POLITECNICO DI TORINO

Master Degree course in Data Science and Engineering

Master Degree Thesis

Machine Learning Regression Model For Crowd-Monitoring Through WiFi Probe-Request Analysis

Supervisors

Prof. Claudio Ettore Casetti

Prof. Paolo Giaccone

Candidate

Roon Mullaaliu

Academic Year 2023-2024

Acknowledgements

I would like to thank my supervisors, Claudio Ettore Casetti and Paolo Giaccone, for their support, availability, and professionalism that guided me through this thesis work.

I want to express my gratitude to Riccardo Rusca and Diego Gasco for their valuable advice, help, and suggestions to improve my work.

A special thanks also goes to my parents for their support and for giving me the opportunity to undertake this journey.

Finally, a special thanks goes to my girlfriend, Giada, who has always supported and encouraged me with patience and love throughout all the years of this journey.

Abstract

In current times, the proliferation of smart and IoT devices is generating vast amounts of data, presenting new challenges for leveraging this information to enhance efficiency, drive innovation, and improve decision-making. One significant application is crowd monitoring, which is becoming crucial in urban environments by improving public safety, optimizing traffic flow, and enhancing the management of events and public spaces. The analysis of wireless network traces has emerged as an effective method for real-time crowd size estimation and movement pattern analysis. WiFi probe requests have proven particularly valuable for providing these real-time insights and behaviors. The primary focus of this Thesis is on estimating the number of people by processing and analyzing captured WiFi probe request messages. Given that modern operating systems randomize MAC addresses for privacy reasons, people counting has become a challenging task. To address this, a machine learning framework has been developed and presented in this Thesis. This framework utilizes a regression model, based on sophisticated neural networks, to provide numerical estimations of people counts. All aspects of the machine learning framework implementation have been considered. This includes a valuable preprocessing phase, that filters and extracts relevant features from the captured data to use in the model. The framework also defines datasets for model training and evaluation. For proper training, vast amounts of data from a realistic software simulator and real capture data have been used, where augmentation techniques have been employed to extend the dataset size, making it suitable for machine learning training. The quality of the regression model in estimation of crowd size was initially tested on traces generated by the software simulator, showing excellent results in a synthetic environment, and demonstrating the validity of the approach. Subsequently, the framework was also tested to predict real captures. Tests were conducted with training performed first on simulated traces and then on real traces. The results were promising in both cases, particularly for the model trained with real data. The model parameters have been rigorously fine-tuned to achieve optimal performance. The most significant results were obtained using a model based on Long Short-Term Memory (LSTM). By analyzing temporally consecutive captures and identifying temporal patterns within them, the LSTM network significantly improved the results obtained by a simple Fully Connected Neural Network. The proposed regression approach incorporates the results of another people counting model, based on clustering, by including its results as features. This allowed for the acquisition and enhancement of positive results demonstrated by the clustering approach. The findings of this research have practical implications across a variety of domains. Its most crucial application lies in improving safety in public environments, pedestrian traffic management, and emergencies, by predicting congestion areas in real-time and enabling prompt action. Additionally, this Thesis addresses privacy concerns related to processing and storing WiFi probe requests, particularly the handling of MAC addresses, which are considered personal data and therefore subject to the regulations under the EU GDPR. It illustrates the problems with current solutions and presents more efficient alternatives based on advanced privacy paradigms like Bloom filters and differential privacy.

Contents

Chapter 1

Introduction

Nowadays, an increasing number of devices are connected to the global network. In our smart society the necessity of internet connectivity and the ubiquity of wireless access points (APs) have made network connections omnipresent, often running without our conscious awareness.

This proliferation of connected devices offers the opportunity to track and estimate the number of people in a specific area. Crowd monitoring has become a crucial aspect of various activities, particularly in large cities that are evolving into interconnected urban environments, often referred to as smart cities.

