method proposed is to sensor the environment for Bluetooth enabled devices through a reduced numbers of users (stationary or dynamic) that are equipped with a Bluetooth scanning mobile phone to determine the crowd density in an area of $2500m^2$. A reduced number of volunteers walk through specific areas during specific times in order to extract all the relevant data. This is tested during three days at the European soccer championship official public viewing event, which has thousands of visitors. In their experimental setup, 10 students are divided in 5 teams of 2 students each, where some of them stand on the same spot (like near entrances), and others walk continuously around the event area, in order to cover all the relevant regions. In terms of the sensing device, each student is equipped with one Android smartphone, that continuously runs an applications that scans for discoverable Bluetooth devices, producing data that is saved onto a microSD card. Each scan is defined as a time interval that contains a dataset with all the unique Bluetooth devices (except the repeated devices), along with other information like the RSSI value, etc. Along with this, all the data extracted can be compared to ground truth information about crowd density with the use of a High Definition (HD) video camera, which can prove (or not) the feasibility of this method.

Feature Engineering

As the authors in Weppner and Lukowicz (2013) want to estimate precisely the crowd density of the whole event area, 6 features were created. In ML, Feature Engineering is the process of creating features based on raw data, in this case from the sensors. This process has the ability to increase the performance of the prediction model, as any model needs to be fueled with relevant data, regardless of the complexity/power of the ML model (Rencberoglu, 2019; Koehrsen, 2018).

This section will explain briefly two of the features created. The first feature, *"Averaged sum of distinct devices discovered by all sensors in scan window"*, describes the average number of unique devices discovered in each sensor for every scan window (snap-shot), which can be seen in Figure 11. An issue that could arise from this method was that the same Bluetooth device could be detected by different sensors, which could give false results. In order to solve this, Weppner and Lukowicz (2013) argue that Bluetooth devices discovered by multiple sensors at the same time don't influence the calculation of the referred feature. One other feature created was *"Ratio of discovered devices in current snapshot to discovered devices in last x minutes"*. This feature allowed the creation of crowd movement insights during every snap-shot. The calculation of this feature was done by dividing the number of occurrences in the union of unique devices discovered by all sensors in a snap-shot, by the number of occurrences in unique Bluetooth devices discovered in the last 15 snap-shots. In the end, a high value of this feature would indicate that in the current snap-shot there was strong crowd movement.

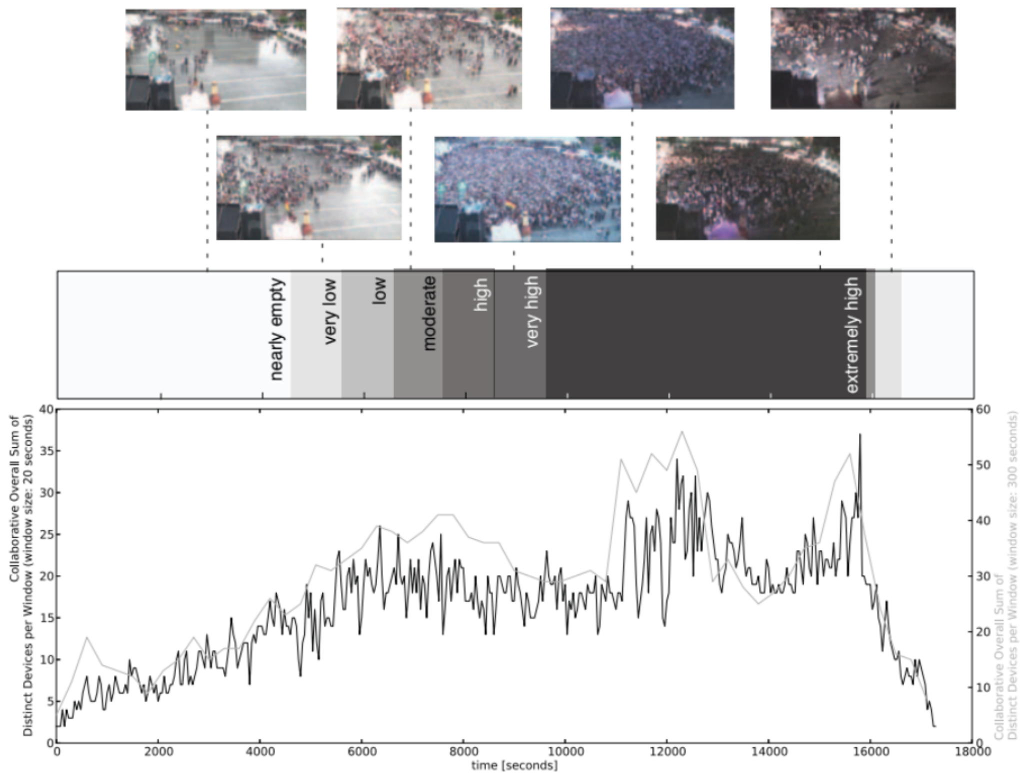
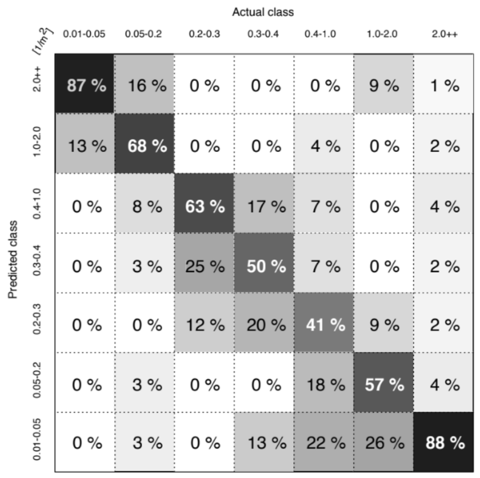


Figure 11: Feature *Averaged sum of distinct devices discovered by all sensors in scan window* along with ground truth crowd density levels in Weppner and Lukowicz (2013).

Model Evaluation

After the sensing and the processing of the data, Weppner and Lukowicz (2013) used a decision tree classifier for classifying the crowd density in six different levels: empty (0.01 - 0.05 people/m^2), very low (0.05 - 0.2 people/m^2), low (0.2 - 0.3 people/m^2), moderate (0.3 - 0.4 people/m^2), high (0.4 - 1.0 people/m^2), very high (1.0 - 0.05 people/m^2) and extremely high (2.0 + people/m^2). The evaluation was done using 10-fold cross validation. The results for each level are illustrated in Figure 12.


Actual class

Predicted class (1/m²)	0.01-0.05	0.05-0.2	0.2-0.3	0.3-0.4	0.4-1.0	1.0-2.0	2.0++
2.0++	87 %	16 %	0 %	0 %	0 %	9 %	1 %
1.0-2.0	13 %	68 %	0 %	0 %	4 %	0 %	2 %
0.4-1.0	0 %	8 %	63 %	17 %	7 %	0 %	4 %
0.3-0.4	0 %	3 %	25 %	50 %	7 %	0 %	2 %
0.2-0.3	0 %	0 %	12 %	20 %	41 %	9 %	2 %
0.05-0.2	0 %	3 %	0 %	0 %	18 %	57 %	4 %
0.01-0.05	0 %	3 %	0 %	13 %	22 %	26 %	88 %

Figure 12: Confusion matrix generated by the decision tree classifier in Weppner and Lukowicz (2013).

An accuracy of 75.3% was achieved, which, if the ground truth data was actually accurate, does provide relevant predictions when it comes to crowd density. But Weppner and Lukowicz (2013) does say that the ground truth information might be noisy, which doesn't contribute for an actual feasibility of the approach and model used. Besides this, Weppner and Lukowicz (2013) also argue that the human body has a high absorption coefficient of the Bluetooth signal, which affects the scanning done by the mobile devices and therefore leading to results that might not be the pretended. Relying on features that are not directly dependent on the absolute number of discoverable devices and using relative features based on the ratio between values observed by different devices, lead to improvements, which is a point to be considered. This process, feature engineering, not only helped on improving the ML model itself, but also with the visualization of meaningful data and to a more robust system overall. Nonetheless, the authors refer that there was a 30% improvement in accuracy when using multiple sensors, compared to using a single device, but don't demonstrate it. Since the presented approach is part of a bigger paradigm, know as participatory sensing, it would have been relevant to know how the volunteers who were using sensors were convinced to be part of the sensing. Lastly, their current approach would have to suffer some changes in order to have the same success, since the MAC address randomization is more present nowadays than it was in 2013.

2.5 MACHINE LEARNING

ML can be defined as a field of AI (see Figure 13) where computers are programmed to learn from data (Géron, 2017; Shalev-Shwartz and Ben-David, 2014). The main advantage of ML over a human's learning is that a computer has the ability to consume huge amounts of data and detect and analyse its patterns that are outside of the human perception (Shalev-Shwartz and Ben-David, 2014). This technological method's evolution arised from the increased data collection and ML algorithm's complexity, as well as the decreased cost of data processing power (Mitchell, 1991). Nowadays, ML is used in many different cases like voice/face recognition, recommendation systems or classification problems. For each kind of problem, there's a different kind of ML algorithm that normally performs better and belongs to one of the three defined ML paradigms: **Supervised, Unsupervised** and finally **Reinforcement learning** (Géron, 2017).

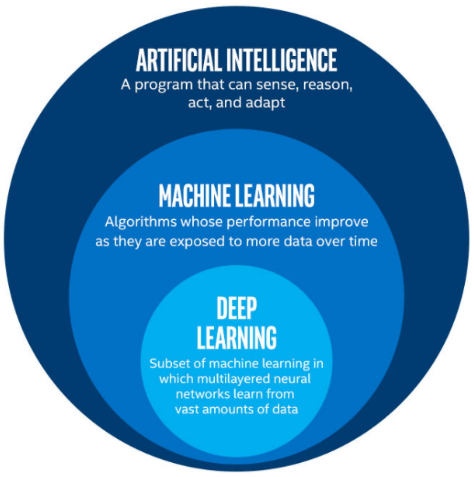

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Figure 13: Difference between Machine Learning and Artificial Intelligence in Singh (2018).

2.5.1 *Learning Paradigms*

ML Paradigms vary according to each type of problem which in their turn have different amounts and types of supervision in training.

Supervised Learning

In the case of Supervised Learning, the data used to train the model contains labels, which are the outputs for each input. The model has a mapping function, which is formed by a algorithm (which differs depending on the kind of problem), that after trained predicts the output data for each input (Géron, 2017; Brownlee, 2020a).

A typical example is a classification problem, where the model needs to predict the correct categorical output, by calculating whether an email is spam or not. Given that the training the dataset contains each email labeled, for each email (input), the model must learn from the training set how to predict the label (Spam or Not Spam), in the most accurate way possible. Another kind of Supervised Learning problem, is the regression problem which happens when a discrete output, like the price of a house for example, has to be predicted by the model (Géron, 2017; Shalev-Shwartz and Ben-David, 2014).

Unsupervised Learning

Unsupervised Learning the difference is that the data used to train the model is unlabeled. The model now tries to learn without any supervision. In this case, the model only has available the inputs without any corresponding outputs, having to discover interesting patterns in the data (Géron, 2017; Brownlee, 2020a). This kind of problem can be divided into main three categories: Clustering, Association Rule Learning and Visualization and Dimensionality Reduction. In the first place, Clustering is grouping up each input such that each one ends in the same cluster. Association Rule Learning is having to pair up groups of data, so that it forms a rule such as people who see the movie X also see the movie Y. Finally, Visualization and Dimensionality Reduction, in the case of Visualization is presenting the data so that it is understandable how the data is organized. In the case of Dimensionality Reduction, it is discovering which inputs reflect better the general dataset, simplifying the data without losing information. (Géron, 2017; Shalev-Shwartz and Ben-David, 2014; Brownlee, 2020a).

Reinforcement Learning

Finally, Reinforcement Learning uses a method based on reward/penalty of each action to train the model. The learning system, called an agent in this context, must learn how to choose the actions that can get the most reward over time, based on its policy. In fact, the policy of the learning system is what defines its behaviour to different situations (Géron, 2017).

One of the most well know examples of the use of this paradigm is the AlphaGo. By implementing Reinforcement Learning algorithms, the robot or program learned how to play the game Go by analysing millions of games and playing games against itself, which allowed the robot to get better over time and eventually beat the world champion Lee Sedol (Géron, 2017).

2.5.2 *Learning Algorithms*

Embedded in each learning paradigm there are a variety of algorithms used for solving typical ML tasks. Below, some of these algorithms are going to be described.

Linear and Logistic Regression

In the first place, Linear Regression is a ML algorithm inserted into the Supervised Learning paradigm. Being used to predict numerical values, this algorithm prediction method is to compute a weighted sum of the input variables (typically features given in a dataset) plus a bias term, which can help the model to make more accurate predictions.

In the second place, Logistic Regression also belongs to the same paradigm referred before, but it is used for classification problems, meaning that it is used to predict categorical outputs. Using a similar equation as Linear Regression, this algorithm outputs the logistic of this result, showing how likely an instance is to belong to a certain class (Géron, 2017).

Decision Tree and Random Forest

Decision Tree is a Supervised Learning versatile algorithm that is able to perform both classification and regression tasks. This algorithm's response is based on the criteria defined by each node of the tree. Given a certain input, each node will be responsible for decisions that will lead to the final prediction, present in the tree's leaves. Despite being a powerful and versatile technique, it might encounter over-fitting problems (Géron, 2017; Ray, 2019). In fact, the Random Forest algorithm, also embedded in the same ML paradigm, is one of the solutions used to overcome this issue. Random Forest can be defined as an ensemble of Decision Trees. Various Decision Trees are trained each on a random subset of the training set. This way, the model makes predictions based on various Decision Trees, being more likely to be more flexible with less bias and variance (Géron, 2017; Desai, 2020).

Principal Component Analysis

The Principal Component Analysis (PCA), is an Unsupervised Learning algorithm used for extracting the most relevant features in a dataset. A dataset might have a large number of features, what PCA does is to find the ones who preserve the maximum amount of variance. In fact, it allows the to turn an intractable problem into a tractable one, without losing much information (Géron, 2017; Desai, 2020).

K-means

The K-means is an Unsupervised Learning algorithm, which purpose is to form clusters - groups of data that were formed because of the similarity between its data points - to discover underlying patterns in the data. After defining the number of clusters that are going to be created, this algorithm proceeds to optimize the position of the centroids (location that represents the center of the cluster), so that it successfully creates the intended groups of data (Desai, 2020; Garbade, 2018).

Artifical Neural Networks

Artificial Neural Networks (ANNs) are the core of Deep Learning (DL). Compared to other algorithms, the ANNs offer new possibilities to deal with highly demanding ML tasks as they are very powerful, versatile and scalable (Géron, 2017). Shortly, ANNs can be defined as a directed graph with neurons instead of nodes and links instead of edges (Shalev-Shwartz and Ben-David, 2014). The neurons are responsible for performing some sort of calculation and the result of this calculation will be multiplied by a weight as it travels through the network. These components are present in the layers of a network including the hidden layers, where the calculation happens, the input layer - which contains the data provided to the ANNs - and also the output layer - responsible for producing the outputs (Josh, 2015). There are various types of ANNs, including the Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) or LSTM, for example. Each one has its own advantages and utility in different use cases.

2.5.3 *Learning Algorithms for Time Series Problems*

As stated before, there are learning algorithms that are more suitable to certain types of problems. Given the fact that in this dissertation one of the goals is to develop ML algorithms capable of predicting the future based on past data - Time Series Problems - some of the main aspects of these algorithms will be described.

To begin with, Time Series problems exist in different contexts. Whether the goal is to predict future stock prices (Géron, 2017), future knowledge of pore-water pressure (Wei et al., 2020) or to forecast the number of trips in special events at companies like Uber (Laptev et al., 2017), they all have one aspect in common: their learning algorithms are capable of working with sequences with varied lengths as inputs and therefore capable of processing sequential data with good results. One of the most used algorithms for this purpose is RNNs. Its unique properties, which are going to be explored more deeply throughout this section, make it suitable to tackle these problems (Géron, 2017; Goodfellow et al., 2016).

One aspect these neural networks have in common, is that they both use the same training algorithm to update their weights: Backpropagation Through Time (BPTT). In order to understand how this strategy works, we must review how the standard training algorithm works. This is simply called *backpropagation training algorithm* and it is used in regular ANNs. The goal of this strategy is to modify the weights of an ANN in order to reduce the network's output error. The way this happens is by finding out how much each connection contributes for the error and then tweaking the connection's weights accordingly, along with each bias term. Basically, for each input the network receives, it can be divided it four phases, respectively: forward pass, computation of the network's error, backward pass and finally the gradient descent step. In the first phase, the algorithm computes the output and preserves all intermediate results so they can be used on the backward pass. Afterwards, through a loss function that compares the network's output with the desired output, the output error is calculated. Then, in the backward pass, the gradients are calculated by measuring how much each output connection contributed to the error and it finishes off by computing how much of these error's contributions came from the layer below, doing this repeatedly until it reaches the input layer. Finally, the learning happens by performing a gradient descent step in order to tweak all the connections (Géron, 2017; Brownlee, 2020b). Note that some authors defend that the backpropagation refers only to the calculation of the gradient and not to the previously explained process (Goodfellow et al., 2016).

BPTT is the backpropagation training algorithm explained above applied to an RNN. Compared to the traditional backpropagation, the differences lies with the data that the algorithm works with. The forward pass phase goes through an unrolled network (see Figure 14). Since in RNNs each recurrent neuron additionally receives as an input its own output from the previous time step, the capacity of unrolling the network though time is possible. This means the output of the network takes into account data from previous time steps. Besides this, the weights' update is slightly different since each recurrent neuron has a sets of weights both for inputs and for outputs of the previous time step (Géron, 2017; Brownlee, 2020b; Werbos, 1990).

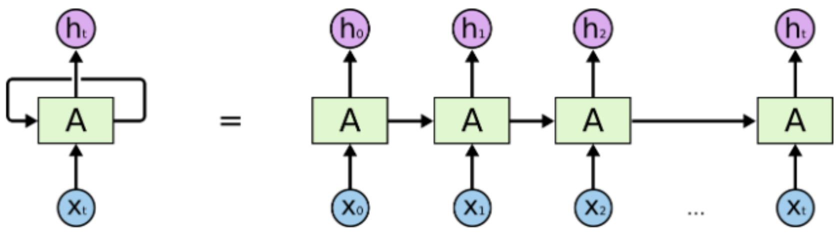


Figure 14: An unrolled Recurrent Neural Network in Olah (2015).

Long Short-Term Memory

RNN is a great DL system for time series problems, because of the features of its recurrent neurons which allows the existence of some form of memory. Despite these benefits, they face issues when dealing with very long sequences, losing information which can be very relevant for predicting future values.

LSTM networks are a particular type of a RNN, being capable of learning long-term dependencies. Therefore, LSTMs are often used to solve sequence derived problems. Having been first introduced in 1997 (Hochreiter and Schmidhuber, 1997), it has been gradually improved over the years (Géron, 2017). Nowadays, it is successfully used in several applications, like speech recognition (Zhang et al., 2016) or trajectory prediction (Altché and d. L. Fortelle, 2018).

In terms of architecture, the benefits of using LSTMs are provided by the nature of a LSTM cell, shown in Figure 15.

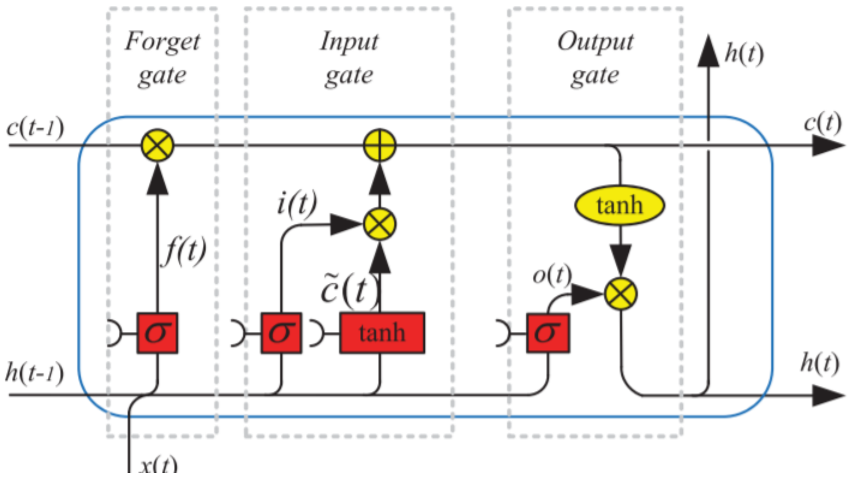


Figure 15: Long Short-Term Memory Cell in Yu et al. (2019).

Compared to a standard recurrent cell, these cells have three different gates which allow the presence of a state within a cell. These gates regulate the information that is added or removed to the cell state and are composed by a sigmoid neural net layer and a element-wise multiplication operation. In the end, the output produced by this cell takes into consideration relevant past information. Additionally, in the process of outputting a result it also decides which information is stored or not in the state. Concluding, since this cell has the ability to store and erase data, and pick relevant information for a given time step, the rate of success at dealing with long term sequences increases (Géron, 2017; Yu et al., 2019; Olah, 2015).

2.6 CONCLUSIONS

After the analysis of several articles, it was possible to identify that crowd sensing can be done in numerous ways and for different purposes, as illustrated in Table 2. In general, from the reviewed the articles, even though some of them belonged to different fields of study, the main approach was to capture either Wi-Fi or Bluetooth probe requests from smart devices as a way to sense the crowd. With the extracted data, the articles showed that is possible to implement crowd sensing for different purposes: crowd density estimation, traffic forecasting or analysing people's mobility are just a few examples.

Table 2: Crowd Sensing Approaches.

	Device	**Field of Study**	**Techonologies**	**RSSI**	**#rawdataset(∼)**	**Models**	**Privacy**
Sensing through Bluetooth	Nokia900	Music Festival	Bluetooth	No	1 million	Graphs IRM	MAC encryption
Sensing through Bluetooth and Wi-Fi	Laptop	Airport	Wi-Fi Bluetooth	Yes	11 million	No information available	No
Sensing in a SmartCity context	SingleBoardComputer	People's mobility in a discotheque Traffic forecasting	Wi-Fi Bluetooth	No	No information available	SOM ETS ARIMA Theta IRM	MAC encryption
Sensing in a MCS paradigm	SmartPhones	Public viewing event	Bluetooth	Yes	4100	Tree Classifier	No

When it comes to comparing Bluetooth based approaches to Wi-Fi one's, it was clear that capturing Bluetooth probe requests presented a higher number of challenges. Firstly, even though either Bluetooth or Wi-Fi need to be manually turned on by the user for enabling the retrieval of data by the sensor, Bluetooth has to be additionally set to discoverable mode as well. Moreover, the detection's range of Wi-Fi signals is much higher than the Bluetooth ones. While the first has a communication range that can vary from 35 meters to 100 meters (depending on the environment), the second has a range of about 10 meters. Finally, in one of the articles reviewed, where their approach was tested on a big airport, the authors in Schauer et al. (2014) revealed that they detected 6211 unique Wi-Fi MAC addresses vs 250 unique Bluetooth addresses per day in the public area, which verifies that fact that Bluetooth probe requests are harder to capture, and therefore using Wi-Fi is a more efficient approach in these cases.

As expected, in order to do any kind of sensing it is necessary to have a sensing device. In the reviewed approaches, different kind of devices were used: Nokia N900 smartphones, laptops, single-board computers or android smartphones in general. It was possible to identify that even though these devices have different capabilities, the main one is being able to capture probe requests. Therefore, the conclusion for this dissertation is that the sensing device used should be capable of doing so. Other aspects are important too, namely the price of the device or for the capacity to connect itself to other networks.

In all the captures done by these devices the structure of data extracted has a typical set of values: Bluetooth / Wi-Fi MAC address, timestamp, scanner id (in case multiple scanners are used) and RSSI. From these fields, the RSSI is the more controversial one. Since it is a very volatile value, some approaches don't use it at all. The ones who do make sure not to take any conclusions that rely solely on this metric. Defining a way to filter its values (with a defined threshold), combining it with other metrics, or creating features that despite being relevant, are not directly dependent on the RSSI, are some of the methods use to mitigate the volatileness of this metric.

Regarding the MAC address, some of the crowd sensing approaches decided to hash this address right after extracting it, in order to secure one's privacy. Crowd sensing provides solutions for modern-day problems, but at the same time, it handles data that, if not manipulated correctly, might bring privacy issues. Therefore, either encrypting the MAC address or being careful to only show data that represents a big group of people are some of the measures that help to protect one's privacy. A big subject arises from this topic: MAC randomization - a solution that has been recently implemented in this field. Some studies have revealed that either there are ways to get around it, or that many devices do not implement it at all. When wanting to do some sort of crowd sensing, this mechanism should not be ignored, since despite protecting one's privacy, it might produce data that leads to wrong conclusions. If a certain device is detected by sensors with the MAC address X, and 1 minute later it changes its address, then, the sensors would produce data indicating that apparently two different devices were detected within 1 minute, which is not true. This is not a common case to happen, since MAC randomization does have flaws, and when occurring efficiently it occurs when the smart device is sending probes to search for available networks.

Another important point from the examined methods is that some of them have ground truth information, giving them the possibility of testing the output of their approach with data that corresponds to the truth. Finding a point of interest where ground truth information is available to test the approach that will be presented in this dissertation, is a key factor to test its feasibility. Either resorting to other sensing devices, like video cameras, or to places where to get in there's needed some sort of check-in method like an airport, were some of the methods used by the approaches reviewed.

In addition, the user's role in the sensing phase must always be recognized. Having an active role, like in opportunistic or participatory sensing, brings challenges that are difficult to overcome.

After the sensing phase, each studied work showed how they operated with the gathered data in order to make it useful and beneficial. In terms of data modeling, the studies revealed that creating new variables with the existent data, a process know as feature engineering, can be beneficial not only for visualization purposes, but also for the fueling of prediction algorithms, making them have increased robustness and accuracy. The use of different algorithms proved to be useful for various purposes: traffic forecasting, pedestrian flow estimation or even for finding relations between each user and the groups they were moving in.

3 EXPERIMENTS

This section will explain, in detail, the various development phases of this project. It will review all the phases from Data Collection to Model Conception and Tuning, explaining the methods used for data collection and how this data provided several insights through its analysis, and, finally, how the modeling was organized to produce good performing models. Moreover, the capabilities of the device used in the Data Collection are tested in the RSSI Experiment.

3.1 ETHICAL CONSIDERATIONS

Before moving to the actual development, it is important to emphasize that all the work done does not compromise any individual's privacy. All the data are collected through an entirely non-invasive methodology and all the work done upon this data is not centered on any individual, but instead on gathering information beneficial for crowd density problems.

3.2 DATA COLLECTION

The first step was to find a proper device for the extraction of data. Since this work is being done in the context of Smart Cities, the device was chosen taking into account the price, the capacity of detecting Wi-Fi probe requests and the capacity of uploading the sensed data to a real time database.

Taking into account these parameters, the chosen device was the low-cost Wi-Fi microchip ESP8266 ESP-12E NodeMCU Amica board (Figure 16). The NodeMCU is a low-power Arduino type board which runs on ESP8266 Wi-Fi module. In addition to the reduced cost of €2.4 per board, it also has a Wi-Fi module, 4MB flash memory, a built-in antenna, open-source support, a micro-USB interface and finally small dimensions (4.8x2.4x0.5cm) and low weight (109g). Such features allow the board to be a sensor suitable for crowd sensing (Fernandes et al., 2018), being here entitled as a smart scanner.

Figure 16: ESP8266 ESP-12E NodeMCU Amica board.

In order to program the board, the application that used was the Arduino IDE. It provides simple one-click mechanisms to compile and upload a program to the smart scanner. This program, called a sketch in this IDE, can be written in the language C or C++. It needs to have implemented at least two basic functions: *setup()* and *loop()*. The first one is a function that is called when the sketch starts after power-up or reset of the board, being normally used for initializing variables, setting up connections or libraries needed in the developed sketch. The second is a function repeatedly executed until the board is powered of or reseted.

The Arduino project's open-source nature has allowed the development of many free software libraries that developers are using to extend their projects. This allowed the use of two external libraries in the development of the code: ESP8266Wifi (Grokhotkov et al., 2020a; Grokhotkov, 2020; Grokhotkov et al., 2020b) and Firebase Arduino (Coyne et al., 2016). Its usage enabled the exploration of the board's Wi-Fi module, allowing the connection to a network and also the detection of Wi-Fi probe requests and, additionally, the possibility of uploading the data into the Firebase Realtime Database (Google, 2020a,b), which was the database chosen to store the sensed data.

The developed sketch - which was based on the software available in Fernandes (2018) - was coded in a way that the ESP8266 ESP-12E NodeMCU Amica board captured Wi-Fi probe requests emitted by people's smart-devices in a non-invasive manner. The *setup()* function, executed once the board is initialized or reseted, is used for setting up the board as an Access Point (AP), connecting it to a network, and if connected successfully, establishing a connection to Firebase. Furthermore, a timer is also initialized for managing the transmission of data to the Firebase database. Finally, four different handlers are registered. These handlers are called once a station connects or disconnects to the board and when a Wi-Fi probe request is captured by the board. In fact, the most important function, *onProbeRequestCaptureData()*, is the one being called upon the reception of a Wi-Fi probe request (Listing 1). If the data structure used for storing the data temporarily has space, the new probe request will be stored in case of it being considered a new sighting. After that, every *sendtimer* seconds all the data is pushed into the Firebase database and cleaned locally. Apart from this, the *loop* function is also enabling the user to send commands through the serial like *Stop/Restart*, *Count*, *Start/Stop_Timer* and *Clear*, among others.

Listing 1: Function called upon the reception of a Wi-Fi probe request.

```
void onProbeRequestCaptureData(const WiFiEventSoftAPModeProbeRequestReceived& evt) {
  if(currIndex < ARRAY_SIZE){
    if(newSighting(evt)){
      probeArray[currIndex].mac = macToString(evt.mac);
      probeArray[currIndex].rssi = evt.rssi;
      probeArray[currIndex++].previousMillisDetected = millis();
    }
  } else{
    Serial.println(F("*** Array Limit Achieved!! Send and clear it to process more
        probe requests! ***"));
  }
}
```

3.3 DATA ANALYSIS

In this section, the dataset acquired in the previous phase will be analysed in the following paragraphs by first understanding and cleaning the data, and then explored in order to gain insights on several nuances that the data collected can offer.

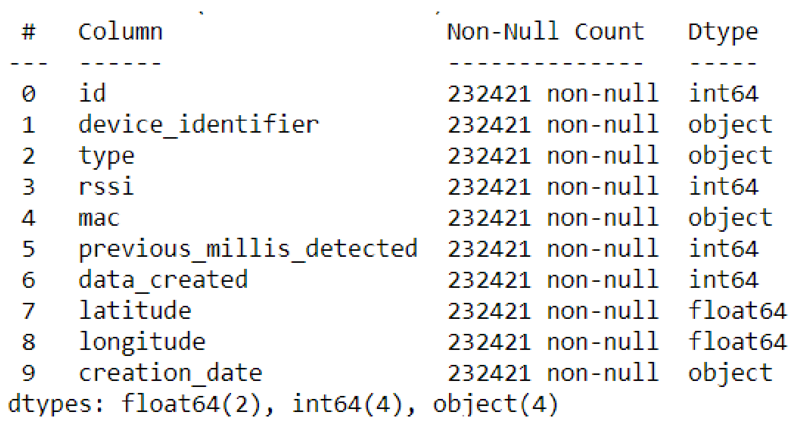
3.3.1 *Materials*

For these experiments, the development was done in the Jupyter Notebook using Anaconda, a data science open-source platform. The Jupyter Notebook is a free open-source web application that allows the creation of documents that contain code, and it can be used for several purposes like: data cleaning and transformation, data visualization, ML and more. The Jupyter Notebook runs on a local runtime, meaning that the processing happens on a single local machine. The machine used had the following specifications: Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz (4 CPUs), 8GB RAM and a integrated graphics card, Intel(R) HD Graphics 5500.

Since the tasks in these experiments were data science oriented, the chosen programming language was Python due to its properties and useful libraries to fulfil the necessary purposes. Libraries like *Pandas*, *Numpy* or *Matplotlib* were used since they offer a range of different tools for data processing and visualization. Moreover, a MAC address lookup API (Massoud, 2020) was used to find the vendor of each MAC address.

3.3.2 *Data Understanding*

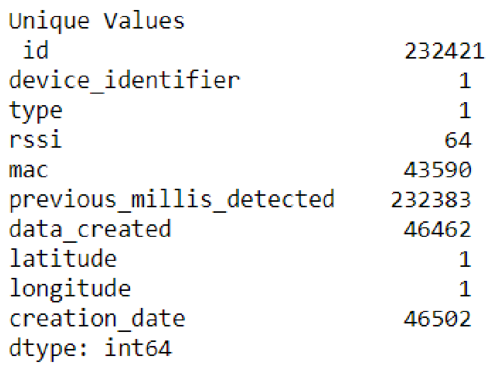
The data acquired by the smart scanner was converted to a CSV file and was loaded using the *Pandas* library. The first step was to understand the data by quickly looking at its structure, attributes and data quality. The referred dataset has **232421 entries** - each one corresponds to a probe request sent to the Firebase by the smart scanner - being characterized by **10 different attributes**: *id, device_identifier, type, rsi, mac, previous_millis_detected, data_created, latitude, longitude* and *creation_date*. As demonstrated in Figure 17, these attributes hold three different data types: *float*, *int* and *object*.



```
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   id                        232421 non-null  int64
 1   device_identifier         232421 non-null  object
 2   type                      232421 non-null  object
 3   rssi                      232421 non-null  int64
 4   mac                       232421 non-null  object
 5   previous_millis_detected  232421 non-null  int64
 6   data_created              232421 non-null  int64
 7   latitude                  232421 non-null  float64
 8   longitude                 232421 non-null  float64
 9   creation_date             232421 non-null  object
dtypes: float64(2), int64(4), object(4)
```

Figure 17: Information about the 10 attributes of the dataset.

The type *object* can hold any kind of Python object, but since the data was loaded from a CSV file, it is possible to conclude that these objects hold a text attribute. In Figure 18, the number of unique values for each attributed is displayed.



```
Unique Values
 id                         232421
device_identifier                1
type                             1
rssi                            64
mac                          43590
previous_millis_detected    232383
data_created                 46462
latitude                         1
longitude                        1
creation_date                46502
dtype: int64
```

Figure 18: Unique values in each attribute.

As expected, the categorical attributes have fewer unique values so it is important to understand which values these attributes hold. Having mentioned that, each attribute is described in Table 3.

Table 3: Attributes information.

(a) Numerical Attributes.

Parameter Name	Description	Type
id	The unique identifier for each entry of the dataset	int
rssi	Corresponds to the RSSI Indicator of the probe request	int
previous_millis_detected	Indicates how many milliseconds passed since the smart scanner began capturing	int
data_created	The data identifier for which the probe request was captured	int
latitude	The latitude of the sensor's localization There are only two unique values for this attribute, meaning that sensor was capturing in two different places	float
longitude	The longitude of the sensor's localization There are only two unique values for this attribute, meaning that sensor was capturing in two different places	float

(b) Categorical Attributes.

Parameter Name	Description	Type
device_identifier	The sensor's identifier	object
type	Indicates the probe type.	object
mac	Indicates the MAC address of the device associated to the probe request There are 43590 unique values for this attribute, meaning that there are **43590 unique MAC addresses**	object
creation_date	The data in which the probe request was captured following the format *Year-Month-Day Hour:Minute:Second:Millisecond* The analysis of this attribute showed that the dataset contains values between 2018-07-25 and 2018-09-19	object

Lastly, it is important to mention that the dataset is ordered by date.

3.3.3 *Data Quality*

In many cases the data may present issues like duplicated or missing values (for example), leading to false conclusions and prejudicing all the posterior phases. Therefore, the dataset was also analysed in that perspective. In fact, the dataset has no missing values and also doesn't contain any duplicate value, resulting in no further exploration for these cases.

3.3.4 *Data Exploration*

The data structure and quality were covered, but no real information was extracted yet. Therefore, this section will get as many insights as possible given by the dataset: by exploring the attributes, their combinations, and creating new ones. This will be all done bearing in mind that one of the objectives of this dissertation is to gather viable information on crowd density data and that the crowd sensor was located in *Universidade do Minho*'s IT department, more specifically, in the Intelligent Systems Lab. The analysis itself was divided in three different parts: **Time Analysis**, **MAC Addresses Analysis** and **Crowd Density Analysis**.

Time Analysis

The current dataset has an attribute, *creation_date*, which can be used to determine correlations between the frequency of captured probe requests and time. Consequently, exploring this attribute can offer interesting insights.

To begin the time analysis, a simple histogram was created, illustrated in Figure 19. This plot allowed to get a general perception on how the number instances - captured Wi-Fi probe requests - varied in time, per day.

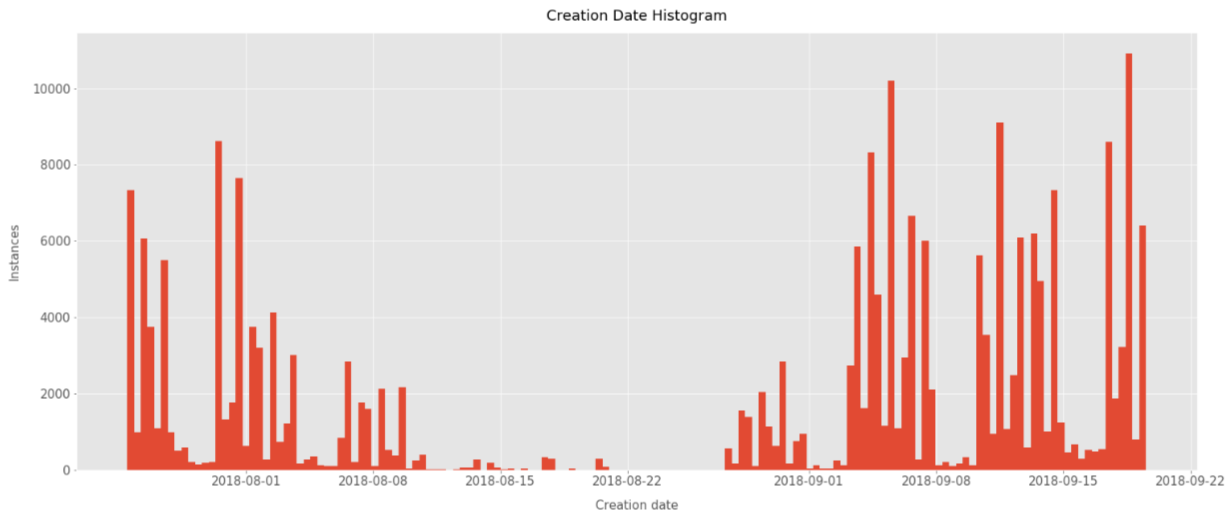


Figure 19: Creation Date Histogram, per day.

By analysing this plot, it is trivial to conclude the following points:

- The numbers of instances varies with time;
- The registered instances go from July to September;
- The month in which the instance is registered affects the number of instances;
- Weekends and weekdays have different numbers of instances.

The number of instances variation can be explained by different factors. In the case of the second half of August there are not many captured devices, which can be explained by this period being a holiday season. Also, weekends seem to have less captured devices when compared to regular weekdays, which is sustained by the fact the data collection site has less people on the weekend. A deeper study was carried out on these specific variations to confirm its veracity.

Nine new features were added to the dataset: *Year*, *Month*, *DayName*, *DayWeek*, *DayMonth*, *Time*, *Hour*, *Minute* and *Weekend* - all extracted from the *creation_date*. The names of the attributes are self-explanatory. For example, *DayWeek* corresponds to the number of the day in the week, *DayMonth* corresponds to the number of the day in the month, and *Weekend* indicates if it is weekend or not. To complement the histogram which showed a general perception of how the number of instances varied, more specific cases were studied, with the aid of the new features.

As Figure 20 shows, three different bar plots were created for the following attributes: *Month*, *DayMonth* and *DayWeek*.

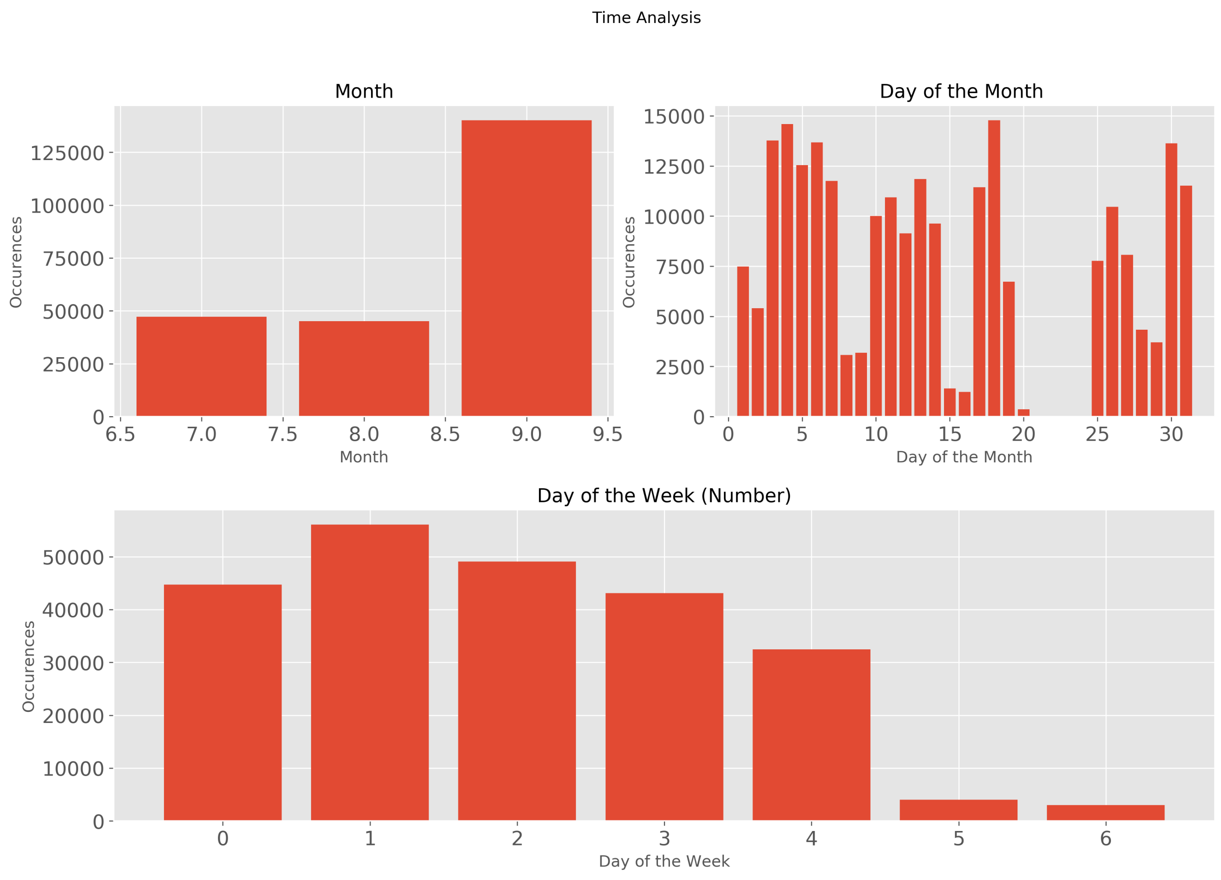


Figure 20: Analysis to months, day of the month and day of the week.