Effective crowd monitoring can significantly enhance the quality of life in these cities by helping authorities ensure public safety in crowded areas, anticipate potential risks, and optimize the flow of people, among other benefits. Despite the potential benefits, this can lead to privacy problems, such as the identification of individuals through their devices. Fortunately, the growing awareness on privacy topic has led to the development of numerous measures to safeguard users' privacy. The European Union has been at the forefront of these efforts, introducing the General Data Protection Regulation (GDPR) [1] in 2018. The GDPR imposes strict regulations on the collection, utilization, and sharing of personal information, with substantial penalties for non-compliance. However, this increased emphasis on user data protection and privacy has made tracking devices more challenging, complicating the implementation of systems for crowd monitoring.

The primary objective of this Thesis is to develop a reliable solution for estimating the presence of people in a specific area. Various crowd tracking methods are currently available, including computer vision systems, LiDAR, infrared sensors, and security cameras. However, these methods face significant challenges, particularly in relation to hardware requirements and privacy concerns. These challenges highlight the need for robust and effective algorithms for crowd monitoring tasks.

The main contribution of this Thesis is the development of a sophisticated regression framework for people counting, based on a neural networks model. This framework leverages the WiFi traces emitted by devices to accurately estimate the number of people present in a given area. By utilizing these WiFi traces, the proposed approach offers a viable and efficient alternative to traditional crowd tracking methods, addressing both hardware and privacy concerns. This approach is able of producing accurate crowd estimates through a rigorous machine learning methodology. It includes the implementation of efficient processing techniques for the analyzed WiFi traces and the definition of various datasets to ensure effective model training and comprehensive evaluation of the proposed framework.

A crucial aspect of this research involves addressing privacy concerns, particularly regarding the MAC address of devices. According to the GDPR, the MAC address is considered personal information as it can potentially be linked to the device's owner physical identity. Since our framework is committed to achieve full compliance with the GDPR, in the framework no personal data, including MAC addresses, are stored, thereby adhering to privacy requirements.

Moreover, as contribution for future work, this Thesis examines flow monitoring tasks that necessitate MAC address storage, analyzing current solutions available in literature. We identifies the limitations of these existing solutions and proposes modern, more robust privacy solutions based on probabilistic approaches.

Chapter 2

Background

This Chapter presents WiFi probe request and how MAC address randomization is implemented in wireless network. More in detail, it discusses the actual state of the art for crowd monitoring systems leveraging WiFi probe request analysis. A particular focus is on network analysis techniques, providing a comprehensive overview of the main tools available.

2.1 WiFi Probe Requests

When a device wants to connect to a network, it needs to know the available APs in the surrounding area. To do this it is possible to use "passive scanning", i.e., listening to beacons sent by APs to verify their presence, but it is not an efficient approach.

What is more commonly used is instead "active scanning", that consist in discover available APs sending a probe request asking which networks are available.

In Figure 2.1 can be seen a detail of the structure of a WiFi probe request defined in the IEEE 802.11 standard.

The main fields of probe requests are:

- **MAC** (Media Access Control) **Address**: A unique identifier assigned to a network interface controller (NIC) for communications on the physical network segment. It identifies the source address of the probe request.
- **Sequence Number**: A numerical value used to track the order of probe requests sent by a device. It helps in managing and organizing the probe requests received by access points.
- **SSID** (Service Set Identifier): A unique name assigned to a wireless network. Devices send probe requests containing SSIDs to discover and connect to available

networks in their vicinity. It can also be set to 0 (i.e., the SSID field is present but empty). In this case, it is reported as "Wildcard" SSID, hence looking for every SSID available.

- **VHT** (Very High Throughput) **Capabilities**: Information about the device's support for very high throughput capabilities as per the IEEE 802.11ac standard, including supported channel widths, MCS (Modulation and Coding Scheme) indexes, and spatial streams.

- **HT** (High Throughput) **Capabilities**: Details regarding the device's support for high throughput capabilities according to the IEEE 802.11n standard, such as supported channel widths, MCS indexes, and spatial streams.

- **Extended Capabilities**: Additional capabilities or features supported by the device, which may include power-saving mechanisms, spatial reuse, or other enhancements beyond basic WiFi standards.

- **Vendor Specific**: Custom information included by device manufacturers for proprietary features or functionalities. It allows vendors to add their own elements to probe requests for specific purposes, such as device identification or configuration.