The first plot, shows that the majority of the instances were registered in September. The lack of instances in August was already explained, but in the case of July the low number of instances registered is simply justified by the fact that the first instance was only registered at the end of July (25-07-2018). In the second case, relative to the *day of the month* plot, there is clearly a gap in the period between the 21$^{\text{th}}$ and 24$^{\text{th}}$ day of the month. This is explained by three different reasons. Firstly, as the recorded instances only start in the 25th of July, this month doesn't contribute for number of instances in the studied period. Moreover, the last recorded instance is on the 19th of September, which also puts September in the same case of July. Finally, only August remains - a month in which not many instances were recorded, specially in the second half of the month. In the third plot, the assumption (retrieved from Figure 19) that weekends seemed to have less recorded instances is verified.

To complete the time analysis and to get all the insights possible through time derived features, bar plots for the attributes *Hours*, *Weekend* and *Minutes* were created.

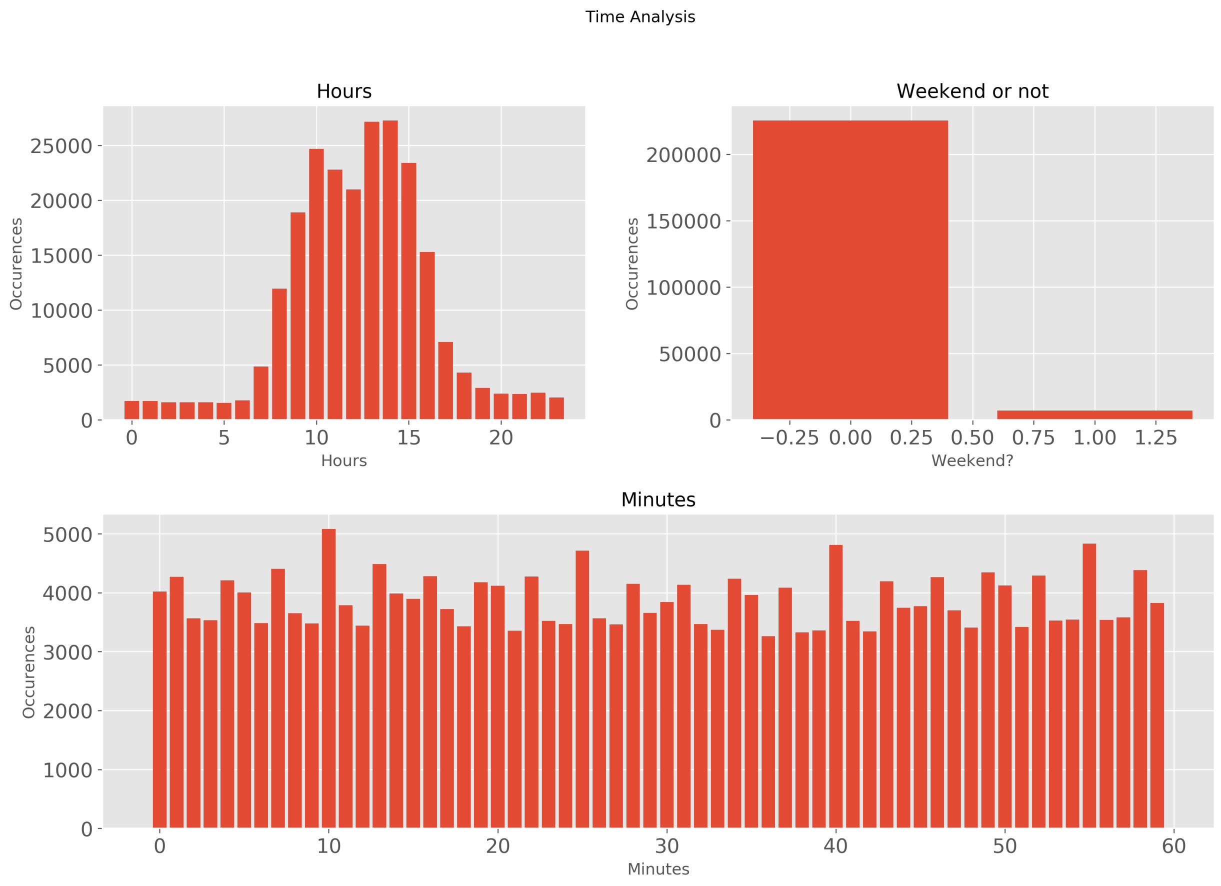


Figure 21: Analysis to hours, weekends and minutes.

As shown in Figure 21, the *weekend* plot and the *minutes* plot don't offer much new information. On contrary, the *hours* plot shows that the time period between 9 and 16 is when there are more devices present and consequently more recorded instances. This is probably due to the fact that it is a working hour period. Being the data acquired from a working place, the density showed in the 9-16 period is justifiable.

To conclude, Table 5 presents the summary of the conclusions explained previously.

Table 5: Time Analysis conclusions.

Feature	Conclusions
Day of the Week	There is a higher number of captured Wi-Fi probe requests during weekdays
Day of the Month	Between the 21th and 24th day of a Month, there's very few recorded instances
Month	September has the most captured Wi-Fi probe requests, whereas July and August have fewer
Hour	Between 9 AM and 16 PM it's the period where the number of captured Wi-Fi probe requests is higher
Minute	No relevant conclusions

MAC Addresses Analysis

The next study will be focused on the MAC addresses attached to the sensed Wi-Fi probe requests. Valuable and different information can be obtained from these addresses. As a matter of fact, it would be interesting to know if the number of the sensed probe requests was more affected by single devices, that continuously kept sending Wi-Fi probe requests and therefore were sensed multiple times, or by a large number of different devices.

In fact, through the number of probe requests per MAC address, it was possible to find that, approximately, five Wi-Fi probe requests were captured per device. Additionally, the respective median was equal to one. These two results mean that a considerable number of devices only gets sensed once. To confirm this, the 75 th percentile was calculated and the result was one, meaning that 75% of the devices are sensed only one time. This result could, in part, be explained by the MAC Randomization mechanism - making the sensor capture probe requests from the same device with different MAC addresses associated - but this topic will be explored further on this analysis.

The first three octets of the address correspond to the Organizationally Unique Identifier (OUI), allowing the vendor identification. This way, the vendors of the captured MAC addresses were studied (Figure 22).

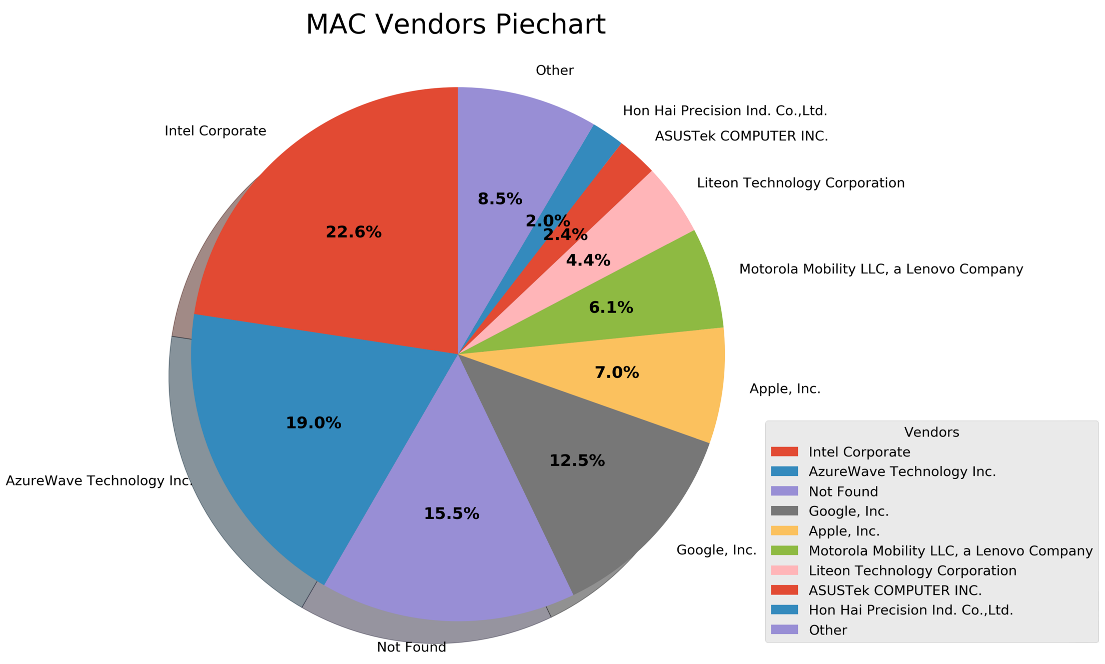


Figure 22: MAC Vendors pie chart.

This chart shows the most represented vendors in the dataset, where *Intel Corporate*, *AzureWave Technology Inc* and *Google, Inc.* are some of the most represented vendors. Besides, the slice in the chart represented by *Other* corresponds to the least represented vendors, while the *Not Found* slice are addresses with an unknown vendor. This can happen if an address is locally administrated or private, for example.

The most frequent address in the captured Wi-Fi requests was also studied to understand the kind of device it was and why it was sensed so many times. The address is associated to *AzureWave Technology Inc*, a company that makes Wi-Fi components for computers and IoT devices - so the device was most likely a computer. Besides that, the first record of the device it's immediately on the first day of sensing and last one is on the last sensing day. Additionally, a time analysis was conducted specifically for this MAC address, as displayed in Figure 23.

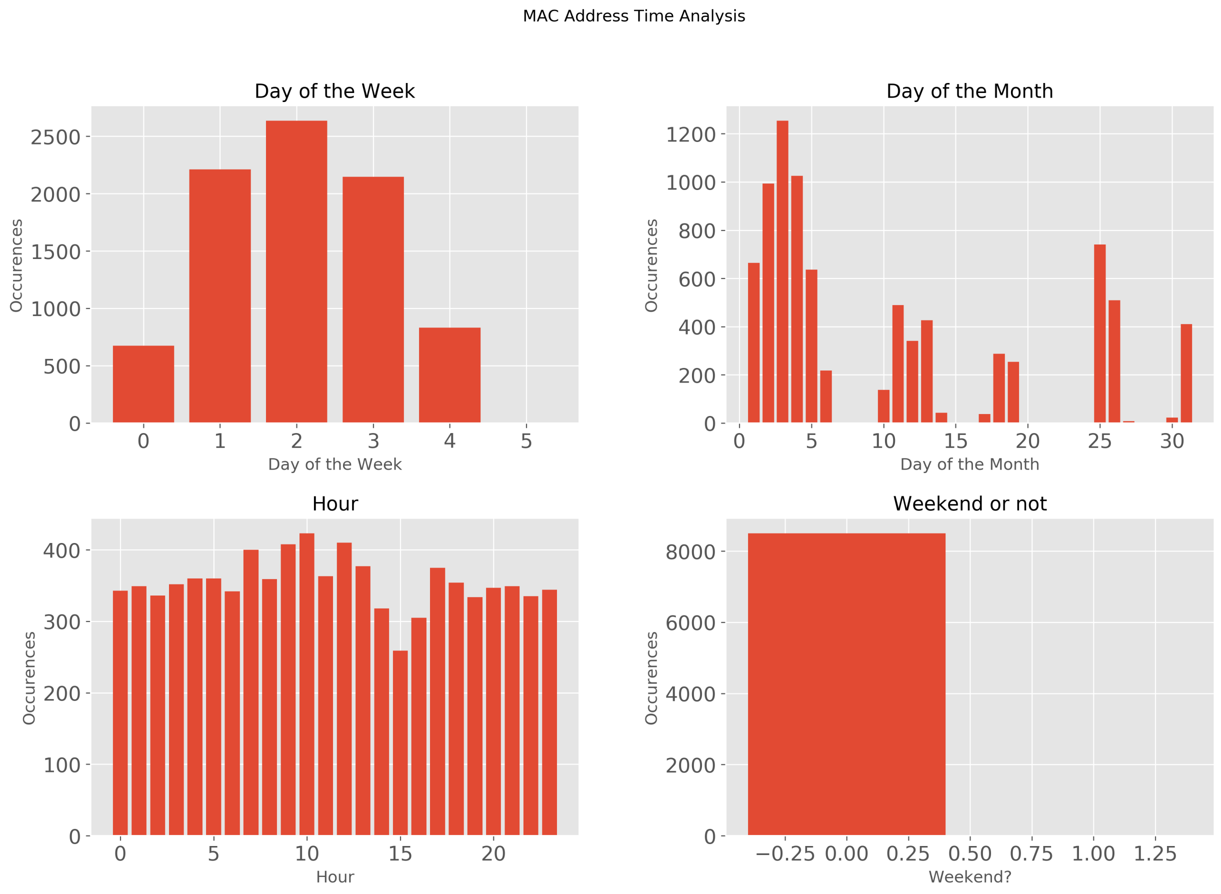
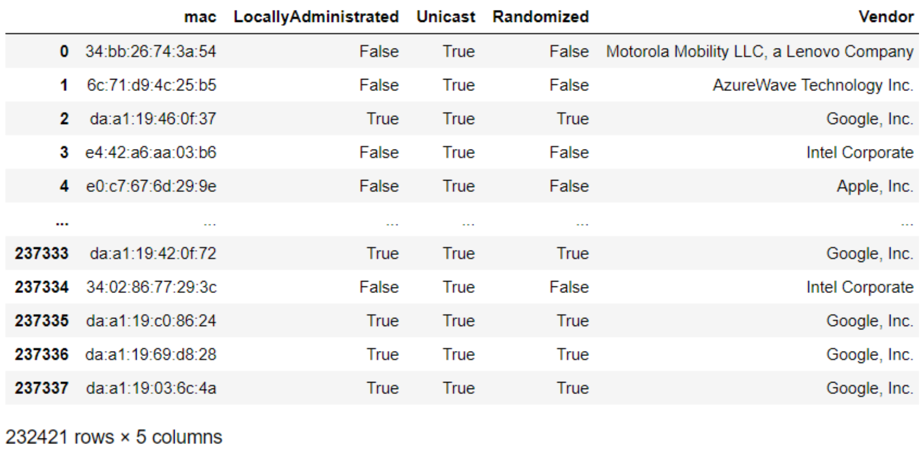


Figure 23: MAC Address time analysis.

This plot didn't add much relevant information. It was expected that perhaps the device was active on the weekends, contributing for so many captured Wi-Fi probes associated with this address. But that didn't turn out to be true, and since there were no relevant fluctuations, no relevant information could be extracted from these plots.

The last analysis on the MAC addresses was focused on MAC randomization. The objective was to find out if it was possible to identify if an address had been randomized, and, if so, calculate how many of them were randomized. Through research it was possible to find that a randomized MAC address is always locally administrated and transmitted in unicast (Vanhoef et al., 2016; Ansley, 2019; Project, 2020). However, there are addresses that have this characteristics and are not randomized (Martin et al., 2017). Moreover, some authors also refer that after the randomization process, the OUI is no longer identifiable (Freudiger, 2015).

The first step was to create a new dataset based on the MAC addresses from the Wi-Fi dataset, partially shown in Figure 24.



	mac	LocallyAdministrated	Unicast	Randomized	Vendor
0	34:bb:26:74:3a:54	False	True	False	Motorola Mobility LLC, a Lenovo Company
1	6c:71:d9:4c:25:b5	False	True	False	AzureWave Technology Inc.
2	da:a1:19:46:0f:37	True	True	True	Google, Inc.
3	e4:42:a6:aa:03:b6	False	True	False	Intel Corporate
4	e0:c7:67:6d:29:9e	False	True	False	Apple, Inc.
...	...	...	...	...	...
237333	da:a1:19:42:0f:72	True	True	True	Google, Inc.
237334	34:02:86:77:29:3c	False	True	False	Intel Corporate
237335	da:a1:19:c0:86:24	True	True	True	Google, Inc.
237336	da:a1:19:69:d8:28	True	True	True	Google, Inc.
237337	da:a1:19:03:6c:4a	True	True	True	Google, Inc.

232421 rows × 5 columns

Figure 24: MAC Addresses dataset.

To each MAC address, four different columns were attached: *LocallyAdministrated*, *Unicast*, *Randomized* and *Vendor*. The first, identifies if the address has been locally administrated, doing so by checking if the address' local assigned bit, which is the second least significant bit of the first octet in a MAC address, is set to 1. In the second place, the *Unicast* column spots if the multicast bit, the address' least significant bit, is set to 0. Thirdly, the *Randomized* column identifies if an address is randomized by checking if the locally administrated bit is set to 1 and the multicast bit is set to 0. Lastly, the Vendor is also on the dataset to study if there is any relation between those that don't have any Vendor associated and the randomization mechanism. This process is partially shown in Listing 2.

Listing 2: Creation of the MAC dataset.

```
d = {
            "mac": sensing_data_WiFi.mac,
            "LocallyAdministrated": mac_check_local(sensing_data_WiFi.mac),
            "Unicast":mac_check_unicast (sensing_data_WiFi.mac),
            "Randomized": mac_check_randomized (sensing_data_WiFi.mac),
            "Vendor":l_vendors
    }
df_mac = pd.DataFrame(data=d)
```

The analysis of the MAC Addresses dataset started by studying main corpus statistics, as shown in Table 6. Firstly, it was possible to conclude that from 232421 captured addresses, 52626 were locally administrated. Moreover, the second conclusion taken was that all addresses were transmitted in unicast. Therefore, bearing this in mind and since a randomized address needs to be locally administrated and transmitted in unicast, all locally administrated addresses in the dataset are candidates to being randomized addresses. Regarding the vendors, from the whole dataset, 51576 addresses didn't have any vendor associated.

Table 6: Corpus statistics.

Category	#MACs
Corpus	232421
Locally Administrated	52626
Unicast	232421
Randomized	52626
Not Found	51567

At this point, it was possible to identify how many MAC addresses were candidate to be randomized - 52626 addresses. As referred previously, some authors mention that randomized addresses do not have any vendor associated. Therefore, the vendor of each one of the candidates to being randomized MAC addresses was analysed, as demonstrated in Table 7.

Table 7: Locally Administrated Statistics.

Category	#MACs
Locally Administrated	52626
Found	29022
Not Found	23604

These results indicated that 23604 MAC addresses were both locally administrated and didn't have any vendor associated. Since the studies from other authors revealed that a randomized MAC address is always locally administrated and have unicast transmission, and also that randomized addresses don't have any vendor associated, a possible conclusion is that 23604 MAC addresses are randomized - **10 % of the total corpus**.

Crowd Density Analysis

Data analysis was finished by performing a study focused on getting crowd insights through the collected data. For this whole analysis it is important to clarify the difference between the number of unique and total MAC addresses in a specific time frame. In comparison to unique requests, total requests involve repeated MAC addresses, meaning that one person carrying one device can contribute with multiple requests. Therefore, when wanting to understand crowd density, this is not a viable metric, whereas the unique number of addresses can be a better alternative. It is also not completely accurate since devices using the MAC randomization mechanism will be registered with different addresses and because there might exist more than one device per person. Nonetheless, MAC randomization is not being correctly used nowadays, and as shown in previous results, only 10 % of the total addresses were randomized. In conclusion, analysing the number of unique MAC addresses will provide viable crowd insights.

Given this context, the first analysis was based on understanding the difference between the number of unique and total MAC addresses per day. To do so, the plot in Figure 25 was created - where the blue line represents the variation of number of total

MAC addresses per day, whereas the red one is relative to the variation of number of unique MAC addresses.

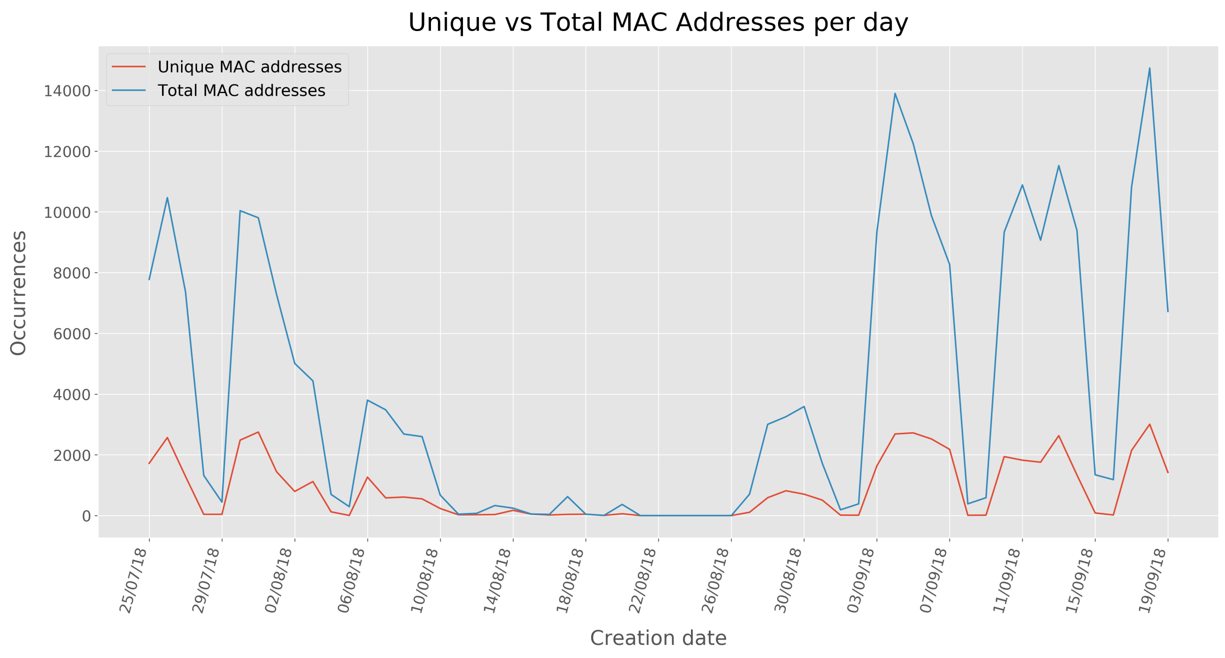


Figure 25: Unique vs Total MAC Addresses per day.

The first detail that stands out is that both line plots have the same fluctuations. This is valuable because all the insights obtained in the time analysis done previously (which was relative to all requests), also apply to unique addresses and therefore to the crowd itself - an hypothesis also studied and confirmed in Figure 26. We can conclude, for example, that in the data collection's site there is a higher crowd density at regular week days. Other than that, despite following the same variations, it is clear that the number of unique of MAC addresses is remarkably lower, probably due to single devices being sensed multiple times - supporting the argument that the randomization mechanism is not widely used.

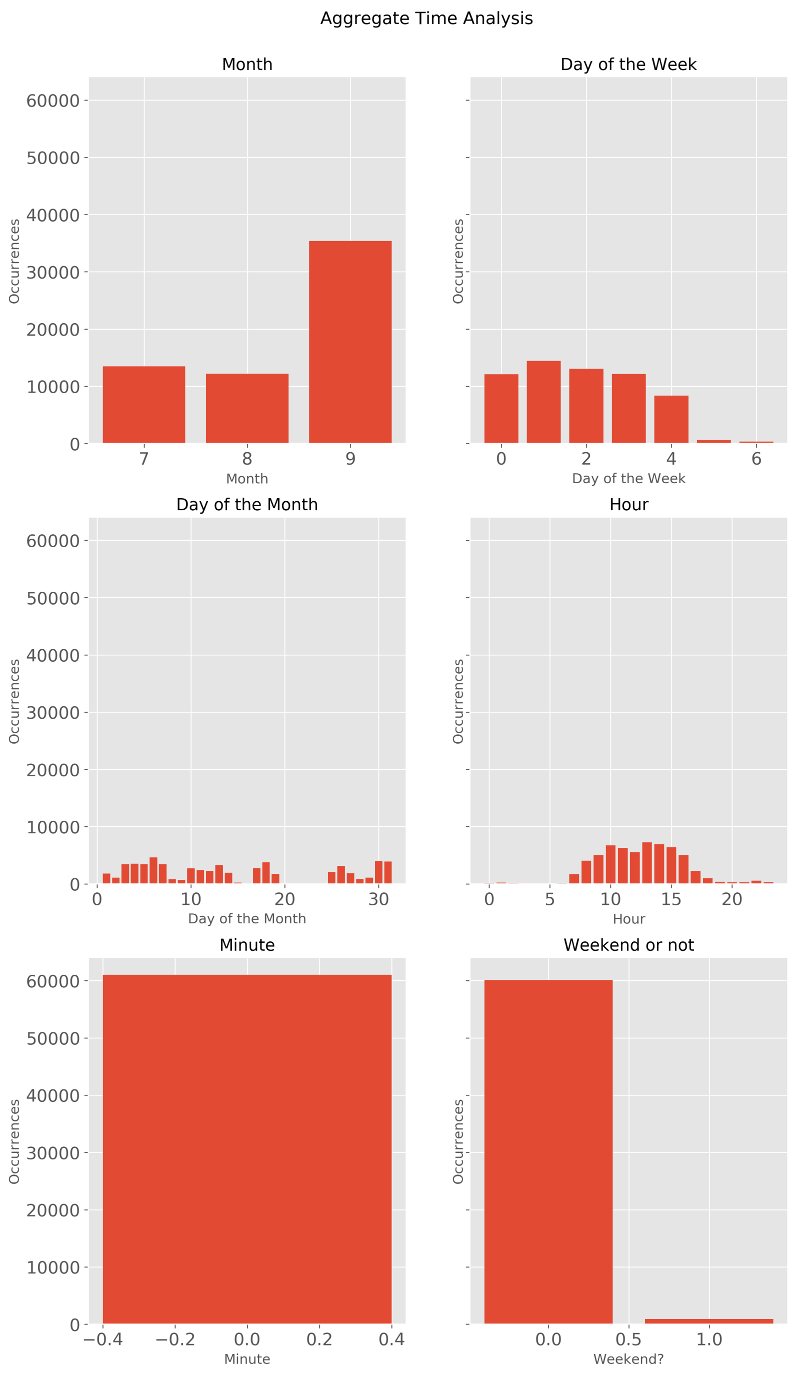


Figure 26: Aggregate Time Analysis.

It would also be valuable to know the number of persons in a given time frame. This way, Figure 27 shows the different number of *Not Unique*, *Unique* and *All* MAC addresses in a given period of time.

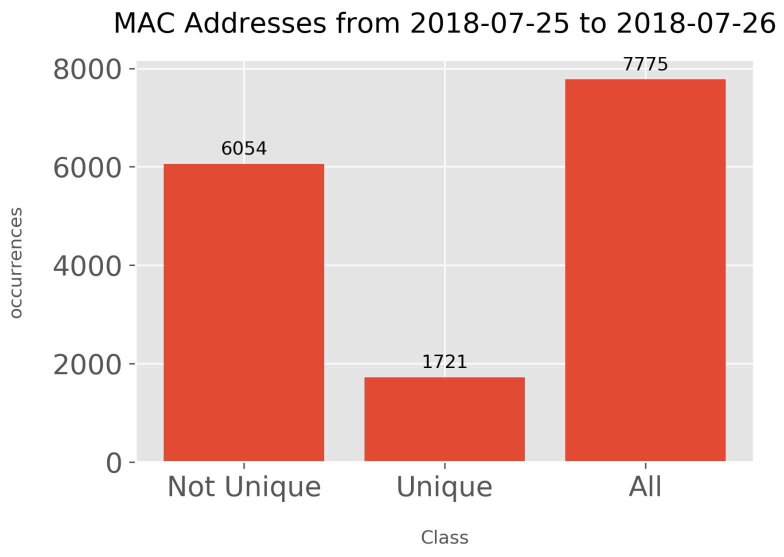


Figure 27: Unique MAC Addresses Plot.

For example, in this case, on the 7th of July of 2018, there were about 1721 devices present in the data collection site and therefore a similar number of persons.

Moreover, studying time periods in a timeframe is also useful for more specific insights. For example, as shown in Figure 28, the 11am to 12am hour period period is divided in groups of 15 minutes that compare the number of *Unique* an *Not Unique* MAC addresses. The red bar plots show that there isn't much relevant variations, meaning that the number of persons stayed approximately the same.

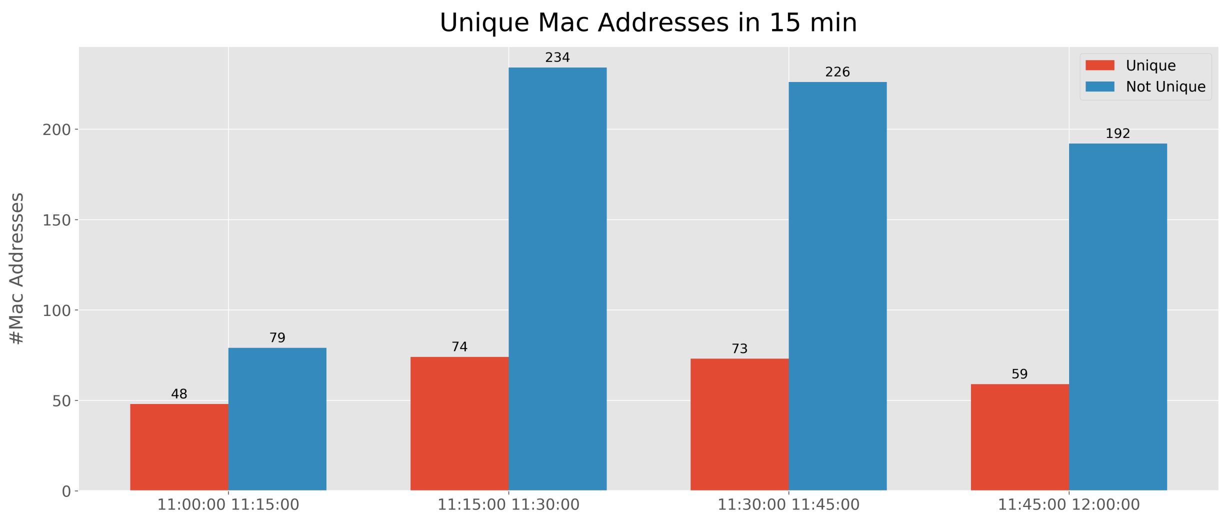


Figure 28: Variation of MAC Addresses in 2018-07-25.

Another case studied was to find how the number of persons varied from 14pm to 20pm - a period that includes some of the busiest hours - on weekdays. To do so, two different days - a monday and a tuesday - were studied, as illustrated in Figure 29 and Figure 30, respectively.

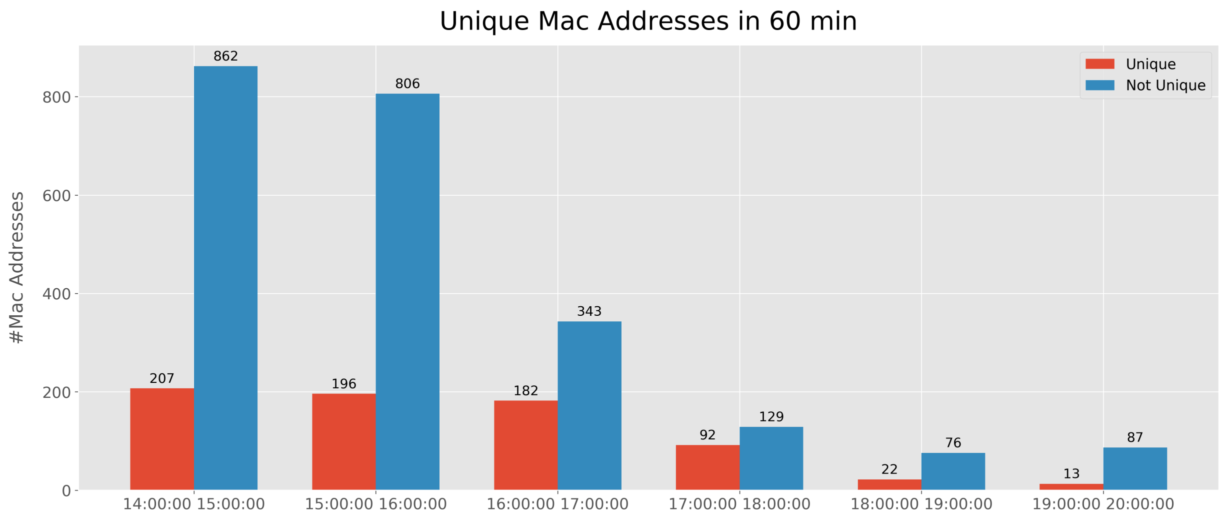


Figure 29: Variation of MAC Addresses in 2018-09-10.

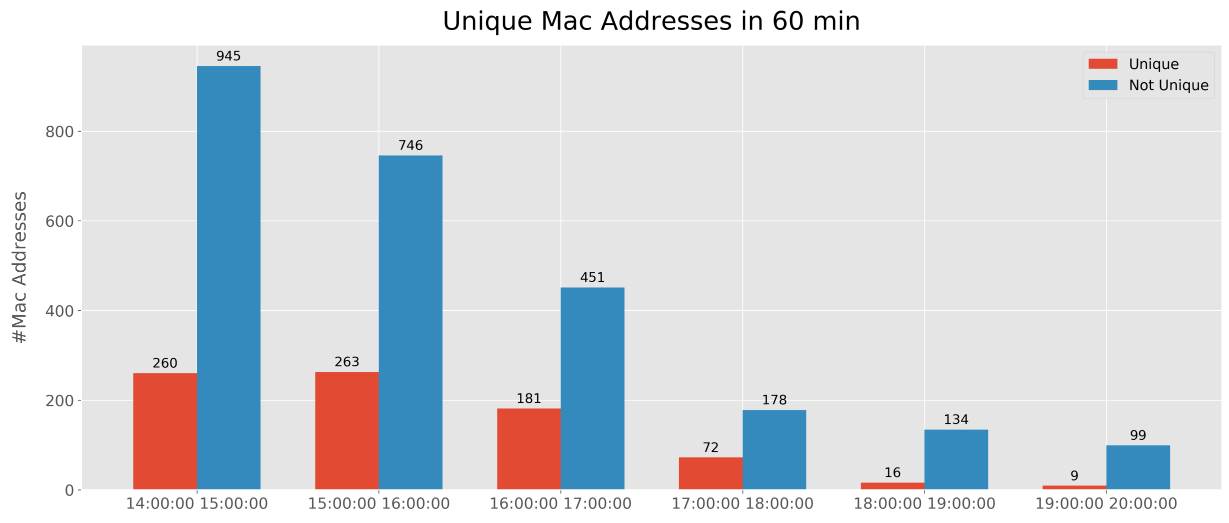


Figure 30: Variation of MAC Addresses in 2018-09-11.

Both plots present the variation, in groups of 60 minutes, of the number of *Unique* and *Not Unique* Mac addresses, from 14pm to 20pm. Both plots are very similar, probably because they are both during weekdays, and also show that the number of *Unique* devices starts decreasing around 5pm, meaning that the number of persons also decreases at this time.

3.4 RECEIVED SIGNAL STRENGTH INDICATOR EXPERIMENT

A RSSI experiment was conducted to test the device's signal detection range and what can interfere with it or not. This experiment was done in an indoor scenario, that included walls, doors and floors, and was divided in two parts. In the first one, the objective was to find how much of an indoor site the smart scanner could cover, while also testing the interference of obstacles. The procedure started by setting up the ESP8266 board as an AP in a specific site. Then, a person with a mobile device would walk away from the device, try to connect that device to the smart scanner, go back the initial site, and check if the connection had been successful by analysing the monitor in Arduino. This process was repeated multiple times, gradually increasing the distance device-smart scanner, until the connection was no longer possible, or increasing the number of obstacles. By doing this, the area that the smart scanner could cover in an indoor scenario was discovered, as well as the effect of obstacles in the smart scanner's signal transmission.

When it comes to obstacles, the signal strength was clearly weakened by it. Nevertheless, it was clear that obstacles like walls had higher interference than doors, for example. In terms of range, from the site where the smart scanner was positioned the signal existed until around 35 metres to the right and 25 metres to the left. Finally, being on the floor below where the smart scanner was, made the signal very weak, but approximately right below the smart scanner and in a range of around 10 meters, the connection with the smart scanner was still possible.

In the second part of the experiment, the objective was to simply analyse the maximum RSSI value that the smart scanner's signal could reach. Note that this wasn't possible in the previous procedure because when a device establishes a connection to the smart scanner, the event registered - *WiFiEventSoftAPModeStationConnected* - doesn't have the RSSI avalaible for retrieval. Therefore, the procedure was to run a sketch where the smart scanner would capture the Wi-Fi probe requests of defined MAC address. This address was from a mobile device. Having the indoor points where the device could not connect to the smart scanner, which were discovered in the previous experiment, the mobile device was moved to those points while having the Wi-Fi turned on. By doing this, the maximum RSSI values were captured by the smart scanner. The conclusion taken was that that if the RSSI was around 90, then the signal would start being very intermittent, reaching values of 93 at maximum before the signal fading.

3.5 MODEL CONCEPTION AND TUNING

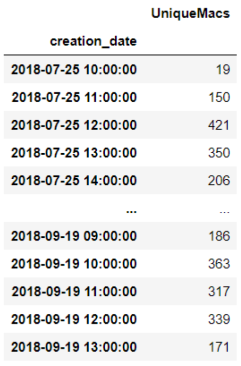
This section will go through all the creation process of ML models for the collected data and all other steps these models imply. The main objective was to create a model capable of accurately predict the number of people in the future based on past data. This is a time series problem. In the following sections, all the steps taken are going to be presented and explained.

3.5.1 *Materials*

The materials used previously were also used for the purposes of model conception and tuning. Additionally, a different platform and various other python libraries were used. When it comes to hyper-parameters optimization, Colaboratory - a platform from Google that allows writing and execution of Python code in the browser - was the tool used for this computationally expensive task. Colaboratory is similiar to the Jupyter Notebook, but since colab notebook run on Google's cloud servers, it uses accelerated hardware. Moreover, ML and DL libraries like *Keras*, *TensorFlow* and *Scikit-Learn*, were used to create and evaluate ML models. The library *Matplotlib* was also used to plot different nuances of the models: predictions and training losses, among others.

3.5.2 *Data Preparation for Machine Learning Algorithms*

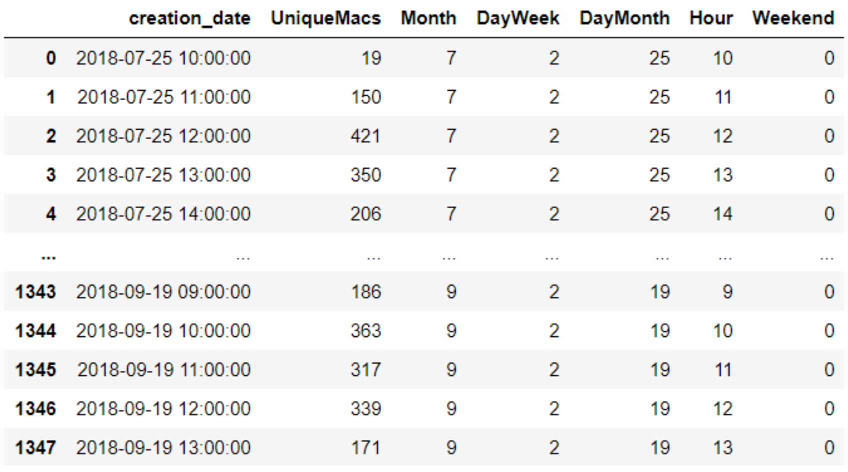
The dataset that was explored didn't have a clear response variable - a variable that would be predicted in order to solve a problem. Since the objective was to predict the number of people, predicting the number of unique MAC addresses would be a good indicator for doing so. Therefore, a new dataset, represented in the Figure 31, was created. It contained a column specifying the number of unique MAC addresses per hour.



	UniqueMacs
creation_date	
2018-07-25 10:00:00	19
2018-07-25 11:00:00	150
2018-07-25 12:00:00	421
2018-07-25 13:00:00	350
2018-07-25 14:00:00	206
...	...
2018-09-19 09:00:00	186
2018-09-19 10:00:00	363
2018-09-19 11:00:00	317
2018-09-19 12:00:00	339
2018-09-19 13:00:00	171

Figure 31: Dataset excerpt containing the response variable.

As ML models normally perform better with more relevant information, one of the first decisions was to perfom feature engineering by adding variables relative to time to the dataset, as shown in Figure 32. As seen in the data exploration, these attributes proved to influence the number of MAC addresses' fluctuations - making these a good option to fuel the models in the future. It is also important to mention that even though these features were created, when experiment with the ML models, multiple combinations of these features were tested. Also, the attribute *creation_date* was dropped posteriorly as its information was now present in other attributes.



	creation_date	UniqueMacs	Month	DayWeek	DayMonth	Hour	Weekend
0	2018-07-25 10:00:00	19	7	2	25	10	0
1	2018-07-25 11:00:00	150	7	2	25	11	0
2	2018-07-25 12:00:00	421	7	2	25	12	0
3	2018-07-25 13:00:00	350	7	2	25	13	0
4	2018-07-25 14:00:00	206	7	2	25	14	0
...	...	...	...	...	...	...	...
1343	2018-09-19 09:00:00	186	9	2	19	9	0
1344	2018-09-19 10:00:00	363	9	2	19	10	0
1345	2018-09-19 11:00:00	317	9	2	19	11	0
1346	2018-09-19 12:00:00	339	9	2	19	12	0
1347	2018-09-19 13:00:00	171	9	2	19	13	0

Figure 32: Dataset excerpt with time attributes.