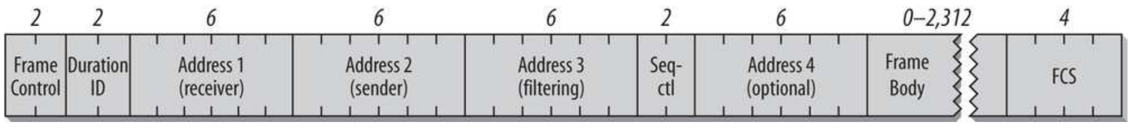


Figure 2.1: Probe request frame structure (reproduced from 2.3).

It is worth noting that probe requests are typically sent sequentially in groups of 2-3 probes, called "burst".

2.2 Capture Methodology

In computer network field, a sniffer is a device that is capable of capturing the traffic flow over a network. Beyond malicious uses, sniffers are particularly useful and used to solve network problems, to analyze and monitor network status, to verify network security, and many other tasks.

Probe requests are sent in clear (not encrypted) through radio signals, therefore anyone can "read" the actual traffic in the area with a sniffer.

2.2.1 Computer Network Sniffers

The hardware choice for building a wireless sniffer varies based on the requirements, such as cost, performance or portability. The most popular solutions are to use Raspberry Pi or Meshlium that are commercial devices available in the market. Another possible solution is to implement custom, proprietary hardware.

Rasberry Pi, shown in Figure 2.2, is a small single-board computer that is frequently used for IoT tasks, including wireless sniffing. In order to be able to sniff the traffic, it requires a USB WiFi adapter that is capable of monitor mode. It uses as Operating System *Raspberry Pi OS* (a lightweight Linux distribution) that can support various sniffing tools. It is extremely cost-effective due to its affordability and wide availability. It is flexible and can be repurposed for wide range of other applications. Moreover, due to its popularity it has extensive documentation and community support. It has performance limitations compared to more specialized hardware and the requires configuration and setup for both hardware and software.

Meshlium By Libellium [2], shown in Figure 2.3 is a specialized hardware developed by *Libelium*, designed for wireless sensor networks and IoT applications. It comes out-of-the-box with all the necessary hardware components for wireless monitoring and is ready to use with minimal setup. It uses a proprietary software, with advanced features for network sniffing. It as high performance since it is optimized for wireless sniffing with dedicated hardware, but it is significantly more expensive and less flexible.

Developing custom proprietary hardware, such as the one developed by *G-Move* (represented in Figure 2.4) for their business activity [3] allows to meet specific requirements, offering the highest level of optimization and performance. On the other hand, it requires high investment cost and complex hardware design and software development.

Figure 2.2: Rapsberry Pi device with WiFi dongle (reproduced from [4]).

Figure 2.3: Meshlium by Libellium (reproduced from [5]).

Table 2.1 provides a summary with pros and cons of each solution.

Figure 2.4: Gmove's device (reproduced from [6]).

	Rasberry Pi	***Meshlium***	***Proprietary HW***
Cost	Low cost	Medium-high cost	Very high production cost
Performance	Limited performance	High performance	Best optimization and performance
Setup	Require setup of HW and SW	Is ready-to use with minimal setup	Complex HW and SW design

Table 2.1: Comparison of hardware options for sniffer.

2.2.2 Packet Sniffing Tools

Various software tools are available to support network monitoring, each with its unique strengths and features. The most notable are:

- **Wireshark** [7]: is one of the most used tools in this domain. It is an open-source packet analysis which owes its popularity for its ability to inspect network traffic in real-time. It supports hundreds of protocols and offers comprehensive filtering capabilities. It also has the capability to save the analyzed traffic.
- **TShark** [8]: is the command-line version of Wireshark. TShark offers many of the features of Wireshark, including comprehensive protocol analysis and real-time packet capture and saving.
- **Tcpdump** [9]: is a command-line packet analyzer, known for its powerful filtering capabilities. Tcpdump allows users to display TCP/IP packets over a network. It also supports real-time monitoring and *.pcap* saving.
- **Kismet** [10]: is a network detector and packet sniffer. Its strength is passive scanning, making it capable of detecting hidden networks and capturing data without

active transmission.

- **Aircrack-ng** [11]: is a suite of tools focused on wireless network security. It includes utilities for monitoring, capturing, and cracking keys. Its component for wireless network packet capturing, is called **airodump-ng**.