Moreover, the matrix in Figure 33 shows the correlations between the attributes. It was expected that there could be strong correlations between the time attributes and the response attribute - the number of unique MAC addresses per hour.

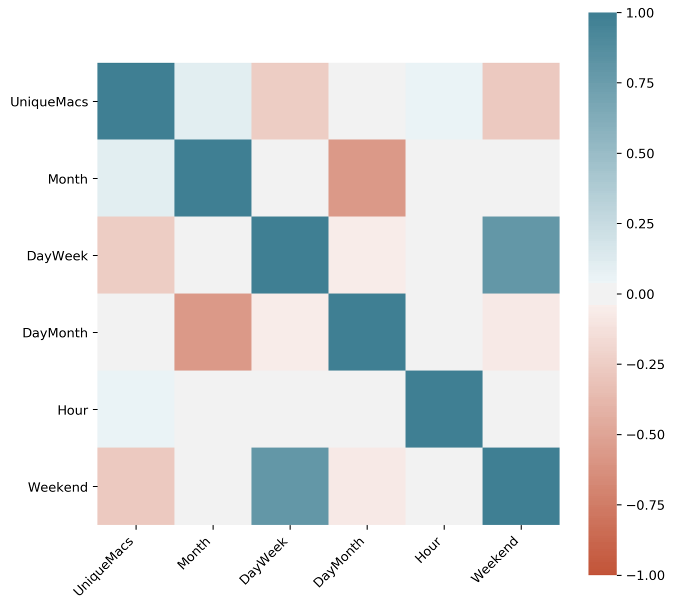
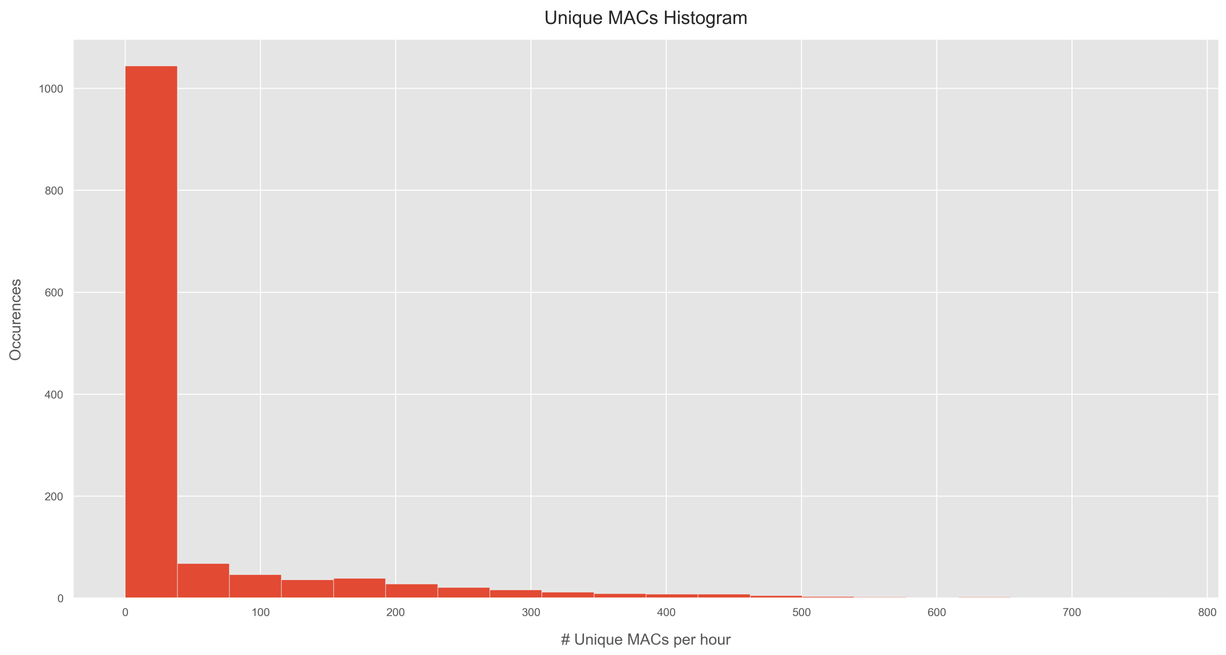


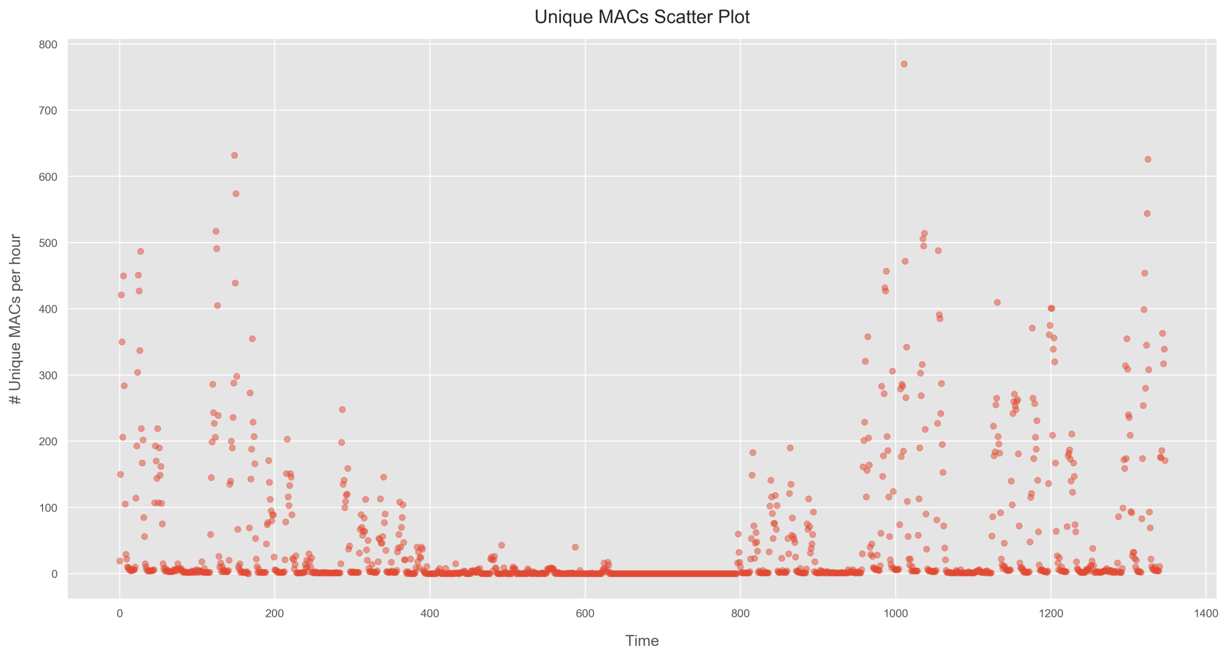
Figure 33: Correlation matrix between the dataset's attributes.

The attributes *Month, DayWeek* and *Weekend* still showed some correlation with the response variable.

One relevant conclusion was that the response variable had mostly low values while having some outliers. As shown in Figure 34, the histogram shows that most of the values of the response variable were in a range of low values (lower than 100). The scatter plot, in turn, besides confirming the previous statement, also shows the presence of outliers in the data.



(a)



(b)

Figure 34: Density and distribution plots on the response variable, represented in a) and b), respectively.

3.5.3 *Machine Learning Models Conception*

The main objective was to build a model capable of predicting crowd density - a regression problem. Therefore, different approaches were tested in order to reach a solution that could provide the most reliable results. These different approaches followed the same process, as illustrated in Figure 35.

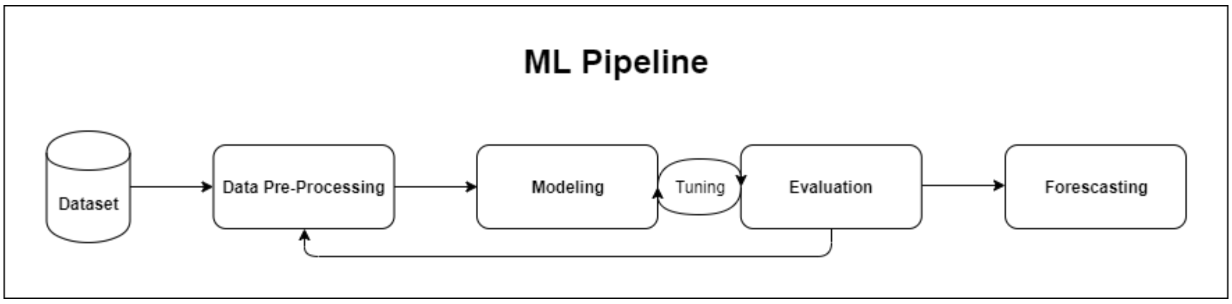


Figure 35: Adopted procedure to conceive the ML models.

From the dataset, the data first goes through a Pre-Processing phase, which prepares the data for the training phase. Several data configurations, model types and parameters are tested to conclude which combinations provides the best results. Having a model trained, it goes through the process of evaluation which implies tuning until the performance is satisfactory. In fact, the type of evaluation also depends on whether the data was regularly divided into one training and one test set, or if it was divided into several training and validation sets. Finally, before reaching the Forecasting phase, the tuned model is trained with the data structure that provided the best results and then its final assessment it's done on an unbiased test set.

Data Pre-Processing

The first step before actually creating and training ML models, is to pre-process the data. The pre-processing of data is not only important, but also necessary to make good models. In Figure 36, the pre-processing pipeline is demonstrated. It contains four different steps. The last three have different options that were all tested in order to see which one would make the most out of the ML model, except for the training/validation/tests sets creation which is responsible for the type of evaluation used later on.

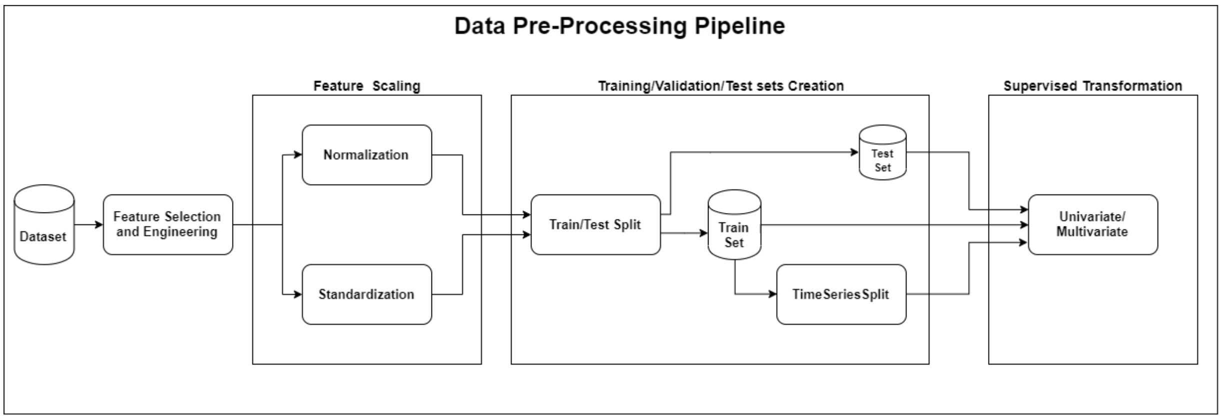

Data Pre-Processing Pipeline
Feature Scaling
Training/Validation/Test sets Creation
Supervised Transformation
Dataset
Feature Selection and Engineering
Normalization
Standardization
Train/Test Split
Test Set
Train Set
TimeSeriesSplit
Univariate/ Multivariate

Figure 36: Pre-Processing Pipeline.

The first step in the pipeline is Feature Selection and Engineering. In this step, the attributes, except for the target attribute, are (or not) removed from the dataset. There are interesting correlations between attributes, in particular with the target attribute, but there is the possibility that some of these attributes don't contribute at all to the ML algorithm. Therefore, experimenting with different combinations of these was one of the decisions made. Next, there is Feature Scaling - a very important process since ML algorithms perform better when the numerical attributes have the same scale. In this step, two scaling methods are used: Normalization (between -1 and 1) and Standardization. After the scaling of the attributes, the Training/Validation/Tests sets creation is responsible for splitting the data so that the model can be trained, tuned and tested with different groups of data, allowing a better evaluation of the model performance. This way, firstly, the Train/Test split divides the data into one training set and one test set with a 83% and 17% ratio. Moreover, the resultant training set is splitted using the *TimeSeriesSplit*, where the data is divided as Figure 37 shows.

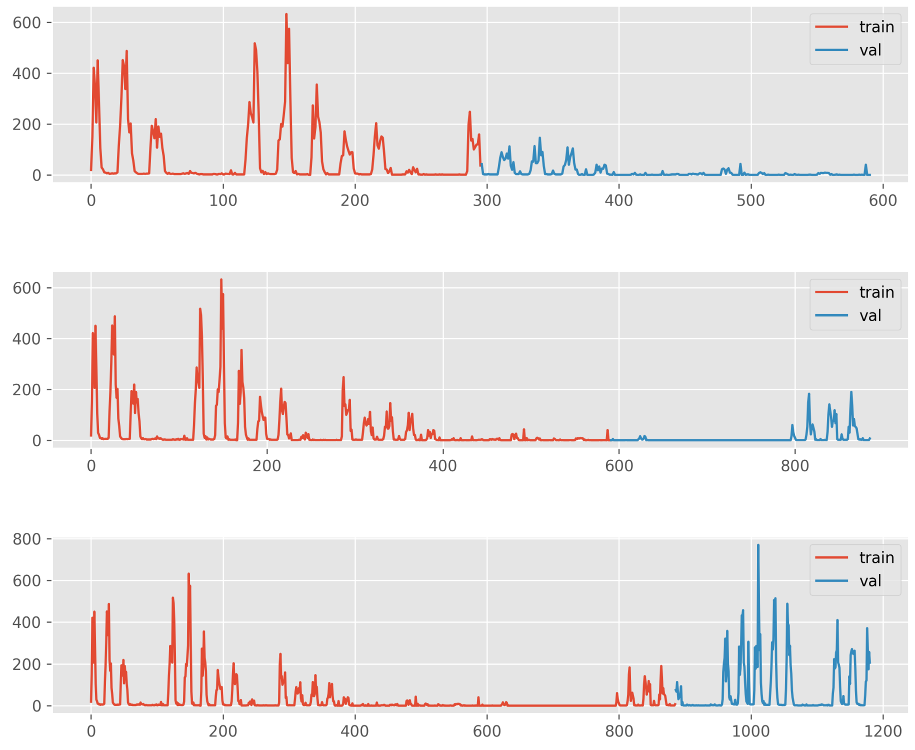


Figure 37: Time Series Split applied to the training set.

The initial training set is divided into 3 folds of training and validation tests, providing better evaluation and therefore better tuning by producing different sets with which the model can be trained and evaluated on, avoiding issues like overfit and producing better models. The 83%-17% ratio is used to so that the model can be tested on a full week of September, which is a month of many variations, being a good candidate for the models to be tested on. Lastly, the test set is solely used for testing the produced tuned models with a completely unbiased set.

It's important to emphasize that the way the *TimeSeriesSplit* was applied, was the best given the available the data. If the data volume was higher, the best option would be to apply the *TimeSeriesSplit* to the full dataset, producing folds that had training, validation and test sets. If this method was applied to the available data, the split would look like Figure 38 shows.

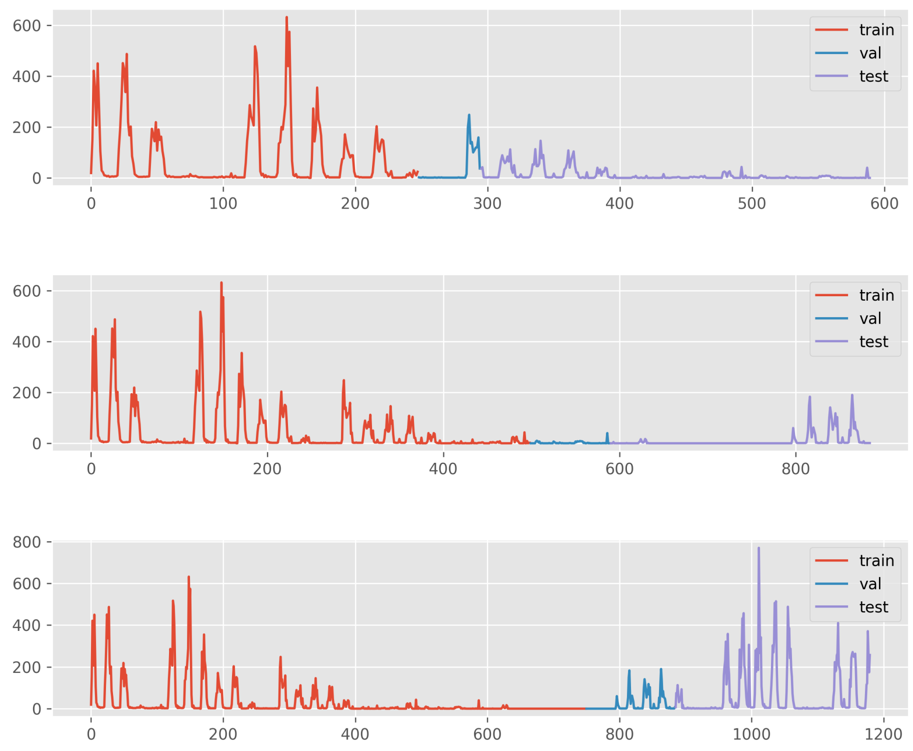


Figure 38: Time Series Split applied to the full set.

This would allow not only the training and tuning on different sets, but also the testing. The issue, in this case, is that the validation data would always be too little and not representative - as it's possible to see in the plots above - not enabling a good tuning and therefore not giving the different tests sets good usage. Consequently, even though the applied Cross-Validation (CV) method only allows the test of the model on a single set of data, it ensures that the model is constantly trained and evaluated on significant validations sets.

Finally, the last phase - Supervised Transformation - consists in reframing this sequence prediction problem into a supervised learning problem, using the *Sliding Window Technique*. This consists in using the values of previous time steps as input variables and use the next time step's value as the output variable. In order to demonstrate the previous transformation, a simpler example will be used. Given a dataset, shown in Table 9a, the supervised transformation produces a new dataset, illustrated in Table 9b.

Table 8: Sliding Window technique applied to a simpler example.

(a) Original Dataset.

Time Step	Y
t	10
t+1	20
t+2	30
t+3	40
t+4	50
.	.
.	.
.	.
t+n	...

(b) Modified Dataset.

Time Step	X	Y
t	[10,20]	30
t+1	[20,30]	40
t+2	[30,40]	50
.	.	.
.	.	.
.	.	.
t+n	...	...

This restructuration allows the prediction of the output variable at a given time step by using the values of previous time steps. Therefore, this is applied to the problem in question by transforming the existent sets of data. As illustrated in Figure 36, it can be divided in two different categories: Univariate or Multivariate. On one hand, the first implies that only one feature is used as an input, meaning that each timestep only includes sequences of one feature. On the other hand, the second means that multiple features are used as inputs, meaning that multiple features are found in every time step. The function in Listing 3 is responsible for this transformation.

Listing 3: Supervised Transformation function.

```
def prepare_supervised_transf(dt_values, target, start_index, end_index, time_steps,
                              dim, target_size):
  data = []
  labels = []
  start_index = start_index + time_steps

  if end_index is None:
    end_index = len(dt_values) - target_size

  for i in range(start_index, end_index):
    indices = range(i-time_steps, i)
    data.append(np.reshape(dt_values[indices], (time_steps, dim)))
    labels.append(target[i+target_size])

  return np.array(data), np.array(labels)
```

By applying the *Sliding Window Technique*, the function above transforms the original dataset, which shape is *(length, n_features)*, into two arrays:

- one relative to the input variables (x), which shape is *(length, n_time_steps, n_features)* and contains various sequences;
- other relative to output variable (y) with the shape *(length, 1)* and contains the number of unique MACs.

So, for every sequence present in *x*, its corresponding output value - the value that needs to be predicted - is present in the array *y*. Moreover, this function can either produce not only univariate, but also multivariate data, depending on the value of the argument *dim* and the argument *dt_values*. For example, if the objective is to produce univariate data, *dim* must be equal to 1 and *dt_values* must be a list containing the values of only one feature. Besides that, the argument *time_steps* is very important since it controls the size of the sequence that is going to be used to predict the next value.

The plot in the Figure 39 - which doesn't contain normalized or standardized values for visualization purposes - shows the result produced by a concrete example of a supervised transformation, where 24 time steps are used to predict the 25th value. In this example, the model would have to predict the number of Unique MAC addresses at 2018-07-26 13:00:00, based on the previous 24 hours.

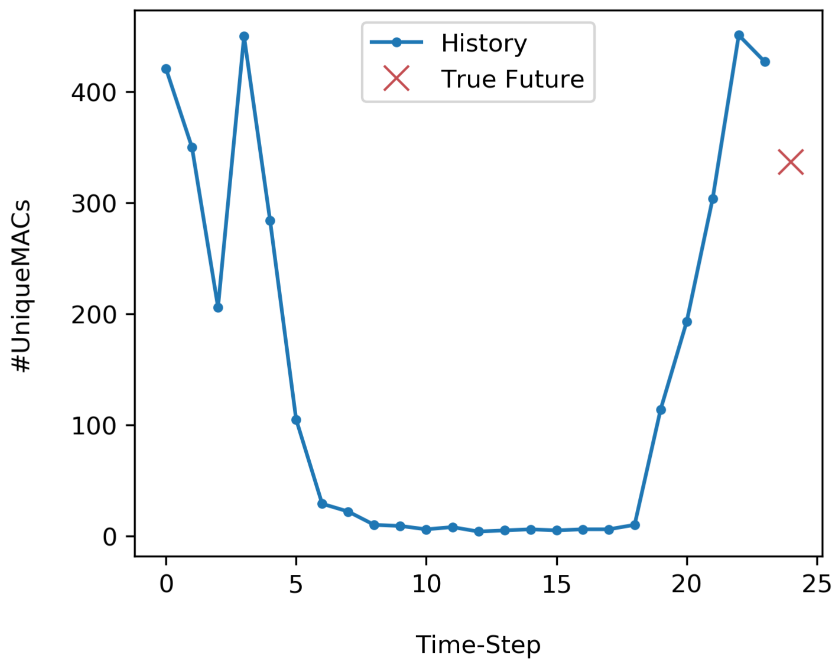


Figure 39: Plot that illustrates an example of a time step's sequence and respective response variable produced by a Multivariate Tranformation.

To conclude, the supervised transformation phase is responsible for preparing the data for entering a ML model. As in other phases, multiple combinations of these kind of transformations are tested, in order to find out which one allows the best performance. The number of time steps present in each sequence, more specifically, the number of hours used to predict the next hour, is the one variable that is varied. Finally, the data is prepared to enter a Single Step Model - a model that typically will predict a single value for each sequence.

Modeling

Having the data processed and prepared for entering a ML model, the next phase is to actually build and train a model. Therefore, the Modeling phase is responsible for creating a model by building and training it, as shown in Figure 40. The group of all the tested models were chosen according to the type of data received. Since the data is from a sequence prediction problem, models like the RNN and the LSTM, that theoretically present more potential of having good results because of its properties, were chosen to tackle the problem along with the ANN, a multi-layer perceptron, which is always a standard to test.

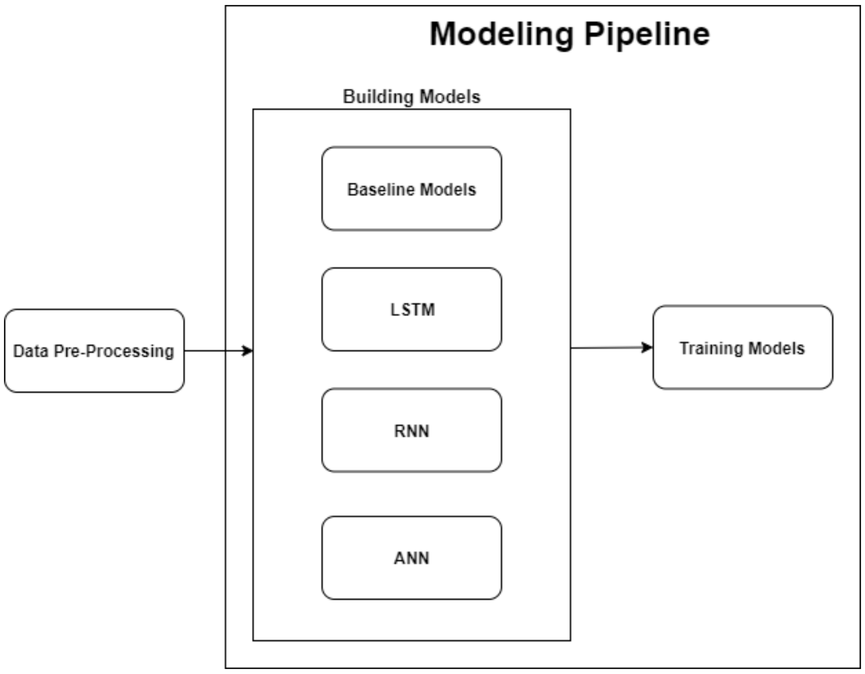


Figure 40: Modeling Pipeline.

One of first things to be done regarding modeling, was to establish baseline models - models that are really simple and still have a chance of presenting good results. They also provide baseline metrics so that more complex models can have a comparison measure. This way, four different baseline models were created:

- Baseline model - a model that simply computes the next value of the sequence by calculating the mean of the sequence values;
- Baseline Network - a ANN with only one input Flatten layer and an output Dense layer;
- Baseline LSTM - Similar to the Baseline network but with one LSTM layer.

Both the baseline network and the baseline LSTM are compiled using the loss function *Mean Average Error (MAE)* and the optimizer *Adam*. Also, they are only trained during 20 epochs to get a quick assessment on its performance with the data provided by the Train/Test split - one training set and one test set - in the Pre-Processing phase and with 24 values in each sequence. Also, other type of Baseline was tested to see if it was even viable: a simple linear regression model.

Apart from the baseline models, other three were explored to a higher degree: LSTMs, RNNs and ANNs. This exploration was divided in two steps: building the models and training them. The first, involves creating and compiling the models. Creating a model is the phase where the type of model, the kind and number of layers used and its number of neurons and activation functions, are defined. To do so, one function per type of model was created, as shown in Listing 4 for the case of the LSTM.

Listing 4: Functions responsible for creating the LSTM model.

```
def create_LSTMmodel(n_neurons, n_features, n_steps, stacked_lstm):

    lstm_model = tf.keras.models.Sequential (name='LSTM_Model')

    if (stacked_lstm == True):
        lstm_model.add (tf.keras.layers.LSTM (n_neurons, return_sequences = True,
            input_shape = (n_steps, n_features)))
        lstm_model.add (tf.keras.layers.LSTM (int(n_neurons/2), return_sequences = True
            ))

    else:
        lstm_model.add (tf.keras.layers.LSTM (n_neurons, input_shape = (n_steps,
            n_features)))

    lstm_model.add (tf.keras.layers.Dense (1,activation="linear"))

    return lstm_model
```

Through the functions' argument it's possible to control different aspects of the model which allowed the quick creation of multiple models. The arguments *n_features* and *n_steps* vary according to the data structure that is going to be used in the training phase. The other two variables, *n_neurons* and *stacked_lstm*, help defining the layers' number of neurons and also how many layers the neural network will have. Additionally, others aspects of models can be changed, like adding other kind of layers (for example, a Dropout layer), by manually changing the function code. This allowed the quick creation of models with different layers and different levels of complexity.

After creating the models, they must be compiled. The compilation phase is where the optimizer, loss function and, optionally, the metrics are defined. Briefly, the choice of the optimizer and loss function will affect the learning algorithm used in the training phase. In fact, multiple optimizers, including its learning rates were tested to assess which combination provided the best results.

The modeling is finished by training the already built model. This phase is where the model actually learns to predict the next value in a sequence by constantly updating its parameters until it converges into a good solution. The function in Listing 5 was used to train the models.

Listing 5: Function responsible for training the models.

```
def train_model (model, x_train, y_train, epochs, val_split = 0,verbose = 1, x_val=[],
                y_val=[]):

    history = model.fit (x_train, y_train, epochs = epochs,
                          validation_split = val_split,
                          shuffle = False, verbose = verbose,
                          batch_size = 32,
                          callbacks=[keras.callbacks.EarlyStopping(
                          patience=25,restore_best_weight = True)])

    return history
```

By receiving the model, training data and the number of training epochs, the function is capable of calling the *fit()* method from the *Keras API* to train the model. These parameters, amongst others, have importance on how the model is trained. Table 10 explains the role of the most important parameters.

Table 10: Parameter's role in the training phase.

Parameter Name	Description
x_train	Contains all the sequences from which the model should learn.
y_train	Contains the next value for each of the x_train's sequences. The loss at the end of each epoch is calculated by comparing the predict value VS the respective value from y_train using the defined loss function.
epochs	Defines the numbers of maximum epochs that the model will be trained on.
validation_split	This is a value that defines how much of the training data is going to be used for validation purposes. Validation data allows better analysis of the model's performance, namely if overfitting exists or not.
shuffle	Shuffle is always define to False. In sequence derived problems, one must not change the order of the data, since this must be preserved.
batch_size	Defines how many sequences will be trained together. This affects not only the model performance, but also the training speed. In these kind of problems, the batch size is even more important since in LSTMs the memory is discarded after each batch.
callbacks	A callback is defined to use *EarlyStopping* - the training stops when the losses in the the validation split stop improving for more than 25 epochs, restoring the models weights from the epoch with the best performance (if *restore_best_weights* is set to *True*).

Different values of the parameters *epochs*, *validation_split*, *batch_size* were tested to have the best performance possible. Moreover, it is important to underline that the training data - *x_train* and *y_train* - varies accordingly the data produced by the Data Pre-Processing phase. Also, the models are both trained with normalized and standardized data in order to see which scaling method works best.

To sum up, in this phase, baseline models are created along with the main models. In these last ones, parameters are constantly changed according to the results from the next phase. In the end, while going back and forth with modeling and evaluation, different kinds of models end up being created, whether in terms of model type, but also regarding model parameters.

Evaluation

One of the most important phases in any ML project is evaluating the model's performance. The main objective is to find the parameters that best suits the model, as well as the type of data. This phase, if not done correctly, can lead to false conclusions, and, ultimately, to a model that doesn't have the capacity of performing well when faced with a slightly different input. This way, the evaluation phase consists in three main steps, illustrated in Figure 41. Firstly, the model's performance assessment is done only with validation sets: in a first stage resorting only to 10% of the training set, and in a second stage using the validations sets provided by the *TimeSeriesSplit*. Afterwards, the model is tuned by going back to the Modeling and/or to the Data Pre-Processing. Finally, after finding the right data configurations and most suitable model with its hyper-parameters, the model is tested on the test set.

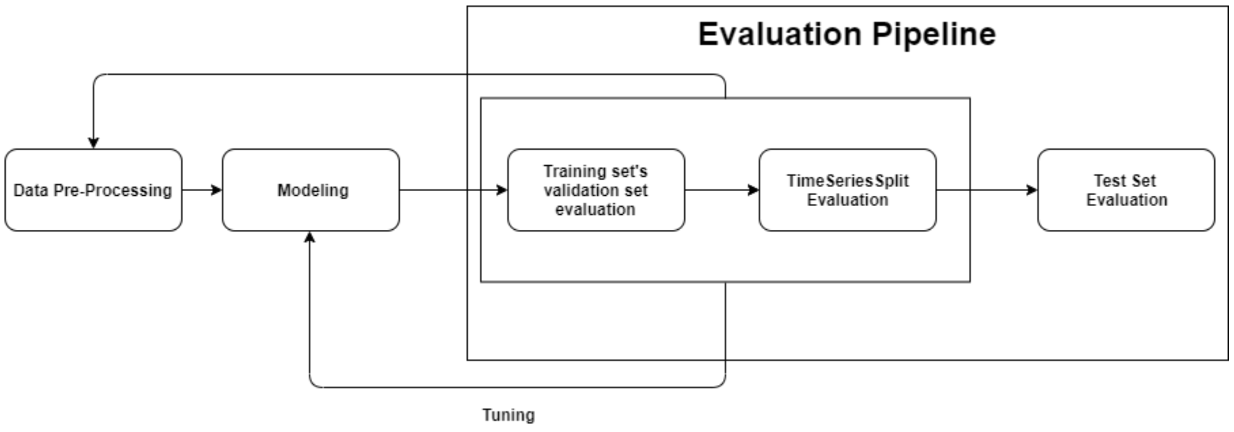


Figure 41: Evaluation Pipeline.

Before explaining the three main steps in the evaluation, three different regression error measures were defined to better assess the model. Each one of these computes the error between the predicted values and the real values, but in different ways, and therefore providing different insights. These are briefly described in Table 11:

Table 11: Error measures.

Metric Name	Description
MAE	Computes the average absolute difference between the predicted and real values. Describes the typical residual and isn't very sensitive to outliers.
MSE	Computes the average squared difference between the predicted and real values. Is gives high weight to larger errors and therefore is very sensitive to outliers.
RMSE	Computes the root of the average squared difference between the predicted and real values. Also useful to identify large the presence of larger residuals.

In the first evaluation stage, the first thing is to analyse the training metrics provided by the training phase. The losses of the training and validation set are studied, as well as other additional metrics defined. Additionally, the training method used makes it possible the plot the learning curves. This allows the identification of whether overfitting or underfitting occured, and also if the training convergence occurred in a good manner. These results can immediately be used to do some tuning. As a matter of fact, parameters like the *optimizer* and its *learning rate*, *batch size*, the number of layers/neurons to adjust the model complexity and also the use of regularization techniques are some of the parameters modified for tuning purposes. Lastly, the defined metrics allow the understanding of the model's performance on the validation set, but these results might not be representative of the model capability on different sets, since it was only tuned and tested in a single sets.

Therefore, a more reliable kind of evaluation - by training and testing the model on different sets of data - is also applied after the previous stage. This procedure is called CV. More specifically, the CV object used is the *TimeSeriesSplit*, which is adequate to sequence problems. In this procedure, the dataset is splitted into 3 sets of data, which contain train and validation sets. The model is then trained on each training set and evaluated on the respective validation set. In the end, a mean of the three evaluations is calculated in order to evaluate the model performance. This way, the model capacity on different sets is more reliably represented in the final metrics - providing a better evaluation and in the end a better tuning phase. A function, illustrated in Listing 6, was created to do this.

Listing 6: Function responsible for training and evaluating using the TimeSeriesSplit.

```
def train_evaluate_tsc (dt,time_steps,features, scaler, model_type,tscv,verbose=1):
    fold_no=0
    metrics = []
    metrics_unscaled = []
    models = []

    for train_index, test_index in tscv.split(dt):
        train_set, test_set = dt[train_index], dt[test_index]
        x_train_mul, y_train_mul = prepare_multivaridata (train_set,
                                                      train_set[:,0],0,None,
                                                      time_steps,features,farmetric)
        x_test_mul, y_test_mul = prepare_multivaridata (test_set,
                                                      test_set[:,0],0,None,
                                                      time_steps,features,farmetric)
        if (model_type == "LSTM"):
            model = create_LSTMmodel (10, features, time_steps, False)
        else:
            if (model_type =="ANN"):
                model = create_ANN (features, time_steps)
            else:
                model = create_RNNmodel (10, features, time_steps, True)
        model  = compile_model (model,"RMSprop", loss= "mae")
        history =  (train_model (model,x_train_mul, y_train_mul, val_split=0.1, epochs
            =200, verbose = verbose))
        metrics.append ((get_metrics (model, x_test_mul, y_test_mul, scaler))[0])
        metrics_unscaled.append ((get_metrics (model, x_test_mul, y_test_mul, scaler))
            [1])
        models.append (model)
        fold_no += 1
    metrics_names = ["MAE","MSE","RMSE"]
    print ("Average scores for all folds:")
    for i, metric in enumerate(metrics_names):
         print (metric, "error", np.mean(((np.array(metrics)).T) [i]),"/", np.mean(((np
             .array(metrics_unscaled)).T) [i]))

    return metrics,models
```

Basically, for each set produced by the *TimeSeriesSplit*, the data is transformed with the supervised transformation and splitted into train and validation sets. Next, according to the model chosen in the argument *model_type*, the model is created and then trained. The model is subsequently evaluated through error measures on the respective validation set, and these are saved on a list. This process is repeated for each *TimeSeriesSplit's* fold, and at the end the mean of each metric is calculated. Moreover, this function also allows the quick test of different sequences, by changing the functions arguments *time_steps* and *features*, which will control the length of sequences and also how many features each sequence has. Finally, the arguments *dt* and *scaler*, are important to define the kind of feature scaling used and the number of features present.

Concerning the tuning part, it was done both manually and also in an automated way. The automated way was done using the *RandomizedSearchCV*, which, as the name suggests, performs a randomized search on the hyper-parameters. It was used only after the best data structure was found, and it took into account other conclusions that restrained the universe in which the search was conducted in. Additionally, it also uses the same CV method used previously to divide the training set.

Lastly, the model is evaluated on the test set by using the already trained and tuned model to predict the values of the test set and then using the defined *error measures* to compare them to the real values.

Forecasting

The next and final phase of the pipeline is Forecasting. The objective is to forecast the number of unique MAC addresses, based on past data. To do so, two types of prediction were explored. One, consists in using the single-step model to predict the next value of a sequence. This is done by simply using the method *predict ()* of the *Keras API* model class. This method is applied to the model and as an input the method receives the sequences from the test set.

Another kind of prediction was also explored. Since the model is a single-step model, predicting multiple values from a sequence was also explored. One way to do this could be building a multi-step model, a model that would immediately output more than one value. Another way, the one explored, was to forecast recursively. For a single sequence, the prediction should have more than one value. The strategy was to have the model predict the next value and then use the predicted value as an actual value from the sequence - forming the sequence for the next time step. The new sequence would then be used as the input sequence and the process would be repeated. The following function, represented in Listing 7, represents this strategy for an univariate model.

Listing 7: Recursive Forecast.

```
def multistep_forecast (model, x_values, time_steps, features, n_steps):

    x_values = x_valid_uni[0].reshape (1,time_steps,features)

    for steps_ahead in range (n_steps):

        y_single_pred = lstm_model.predict (x_values [:,steps_ahead:])
        x_values = np.concatenate ((x_values,y_single_pred.reshape (1,1,1)), axis=1)

    return (x_values [:,time_steps:].reshape((n_steps, 1)))
```

4 RESULTS AND DISCUSSION

The developed pipeline produced several results by testing different data structures, different models and respective parameters. The first section, Preliminary Analysis, through the first stage of the Evaluation - where the model is only tested against a validation set from the training set - will demonstrate how different model's and training parameters affect the error measures and training phase convergence. The second one, Tuning phase, which took into account the conclusions from the preliminary analysis, performs numerous tests using the *TimeSeriesSplit* evaluation to find the optimal data structure and the optimal model.

In the end, through the constant experimentation and tuning, the data structure associated to the best performance model is found, making the prediction of crowd density feasible in real-world contexts.

4.1 PRELIMINARY ANALYSIS

The first thing to do was defining a data structure. It had 24 time steps each one containing 6 attributes: *UniqueMacs*, *Month*, *DayWeek*, *DayMonth*, *Hour* and *Weekend*. Then the baseline models were tested and all of them showed the potential for good results, except for the linear regression model. Moreover, since the LSTM was the most promising because of its properties, this type of model was tested manually with several different parameters. Going back and forth with modeling and evaluation, it was possible to conclude some points and make decisions.

First, regarding the model's complexity, which is increased by the number of layers, neurons and activation functions, the main thing to notice was that if the model was **more complex, the performance would decrease**. Increasing the model's complexity and at the same time applying regularization techniques could be a option that would have good results in both train and validation sets, but it only decreased the overfitting whilst not increasing the final validation set performance.

In terms of the model compilation, which include mainly parameters like the loss function and the optimizer, it's important to mention some aspects. In regression problems, the loss function normally varies between the MSE or MAE. In fact, both can produce models with different characteristics. Having this in mind, LSTM models were trained multiple times using these two loss functions.

On one hand, the model that used the MAE as its loss function optimized this metric by reducing the average difference between the predictions and the real values, producing more consistent predictions. On the other hand, using the MSE as the loss function, enabled the model to produce predictions that don't have a big difference to the real values, following the peaks values much better when compared to the MAE. Given this fact, in this point a decision was taken to define the MSE as the loss function. When wanting to predict the number of people per hour, the really high/low number of this measure must not be neglected since this can be a key factor for making decisions based on these predictions. A possible counter argument for this decision could be that peak values could not be representative of the actual number of persons and therefore these shouldn't be given much weight. But, actually, these peaks are produced by unique MAC addresses, so there is not one device responsible for the immense number of probe requests.

Between the possible optimizers, several were tested and the following had the best results: *RMSprop* and *Adam*. The result was not surprising since the *Adam* optimizer is widely used as it generally provides good results. Also, the learning rate was varied.

The error metrics didn't decrease, but instead the only aspect that varied was the convergence speed. Since this wasn't an issue, the learning rate was dropped as a parameter to be tested manually. The same result was also applied when the batch size was varied.

After testing several LSTM configurations, two more models were tested: RNN and ANN. Like the LSTM, these two models also presented the same kind of results in terms of the model complexity, loss function and optimizer. Moreover, the results were also promising and surprisingly similar to the LSTM which, right from the start, had the best prospects.

To conclude, this phase allowed us to obtain some initial insights and also to make some decisions. On one hand, using MSE as the loss function showed to have more potential given the nature of the problem being faced. On the other hand, less complex models seemed to perform better as well. Also, the optimizer to be used was defined since it provided the best results. Moreover, this phase showed that all models have a good foundation to work with. None of them had a really differentiated performance from each other, which enabled all of them to continue to be tested and improved. All the tuning done in this phase was manual. In the next phase, the models' hyper-parameters will be tuned in an automated way.

4.2 TUNING PHASE

This phase, unlike the previous one, which was mainly to gather more initial insights on the model's hyper-parameters, will show which data structure is better to fuel the model and also will tune the hyper-parameters in an automated manner. In the end, the best performing model along with the proper data structure, will be found.

It should be noted that the results presented in this section are always the average of the same test done multiple times. Since the training algorithm used to update the networks' weights does not always converge in the same manner, averaging the results of multiple tests is the most reliable method to precise a model's performance.

4.2.1 *Data Structure*

Regarding the data structure, the first thing to do was to analyse which type of feature scaling provided the best results. To do so, the three model types tested in the previous phase were explored using CV, with the initially defined data structure of 24 time steps and 6 features, and varying the feature scaling method, as shown in Table 12.