2.3 MAC Address

One of the most important field inside the probe request is the MAC address. It stands for Media Access Control Address. Primarily, it is used as unique device identifier by the manufacturer. It is also a unique identifier for a network interface (i.e., an hardware component that connects a device to a network). Each network interface (e.g., WiFi, Ethernet, Bluetooth) has its own MAC address, and it is used in its relative IEEE 802 network protocols since it allows the device to be uniquely identified in a network.

MAC address is a 48 bit long identifier. It is organized in octets (a group of 8 bits also referred as Byte), for a total of 6 Bytes. The 6 octets can be separated to form two groups of 3 octets each:

- The first 3 octets form the Organizationally Unique Identifier (OUI): it uniquely identifies a vendor, manufacturer, or other organizations.
- The last 3 octets form the network interface controller: unique identifier of the network interface inside a manufacturer.

The IEEE standard provides the possibility to divide a MAC address into 2 categories:

- Universally administered addresses: the address is assigned by the manufacturer and it is considered to be globally unique.
- Locally administered addresses: the address is assigned by the network administrator, it replaces the vendor address and it is only locally unique.

This means that only universally administered addresses actually have an meaningful OUI.

The distinction between universally administered addresses and locally administered addresses can be specified by setting the 7$^{\text{th}}$ bit of the first octet to 0 as shown in Figure 2.5.

2.3.1 MAC Address Randomization

Since a global MAC address uniquely identifies a device, it can be used to identify an individual person. Moreover, the static nature of MAC addresses presents a significant

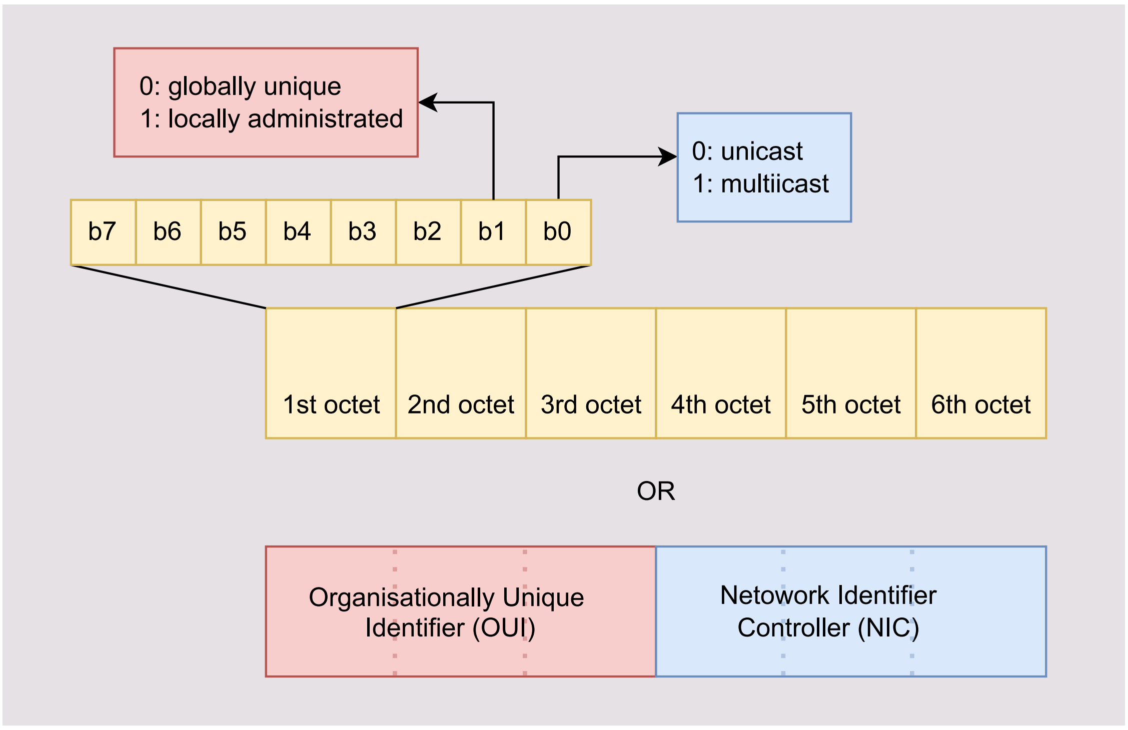


Figure 2.5: Reproduction of MAC address structure.