Table 12: Feature Scaling Results.

Feature Scaling	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
Norm	48.6	66.47	53.45	82.17	54.08	76.25
Std	40.51	63.84	37.21	56.73	45.56	62.01

Table 12 shows the difference between two different methods of feature scaling. Three different models were trained and evaluated multiple times, and the average results of each one of them demonstrate that standardization provides the best results for both the three models being tested and the two metrics MAE and RMSE. From this point, standardization was defined as the feature scaling method to be used.

Features

Posteriorly, the number and the type of features were tested. The objective was to explore the effect of these attributes on the model's performance. Before explaining this procedure, it's worth to remind the data structure shape: *(length, n_time_steps, n_features)*. In this procedure, the number of features or *n_features* was in a first instance decreased according to the attributes more or less correlated to the response variable - dropping the less correlated attributes gradually - creating 3 different combinations of attributes: the first, with 6 attributes that initially defined; the second, with the same attributes as the previous structure but dropping *DayWeek* and *DayMonth*; the third, with only one 1 feature, *UniqueMacs*. The results are depicted in the Table 13.

Table 13: Feature Experimentation Results.

Data structure (time_steps, n_features)	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
(24, 6)	40.51	63.84	37.21	56.73	45.56	62.01
(24, 4)	35.90	59.06	33.72	50.23	30.03	51.40
(24, 1)	21.28	37.64	24.17	44.25	18.32	34.90

From this table, it's easy to conclude that the performance increases when the number of features is decreased. This implies that the attributes used were not relevant and therefore were not helping the model's performance. At this point, a review of the feature engineering/selection phase was imminent. But before creating new features, an experiment was made to dictate whether any of the existent attributes were relevant or not. This was done by testing a structure with a low number but more correlated features. Therefore, the test, showed in Table 14, was conducted with 24 time steps and 2 features, where the added feature was the one that was more correlated: *Weekend*.

Table 14: Most Correlated Feature Results.

Data structure (time_steps, n_features)	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
(24, 2)	32.45	52.59	27.26	45.61	22.09	38.95

Still, the performance achieved by this structure was better than the previous ones - (24, 6) and (24, 4). Consequently, these results leaded to a decision of going back to the data pre-processing phase, more specifically by making a review of the feature engineering/selection phase with the aid of the already done data analysis (Section 3.3). In this review, features that had a higher correlation to the response variable were created: a feature that identified working hours and one that identified the holiday month. As a matter of fact, whilst the most correlated attribute before this review had a standard correlation coefficient of -0.27, these two new attributes had, respectively, 0.46 and -0.32.

The first tested structure included these two attributes, plus the *Weekend* attribute since it already had some correlation ratio and finally the response variable: *UniqueMacs*. The performance, shown in the first row of Table 15, was better than all the other combinations of features tested, apart from the one using only 1 feature. This was a good indicator that the relevance of the attributes was taking part on the models' performance. Also, the results until this point showed that the models benefited from using only features with good correlation coefficients, even if that meant using a low number of them. Therefore, the most correlated attribute created until this point was tested along with the response variable (shown in the second row of Table 15), forming a data structure with 2 features: *UniqueMacs* and *WorkHour*.

Table 15: Feature Engineering Results.

Data structure (time_steps, n_features)	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
(24, 4)	26.04	44.15	26.73	43.84	25.09	40.89
(24, 2)	22.5	39.77	22.34	38.95	21.5	36.46

In the case of the RNN model, this data structure offered the best results yet, whereas in the case of the LSTM the results were very similar to best performing LSTM yet (with the (24, 1) structure) and, finally, the ANN wasn't too far from the its best performing structure too ((24, 1)). But, there was still room for improvement in terms of feature engineering, so two new attributes were created: *HolidayorWeekend* and *LessDensity*. The first, identified if the entry of the dataset was registered in a holiday month or in a weekend, producing a correlation coefficient of *-0.49*. This feature was tested along with the *WorkHour* feature and the response variable. The data analysis showed that even in a holiday month or in a weekend, the working hour was still a factor the influenced the density of persons per hour. Secondly, the *LessDensity* attribute identifies all the cases that are prone to have few persons: including being in holiday month, a weekend or out of working hours. This attribute produced a correlation coefficient of -0.81 (the highest yet), however it neglected some aspects, such as, for example, the distinction between working hours at regular weekdays and working hours at weekends, which had different levels of crowd density. In the end, two separate data structures were tested, the results of which are demonstrated in Table 16. The first row represents the structure with three features: *HolidayOrWeekend*, *WorkHour* and *UniqueMacs*. And, the second, the structure with two features: *LessDensity* and *UniqueMacs*.

Table 16: Final Features Results.

Data structure (time_steps, n_features)	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
(24, 3)	19.03	35.78	21.2	37.43	21.8	37.98
(24, 2)	19.62	35.95	20.45	36.23	18.83	35.01

These results showed that the tested features combination had good results. In fact, both the LSTM and RNN and their best performance with these tests. The ANN had a similar result to it's best performing structure ((24, 1)). With all these experiments, the data structure that provided the best results for each model was found, concluding the tuning stage for the parameter *n_features*.

Time Steps

The number of time steps, this is, the number of hours that the model uses to predict the number of unique MAC addresses in the next hour, is another parameter of the data structure that was tested. We could argue that a model that needs less information - less time steps, in this case - to make an accurate prediction, would be a better model than the one that needs more data. Additionally the models' batch size is an aspect that was also tested, since in LSTMs and RNNs this parameter controls when the model's memory is discarded. Firstly, the three usual models will be tested by gradually decreasing the number of time steps. The features used were a combination that already had provided some good results: *UniqueMacs* and *LessDensity*. Throughout this experience these features will not be changed, so that the effect of the timesteps' variation is clear. Finally, the batch size for this experience was set to 32. These were the results of the first phase (Table 17):

Table 17: Time Steps Experimentation results.

Data structure (time_steps, n_features)	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
(48, 2)	20.71	36.69	22.12	40.27	23.89	38.53
(24, 2)	19.62	35.95	20.45	36.23	18.83	35.01
(12, 2)	19.48	36.13	21.21	37.30	21.16	37.2

The findings showed that there were no improvements in increasing the time steps number, in this case to 48. Furthermore, reducing the time steps to 12 didn't produce any significant variations, having in the ANN model case reduced the performance. Consequently, the number of time steps was defined to 24 as it provided consistent good results when compared to other alternatives. When it comes to the batch size, the procedure was simple. Having the result of a defined data structure - whether in terms of time steps and features - the batch size would be varied in order to see its effect on the final metrics. In terms of training speed, increasing the batch size proved to speed up the training, but in terms of results, it produced similar or worse evaluations when compared to the batch size used for all the previous testings, which was 32. Reducing the batch size decreased severally the training speed, having been set aside for this reason.

Summary

The experimentation, using CV with different feature scaling methods, features, time steps and batch size, leaded to decisions that tuned three different models by finding their optimal data structures. From the initial results, the different testing studies made the models suffer a great increase in performance, which showed the importance of proper data for fueling the models. The conclusions taken were similar for all the models, but different in some aspects. In the case of the LSTM model, the standardized data with 24 time steps and 3 features (*UniqueMacs*, *HolidayOrWeekend* and *WorkHour*), provided the best results. For the RNN model, using 24 time steps and 2 features (*UniqueMacs* and *LessDensity*), while also having standardized data, provided its best performance. Finally, for the ANN model, using the same feature scaling method, the data structures with 24 time steps and 1 feature or 24 time steps and 2 (*UniqueMacs* and *LessDensity*), provided very similar results. In this case, it was defined the structure with two features as being the optimal one.

4.2.2 *The 3rd Fold problem*

In the above testing, the results were improved considerably but the best results still had high measure values (considering the nature of the response variable), such as, approximately, 19 for MAE and 36 for RMSE.

The training set is divided into 3 folds, which in their turn contain training and validation sets. The cause of the error inflation was in the third fold. It had really high values of MAE and RMSE when compared to the first two folds. While the models' average best results for the three folds was rounding around 19 for MAE, the models always produced errors of around 36 for the third fold, a value above the mean of the three folds. The same was applied to RMSE, presenting a RMSE value of approximately 36 as the three folds average, whilst having values of around 68 for the third fold, being again above the three folds mean.

Taking a deeper look into each of the three folds, the 3rd fold's validation set, is much more challenging when compared to the other two folds. It has more fluctuations and also these have much higher peaks of values, making these values more challenging to predict. Additionally, the training set used in the third fold is not representative enough. The training set includes six days of July and almost all the month of August. July has some representative variations but they are not many, and August is a month where the number of unique MACs is really low and doesn't have much significant variations. Consequently, the training set lacks in quantity and quality not representing really well the testing environment, causing sample bias. Nonetheless, it's important that the models have the ability to perform well when faced with different kinds of data, and more even when (in the case of the 3rd fold) faced with a month where the model could have practical use.

An aspect that could be limiting the models' performance on the 3rd fold, could be that the models were sticking to much to the training data, overfitting it. Since the validation set on the 3rd fold is quite different from what appears in the training set, the use of techniques to avoid overfitting could help enhancing the performance on this fold. The best solution would be collecting more (representative) training data. Nonetheless, by lowering the models' capacity of memorizing the training data, there's a chance that they could then perform better on the 3rd fold. This way, two different regularization techniques were used - dropout and l2 regularization - among the increase of training epochs and change of loss function to MAE, which optimized the models according to the CV. In Table 18, the three models were tested with the referred regularization methods.

Table 18: Regularizer Results for the three models.

Regularizer	**3 Folds Average**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
Dropout (0.30)	13.51	32.28	15.58	35.04	14.74	33.07
Dropout (0.50)	14.88	34.34	15.55	36.41	15.25	34.70
l2 (0.001)	13.57	31.30	15.90	35.54	15.73	32.82
l2 (0.001) + Dropout (0.30)	13.30	32.10	15.51	35.29	15.08	33.08
l1_l2 (0.001)	13.14	31.12	15.72	35.60	15.58	32.74
l1_l2 (0.001) + Dropout (0.30)	15.79	32.73	15.31	35.13	16.10	34.11
Regularizer	**3rd Fold**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
Dropout (0.30)	28.28	65.15	33.02	72.30	32.20	68.58
Dropout (0.50)	29.60	68.64	34.01	77.49	33.80	72.75
l2 (0.001)	29.54	65.00	33.62	72.42	34.83	67.49
l2 (0.001) + Dropout (0.30)	27.90	65.43	32.72	72.99	32.70	68.79
l1_l2 (0.001)	27.43	62.26	32.89	72.64	34.57	67.54
l1_l2 (0.001) + Dropout (0.30)	35.52	67.35	32.67	73.15	34.74	71.73

For the LSTM model, there were improvements both in the 3 Folds Average and in the 3rd fold, specially using the l1_l2 regularizer. Moreover, in the RNN case, there were also improvements on both evaluations. Additionally, using dropout with a 0.50 rate reduced significantly the performance in the 3rd fold. Finally, the ANN model had a similar performance compared to the RNN, but presented better RMSE results.

In conclusion, the insight on why there were high error values, allowed to produce solutions that leaded to some improvements. For example, before applying the change of the loss function and also regularization, the best performance LSTM had a MAE of 19.03 and a RMSE of 35.78 for the average of the 3 folds, and a MAE of 36.84 and a RMSE of 72.68 for the 3rd fold. After the changes, the modified model had a MAE of 13.14 and a RMSE of 31.12 for the average of the 3 folds, and MAE of 27.43 and RMSE of 67.35 for the 3rd fold. Furthermore, it allowed to also restrain the universe in which the hyper-parameterization, which will be explained in the next section, would explore. Last but not least, it's import to emphasise that given the nature of the training data, the best solution of all would be to collect more data, giving the models more relevant data to learn with and therefore increasing its performance.

4.3 HYPER-PARAMETERS OPTIMIZATION & FINAL MODEL

In this section, the three models are going to go through a hyper-parameters optimization phase, taking into account all the conclusions reached until to this point. On one hand, the data structure experiments enabled to find the best combination of time steps and features that could provide the best performance for each model. On the other hand, the deeper analysis on the metrics provided insights that leaded to the change of the training's loss function, as well as the use of regularization techniques. These last experiment with the regularizers, allowed to understand which of them boosted each model's performance. Therefore, the universe in which the hyper-parameters optimization will happen was shortened by the previous experiments. In fact, the parameter distribution in which the randomized search, a automated hyper-parameters optimization method, did its search, it's presented in Table 20 for each model.

Table 20: Parameter Distributions used in the Randomized Search.

Parameter	**Distribution**	
	LSTM	RNN & ANN
n_hidden	[0, 1, 2, 3, 4, 5]	[0, 1, 2, 3, 4, 5]
n_neurons	np.arange(1, 100)	np.arange(1, 100)
learning_rate	reciprocal(3e-4, 3e-2)	reciprocal(3e-4, 3e-2)
drop_out	[0, 0.2, 0.3, 0.35]	[0, 0.2, 0.3, 0.35]
regularizer	[l1_l2, l2]	None
regularizer_rate	[0, 0.01, 0.001, 0.0001]	None
optimizer	[Adam, RMSprop]	[Adam, RMSprop]

The randomized search, on the LSTM and ANN models, searched for 60 iterations making a total of 180 fits, during a period of, respectively, 2 hours and 1 hour. Moreover, for the RNN the search was limited to 18 iterations, making a total of 45 fits, and taking 3 hours to do so, being this the reason for a less amount of searching iterations. The combination of searched hyper-parameters that optimized the models, both for the metric MAE and RMSE, are displayed in Table21.

Table 21: Optimal Parameters found in the Randomized Search.

Parameter	**Value**		
	LSTM	RNN	ANN
n_hidden	0	0	0
n_neurons	25	49	87
learning_rate	0.001	0.0085	0.00066
drop_out	0	0.3	0
regularizer	l1_l2	None	None
regularizer_rate	0.001	None	None
l2	None	0.0001	0.0001
optimizer	Adam	Adam	RMSprop

All optimized models don't have hidden layers and a significant number of neurons, meaning that models with a low complex degree perform better - confirming an hypothesis that was already mentioned in the preliminary analysis. Furthermore, all optimized models present some regularization whether in terms of drop out or of penalizer (l2 or l1_l2), also confirming previous work done.

The results from these models were compared with the best performing models of each model type until the randomized search. Additionally, the models were not only evaluated with the average of the 3 folds (using CV), but also evaluated on the test set, which had never been tested yet. In Table 23a, the *TimeSeriesSplit* results in the training set are presented, where as Table 23b presents the results from the test set.

Table 22: Hyper-Parameters Optimized Models vs Non Hyper-Paramaters Optimized Models.

(a) Average Results on the training fold's validation sets.

Hyper-Parameters Optimization	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
Yes	13.17	31.25	15.17	32.67	14.73	31.94
No	13.30	32.10	15.51	35.29	14.74	33.07

(b) Results on the test set.

Hyper-Parameters Optimization	**Results**					
	LSTM		RNN		ANN	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
Yes	23.62	49.42	29.33	56.96	30.68	56.78
No	23.40	49.91	32.81	63.67	30.91	56.56

The best performing model in the test set was the optimized LSTM, since the RMSE it's the preferred metric and the MAE error difference to the non-optimized model is not significant. Also, this model was the one that performed best in the training set's validation folds produced by the *TimeSeriesSplit*. For a better interpretation of the metrics produced on the set, the plot in Figure 42 compares the predictions from the best performing model to the real values in the test set.

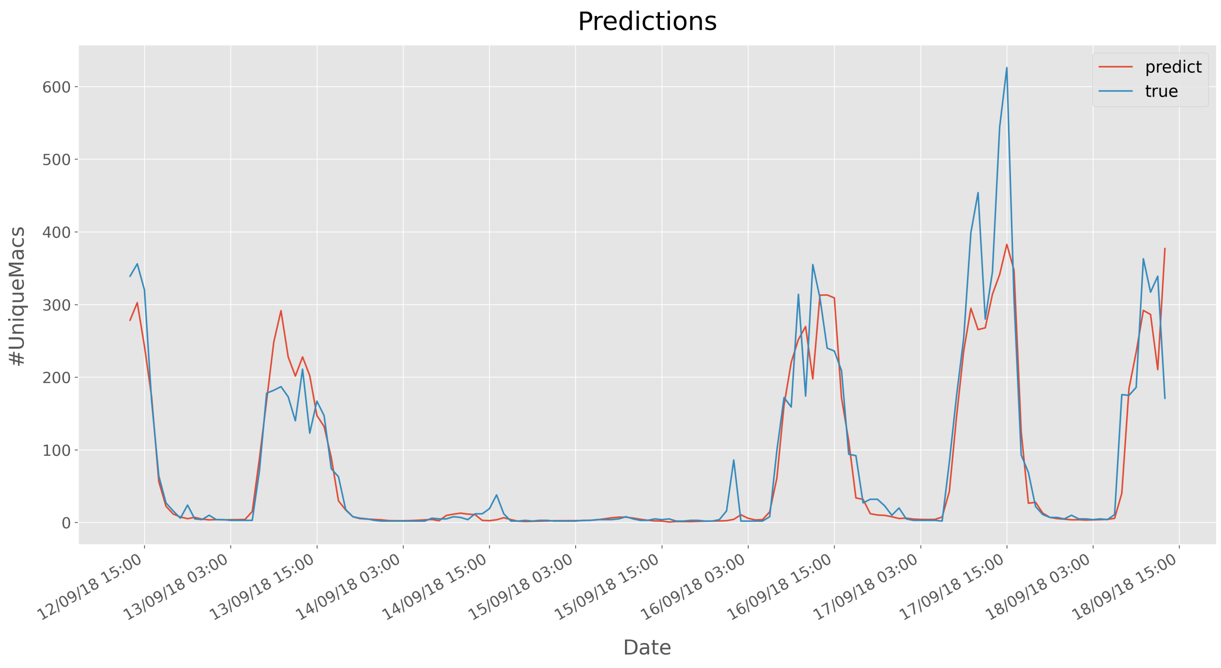
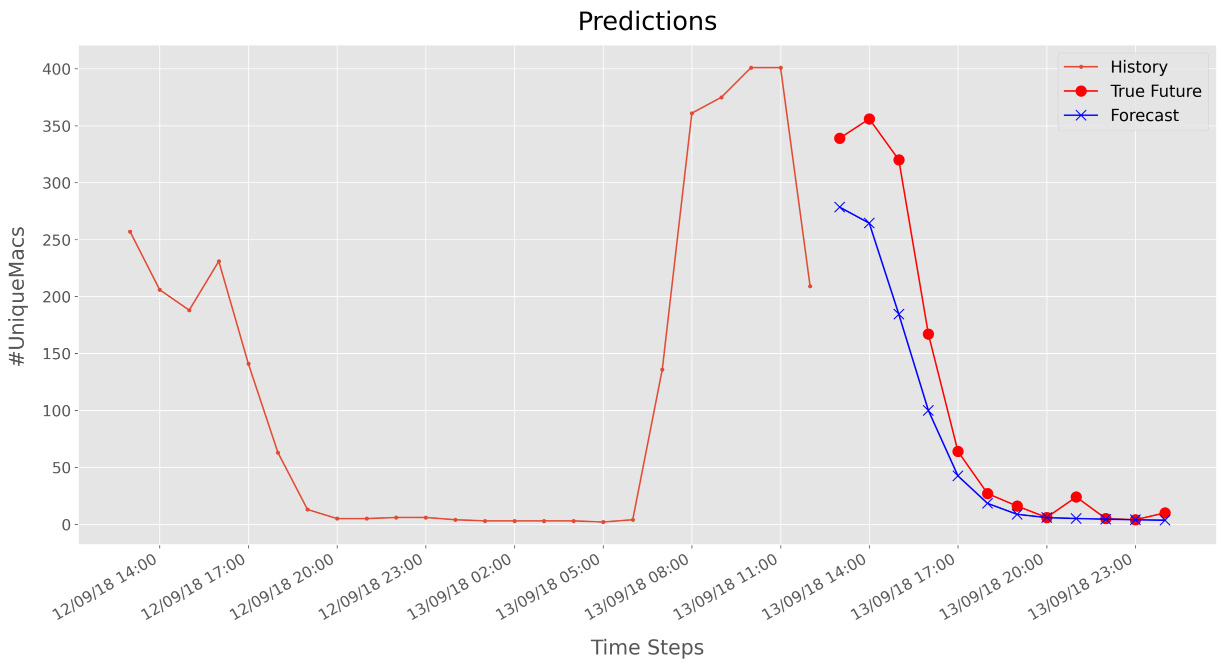


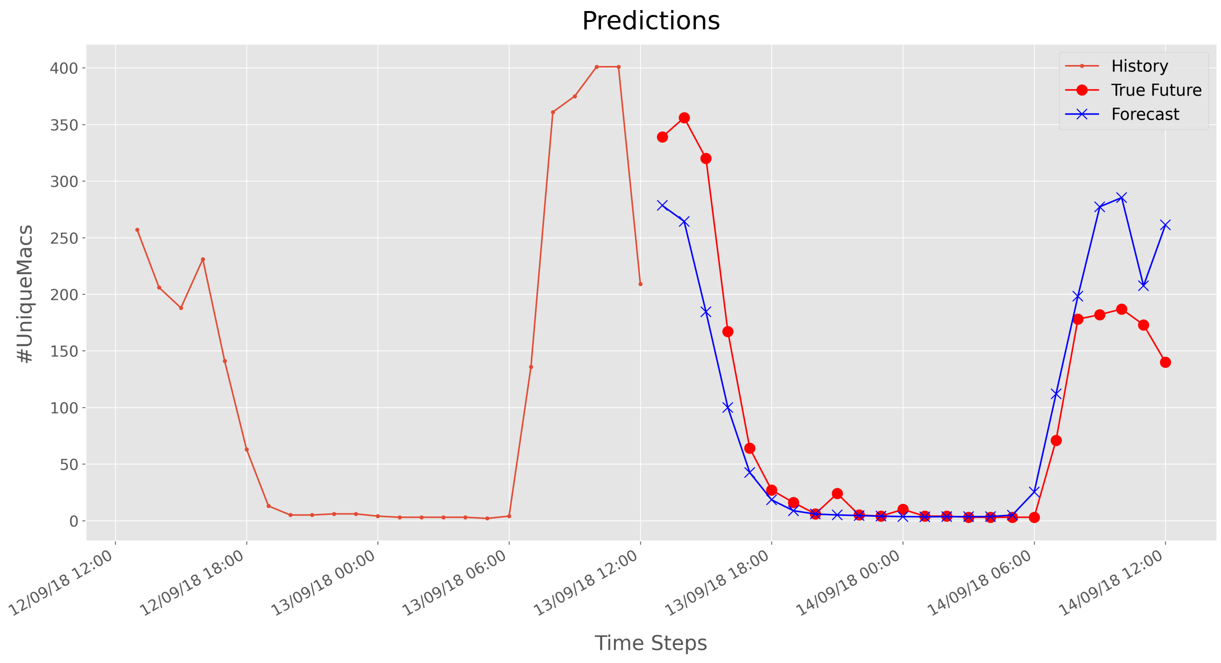
Figure 42: Best model predictions vs real values in the test set.

The predictions are consistent and follow the true values' variations very well. There is a maximum threshold prediction value, around 350, that the model does not exceed, making it have a worse performance when the number of persons per hour is really high, as shown at *17/09/18 15:00*.

The last results prove that this model can have good performance on unseen data. In reality, it is a model that with the data of 24 hours, can predict with reasonable precision the number of persons at the 25th hour. But, this model would be much more valuable if it could make predictions for multiple time steps ahead. For this reason, the recursive forecast was applied, making the model predict values for the next 10 hours and 24 hours, as shown in Figure 43.



(a) Recursive forecast up to 10 hours ahead,



(b) Recursive forecast up to 24 hours ahead.

Figure 43: Multi-Step Predictions generated using recursive forecast.

The results show that in general the model is able to follow the variations very close even when predicting up to several hours ahead. Nevertheless, the second plot in

Figure 43 shows that the model failed to follow the variation in the last hour. This was expected, as the used prediction method mainly relies on using the already predicted values, which makes the prediction error increase as number of predictions increases.

4.4 SUMMARY

The preliminary analysis was useful to indicate which aspects were more important to have a model with good performance. Besides indicating that the three initially defined models (LSTM, RNN and ANN) had potential to be explored, it mainly indicated that less complex models would get better results, which allowed the tuning phase to be more focused in the data structure, which, in turn, provided great increases in the models' performance.

The data structure tests were divided into features and time steps. In the first, the main finding was that feature engineering enhanced all models, making them have increased robustness and accuracy. In fact, the literature review on several crowd sensing approaches had already proved that this process produced good results in similar contexts, and the present problem was no exception. Also, the creation of new attributes was done because the ones being used until then weren't producing satisfactory results. In addition, using a low number of features also helped making better models. Apart from this, the variation of time steps wasn't able to produce any significant improvements, as well as the batch size variation.

It was also found that the used training data wasn't representative or numerous when compared to the validation data in the 3rd fold, which was being the main reason for the lack of model performance. The use of regularization techniques in general slightly boosted the models' performance, by making them stick less to the training data. Moreover, the hyper-parameters optimization using the randomized search method didn't lead to significant improvements, but still managed to enhance performances.

Resultant from all the tests done, the best performing model in both training and test sets was a LSTM with only one input layer and one output layer. The input layer had 25 neurons and used the l1_l2 regularizer with a 0.001 rate. It was trained with a data structure of 24 time steps and 3 features, including the feature being predicted. This model was able to forecast up to 10 hours ahead with reasonable precision, making it adequate for real-world contexts where the crowd density forecasting is useful to enhance decision making in different areas of our lives.

In conclusion, the three models all used some kind of regularization technique, all had a low complex degree, used standardized data, MAE as their loss function, and relevant attributes in their data structure. The use of the CRISP-DM methodology promoted the constant adaptation to the results by going back and forth in the phases defined in the ML pipeline. Finally, LSTMs, the model that theoretically presented better chances of having the best performance, ended up exactly being the best performing one. Despite this, this model needed the right data pre-processing, as well as the correct use of loss function and hyper-parameters.

5 CONCLUSION

The first section will review the results achieved and how they met the initially proposed objectives and the elicited research hypothesis. Additionally, it will address what could be done in the future to improve this work and to make the most out of it.

5.1 CONCLUSIONS

The main of objective of this dissertation was to develop a feasible solution to estimate crowd density, more specifically, by using a smart scanner to passively detect devices such as smartphones or smartwatches through probe requests emitted by these devices. The sensing method would take into account aspects like privacy, type of sensing device, technologies used and the user role in the sensing phase. Furthermore, a RSSI experiment would be conducted to explore the capacities of the chosen smart scanner and the collected data would also be used to conceive and develop ML models to forecast the density of the sensed areas. In the end, the objective was to prove that this crowd sensing approach had results that could be applied to real-life cases, ultimately making a possible contribution to the quality of life improvement in these.

The State of Art section was important for several reasons. Firstly, it showed how Smart Cities and AmI are integrated with each other and how they can be useful to our daily life. The deeper study into AmI allowed the awareness of how its applications are organized and what aspects are important when developing solutions in this paradigm.

In fact, it showed that privacy and security challenges must be acknowledged. Secondly, it confirmed that the use of smartphones is increasing at a fast pace and that the majority are equipped with Wi-Fi communication interfaces. This was very important for the robustness of the data collection method since it relied on this aspect to gather reliable data on crowd density. Moreover, it presented how the MAC randomization mechanism has not been widely and correctly used in nowadays' devices. This was an aspect that despite protecting one's privacy, could create noise in the crowd density data.

Furthermore, different crowd sensing methods were reviewed. It was important to perceive that crowd sensing can be done in a different number of ways and what usual challenges and solutions arise in these methods. Firstly, it demonstrated how crowd sensing is used for different purposes and the positive insights that one can gain from it. Secondly, it showed that capturing probe requests was a crowd sensing method with good results. Thirdly, it was concluded that capturing Wi-Fi probe requests is much more reliable than Bluetooth ones, helping to define the type of probe requests the sensing phase should take into account. Moreover, it showed the usual information that could be gathered from sensing smart devices, including RSSI values that, unlike other attributes, was controversial and wasn't reliable on its own. On the topic of attributes, the literature also emphasized the importance of feature engineering on the success of ML models, an aspect that was taken into account during the experiments done in ML. Last, but not least, it highlighted that using passive sensing techniques, which is the type of method used in this work and means that users don't have any kind of explicit part in the data collection, was more reliable and easier to implement when compared to other techniques.

The development of the solution itself allowed the practical understanding of the proposed approach viability. Before diving into the main practical results, it's important to highlight how the CRISP-DM methodology was important to define the main development phases and how dependent they were of each other, two aspects that helped to enhance the quality of the presented results.

When it comes to the data collection, this phase showed how it was to possible to passively capture probe requests with a cheap and practical device: ESP8266 ESP-12E NodeMCU Amica. Also, it proved that is possible to store data in a real time, useful if the sensor was applied in a real-life scenario. Furthermore, the RSSI experiment done with this crowd sensor revealed that despite having limitations when it comes to sensing in an indoor environment, it still has good capacity to do so, having proved to capture probe requests approximately up to 30 metres in the floor where it was located and even in certain areas of the floor below.

Regarding the data analysis, its main objective was to turn the collected data into reliable crowd density data. This way, the data went through a complete processing and exploration phase, which besides helping to understand the collected the data and its main parameters presented insights on crowd density and its variations. In fact, after eliminating the noise in data by replacing the number of unique MAC addresses per hour per its total number, crowd density variations became clearer. It was possible to discover how crowd density varied with different hours, months and days, variations that were plotted in different ways that helped to perceive better the number of people fluctuations. Moreover, it was possible to see the huge difference between the number of unique addresses and its total number. Besides that, the study on the collected data's MAC addresses explored the MAC vendors, the metrics of probe requests per MAC address and the MAC randomization mechanism. Mainly, it showed that on one hand, there wasn't a significative number of randomized addresses and that these addresses were, on its majority, being captured only one time. On the other hand, it demonstrated that the most represented MAC vendor was most likely from a computer, which doesn't help to estimate crowd densities as it contributes for noisy data.

As for the model conception and tuning, the LSTM model was capable of predicting the number of unique MAC addresses per hour up to 10 hours ahead and the results on the test set - which was challenging and unbiased - were encouraging. Before achieving this performance, numerous tests were done using different data configurations, different hyper-parameters and model types. These tests showed that feature engineering boosted the models' performance. On the contrary, the number of time steps used in the training data didn't have a big impact on the models' performance. Additionally, it exposed a big challenge which was dealing with the lack of representative data. For this, regularization techniques proved to help the models have a greater performance.

In conclusion, the investigation part of this dissertation helped to develop a more robust solution and also gain more knowledge in the domain of crowd sensing. In turn, the results of the implemented solution showed that it is possible not only forecast, but also to passively estimate crowd density resorting to a cheap, but qualified sensor and to crowd detection algorithms for the sensing phase.

5.2 FUTURE WORK

In the first place, a work that could be done to reduce the gap between actual crowd density and the collected data would be to remove all the probe requests that had a MAC address whose vendor was associated with a computer. To do so, a study could be done to retrieve the list of the most common vendors associated to computers, and, if these vendors didn't collide with vendors from smart devices, use this list to filter the data. This would allow the elimination of 1 person being counted as multiple persons, for example, if a person had in its possession a computer and a smartphone.

In the second place, model conception and tuning could have considered more hyper-parameters and features. Other option would be to do all the tuning in an automated manner, that is, not only using automated hyper-parameters optimization but also automated data structure optimization. With the aid of more computer power, this would allow the test of a wider range of configurations that could lead to better performances. Another aspect that would provide better predictions would be collecting more data.

Finally, the developed solution is part of the AmI paradigm, whose final phase consists in connecting the sensing and reasoning phases to the real-world. Furthermore, one of the goals of this dissertation was also to open future perspectives in applying these kind of solutions in real-life scenarios, making positive contributions to our lives. Therefore, choosing a point of interest where crowd sensing is useful would be interesting to turn these studies into real-life measures: by developing a web application that would use the conceived methods allowing users to check current and future crowd densities in a certain site; by implementing measures through the data analysis and forecasting in areas like marketing, urban security, or even the current COVID-19 situation.

BIBLIOGRAPHY

Altché, F. and d. L. Fortelle, A. An lstm network for highway trajectory prediction. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 353–359, 2018.

Ansley, C. Mac randomization in mobile devices. *Fall Technical Forum*, 2019.

Ares, A. F., Mora, A., Arenas, M. G., García-Sánchez, P., , Romero, G., Santos, V. R., Castillo, P. A., and Guervós, J. M. Studying real traffic and mobility scenarios for a smart city using a new monitoring and tracking system. *Future Generation Computer Systems*, 76, 11 2016.

Barbera, M., Epasto, A., Mei, A., Perta, V., and Stefa, J. Signals from the crowd: Uncovering social relationships through smartphone probes. pages 265–276, 10 2013.

Beresford, A. and Stajano, F. Mix zones: User privacy in location-aware services. pages 127 – 131, 04 2004.

Brownlee, J. Supervised and unsupervised machine learning algorithms, 2020a. URL https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

Brownlee, J. A gentle introduction to backpropagation through time, 2020b. URL https://machinelearningmastery.com/gentle-introduction-backpropagation-time/.

C.Augusto, J., Nakashima, H., and Aghajan, H. *Handbook of Ambient Intelligence and Smart Environments*, chapter Ambient Intelligence and Smart Environments: A State of the Art. Springer, Boston, MA, 2010.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. *CRISP-DM 1.0*. 2000.

Cook, D. J., Augusto, J. C., and Jakkula, V. R. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277—-298, 2009.

Coyne, Ed, Euphrosine, J., and Kotlyachkov, S. Firebasedemo, 2016. URL https://github.com/FirebaseExtended/firebase-arduino/tree/master/examples/FirebaseDemo_ESP8266.

Desai, R. Top 9 algorithms for a machine learning beginner, 2020. URL https://towardsdatascience.com/top-10-algorithms-for-machine-learning-beginners-149374935f3c.

E.Longo, Redondi, A., and Cesana, M. Pairing wi-fi and bluetooth mac addresses through passive packet capture. pages 1–4, 06 2018.

Ermes, M., Parkka, J., Mantyjarvi, J., and Korhonen, I. Detection of daily activities and sports with wearable sensors in controlled and uncontrolled contitions. *IEEE Transactions on Information Technology in Biomedicine*, 12:20–26, 2008.

Fernandes, B. Crowd-sensing, 2018. URL https://github.com/brunofmf/Crowd-Sensing.

Fernandes, B., Silva, F., Analide, C., and Neves, J. Crowd sensing for urban security in smart cities. *Journal of Universal Computer Science*, 24(3):302–321, 2018.

Freudiger, J. How talkative is your mobile device? pages 1–6, 06 2015.

Ganti, R., Ye, F., and Lei, H. Mobile crowd sensing: Current state and future challenges. *IEEE Communications Magazine*, 49:32–39, 11 2011.

Garbade, M. J. Understanding k-means clustering in machine learning, 2018. URL https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1.

G.Delapierre, H.Grange, B.Chambaz, and L.Destannes. Polymer-based capacitive humidity sensor: characteristics and experimental results. *Sensors and Actuators*, 4:97–104, 1983.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. The MIT Press, 2016.

Google, 2020a. URL `https://firebase.google.com/`.

Google, 2020b. URL `https://firebase.google.com/docs/database`.

Greenstein, B., Mccoy, D., Pang, J., Kohno, T., Seshan, S., and Wetherall, D. Improving wireless privacy with an identifier-free link layer protocol. pages 40–53, 01 2008.

Grokhotkov, I., 2020. URL `https://arduino-esp8266.readthedocs.io/en/2.7.4_a/esp8266wifi/readme.html`.

Grokhotkov, I., Philhower, E. F., and Gauchard, D. Esp8266wifi, 2020a. URL `https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi`.

Grokhotkov, I., Philhower, E. F., and Gauchard, D., 2020b. URL `https://github.com/esp8266/Arduino/blob/2.7.4/doc/esp8266wifi/readme.rst`.

Guo, B., Wang, Z., Yu, Z., Wang, Y., Yen, N Y., Huang, R., and Zhou, X. Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Computing Surveys*, 48, 08 2015.

Géron, A. *Hands on Machine Learning with Scikit Learn and TensorFlow*. O'Reilly Media, 2017.

Harrison, C., Eckman, B., Hamilton, R., Hartswick, P., Kalagnanam, J., J.Paraszczak, and Williams, P. Foundations for smarter cities. *IBM J. RES. DEV.*, 54(4), 2010.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9: 1735–80, 12 1997.

Holst, A. Number of smartphone users worldwide from 2016 to 2021, 2019. URL `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`.

Hyndman, R. and Khandakar, Y. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 26, 07 2008.

Josh. Everything you need to know about artificial neural networks, 2015. URL `https://medium.com/technology-invention-and-more/everything-you-need-to-know-about-artificial-neural-networks-57fac18245a1`.

Jung, J. J. and Muñoz, A. Intelligent services for smart cities. *Journal of Universal Computer Science*, 24(3):246–248, 2018.

Koehrsen, W. Feature engineering: What powers machine learning, 2018. URL `https://towardsdatascience.com/feature-engineering-what-powers-machine-learning-93ab191bcc2d`.

Laptev, N., Yosinski, J., Li, L. E., and Smyl, S. Time-series extreme event forecasting with neural networks at uber. 2017.

Larsen, J. E., Sapiezynski, P., Stopczynski, A., Mørup, M., and Theodorsen, R. Crowds, bluetooth and rock'n'roll: Understanding music festival participant behavior. 2013.

Li, L. Technology designed to combat fakes in the global supply chain. *Business Horizons*, 56:167–177, 2013.

L.Oliveira, Schneider, D., Souza, J., and W.Shen. Mobile device detection through wifi probe request analysis. *IEEE Access*, PP:1–1, 06 2019.

Martin, J., Donahue, T. Mayberryand C., Foppe, L., Brown, L., Riggins, C., Rye, E., and Brown, D. A study of mac address randomization in mobile devices and when it fails. *Proceedings on Privacy Enhancing Technologies*, 2017, 03 2017.

Massoud, S. Mac address lookup api, 2020. URL https://macvendors.co/api/.

Mitchell, Tom M. Machine learning and data mining. *Communications of the ACM*, 42 (11), 1991.

M.Murphy. Cellphones now outnumber the world's population, 2019. URL https://qz.com/1608103/there-are-now-more-cellphones-than-people-in-the-world/.

Mozer, M.C. *Smart Environments: Technology, Protocols, and Applications*, chapter Lessons from an Adaptive Home, pages 273–298. Wiley, 2004.

O'Connell, P. L. Skorea's high-tech utopia, where everything is observed, 2005. URL https://www.nytimes.com/2005/10/05/technology/techspecial/koreas-hightech-utopia-where-everything-is-observed.html.

Olah, C. Understanding lstm networks, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Petkovics, Á., Simon, V., Gódor, I., and Bence, B. Crowdsensing solutions in smart cities towards a networked society. *EAI Endorsed Transacitons on Internet of Things*, 1, 10 2015.

Project, Android Open Source. Privacy: Mac randomization, 2020. URL https://source.android.com/devices/tech/connect/wifi-mac-randomization.

Qin, E., Long, Y., Zhang, C., and Huang, L. Cloud computing and the internet of things: Technology innovation in automobile service. *LNCS*, 8017:173 – 180, 2013.

Ramos, C., Augusto, J. C., and Shapiro, D. Ambient intelligence—the next step for artificial intelligence. *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 23(2):15 – 18, 2008.

Ray, S. A quick review of machine learning algorithms. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019.

Rencberoglu, E. Fundamental techniques of feature engineering for machine learning, 2019. URL https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114.

Roman, R., P.Najera, and J.Lopez. Securing the internet of things. *IEEE Computer*, 44: 51–58, 2011.

Schauer, L., Werner, M., and Marcus, P. Estimating crowd densities and pedestrian flows using wi-fi and bluetooth. *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 171–177, 2014.

Shalev-Shwartz, S. and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

Singh, S. Cousins of artificial intelligence, 2018. URL https://towardsdatascience.com/cousins-of-artificial-intelligence-dda4edc27b55.

Stanford, V. Biosignals offer potential for direct interfaces and health monitoring. *IEEE Pervasive Computing*, 3:99–103, 2004.

Sun, Y., Song, H., J.Jara, A., and Bie, R. Internet of things and big data analytics for smart and connected communities. *IEEE Access*, 4:1–1, 01 2016.

Ting, S.L. and Ip, W.H. Combating the counterfeits with web portal technology. *Enterprise Information Systems*, 9:1–20, 2013.

Vanhoef, M., Matte, C., Cunche, M., Cardoso, L., and Piessens, F. Why mac address randomization is not enough: An analysis of wi-fi network discovery mechanisms. pages 413–424, 05 2016.

Wei, X., Zhang, L., Yang, H., Zhang, L., and Yang-Ping, Y. Machine learning for pore-water pressure time-series prediction: Application of recurrent neural networks. *Geoscience Frontiers*, 04 2020.

Weiser, Mark. The computer for the 21st century. *Scientific American*, pages 94–104, 1991.

Weppner, J. and Lukowicz, P. Bluetooth based collaborative crowd density estimation with mobile phones. pages 193–200, 03 2013.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

World Health Organization. Global status report on road safety. 2015.

Xu, L. D., He, W., and Li, S. Internet of things in industries: A survey. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 10(4), 2014.

Yu, Y., Si, X., Hu, C., and Zhang, J. A review of recurrent neural networks: Lstm cells and network architectures. *Neural Computation*, 31:1–36, 05 2019.

Yuan, Y., Qiu, C., Xi, W., and Zhao, J. Crowd density estimation using wireless sensor networks. *2011 Seventh International Conference on Mobile Ad-hoc and Sensor Networks*, 2011.

Zanella, Andrea, Bui, Nicola, Castellani, Angelo, Vangelista, Lorenzo, and Zorzi, Michele. Internet of things for smart cities. *IEEE INTERNET OF THINGS JOURNAL*, 1(1), 2014.

Zhang, Y., Chen, G., Yu, D., Yao, K., Khudanpur, S., and Glass, J. R. Highway long short-term memory rnns for distant speech recognition. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5755–5759, 2016.

Zhou, X. Research on wi-fi probe technology based on esp8266. In *2017 5th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering (ICMMCCE 2017)*. Atlantis Press, 2017/09.