privacy risk, as they can be used to track a device's movements and activities across different networks. To mitigate this risk, MAC address randomization has emerged as a crucial technique, designed to enhance user privacy and security in wireless networks. Each time a device tries to connect to a network it generates a new random MAC address exposing it instead of its true network interface MAC address [12]. Locally administered addresses are used to create these random addresses.

Precisely to improve user privacy and security, major device vendors have begun to implement MAC address randomization in their OS [13], [14]. This is how it is performed in major operating systems:

- **macOS** [15] performs MAC address randomization during WiFi scanning. When a Mac searches for available networks, it uses a randomized MAC address instead of the actual address of the device.

- **Windows 10** [16]: Windows offers the option of using randomized MAC addresses when connecting to specific networks and in WiFi scanning when searching for available networks. Users can enable this feature for each WiFi network in the network settings. In Windows, MAC address randomization is not enabled by default, it requires manual configuration.

- **Linux** does not provide MAC address randomization by default, but it can be set through the use of various tools and scripts. One common tool is *NetworkManager* [17], which can be configured to randomize the MAC address during WiFi scans and even when connecting to networks.

- **Android** [18] supports MAC address randomization starting from version 6.0 (Marshmallow) [19]. During the WiFi scanning process, Android devices use randomized MAC addresses. In more recent versions (Android 8.0 and later) [20], the feature has been enhanced, and some devices also support using randomized MAC addresses when connecting to WiFi networks. This can be enabled in the WiFi advanced settings. In the latest versions of Android, MAC address randomization is enabled by default.

- **iOS** [15] implements MAC address randomization for WiFi scanning starting from iOS 8 [21]. When an iOS device scans for networks, it uses a randomized MAC address. From iOS 14 [15], Apple introduced further privacy enhancements, including the ability to use randomized MAC addresses when connecting to individual WiFi networks. This option can be enabled or disabled in the settings for each network. On iOS, MAC address randomization is enabled by default.

2.4 Crowd-monitoring

Crowd counting refers to observations and analysis of the movement, density, and behavior of groups of people within a specific area. Crowd monitoring has crucial importance in certain situations, especially in real-time scenario.

This Section presents two crowd monitoring tasks: people counting and flow monitoring; discussing how WiFi probe request analysis can be exploited to tackle the two tasks. Together, people counting and flow monitoring provide a comprehensive picture of a crowd, that can drive decision-making on creating more efficient and safe environments.

2.4.1 People-counting

People counting is a method used to track the number of individuals entering, exiting, or passing through a specific area. Beside security and safety, knowing how many people are present in a specific place at a specific time has important applications: for example, in providing business insights for retail sector; or to improve better resource allocation and management in public transport and events.

People counting can be performed through various modalities, including:

- **Video-Based Counting**: This method utilizes cameras equipped with computer vision algorithms to detect and track individuals as they enter or exit a specified area. Video-based counting systems can provide accurate and real-time data, but may raise privacy concerns depending on the deployment context.

  An example of the usage of video-based systems for people tracking is described in [22]. The paper proposes a method for counting people in crowded scenes using video analysis. It utilizes a classifier, able to recognize people's heads, based on Statistically Effective Multi-scale Block Local Binary Pattern (SEMB-LBP) features for people detection and a modified compressive tracking (CT) tracker for people tracking. The camera was installed overhead, to simplify the detection and tracking process. The experimental results showed that the system had an average accuracy rate of 86% on real video sequences.

- **Infrared sensors**: this method uses sensors to analyze the environment. Infrared sensors can detect changes in heat signatures to identify the presence of people in a particular area. Another sensor is Break-Beam Sensors: an infrared beam across an entrance that can count an entry or exit when the beam is interrupted. By using multiple beams or directional sensing, these can distinguish between entering and exiting.

  Sensors are often in proximity of doorways or entrances to count individuals as they pass through. Different sensors can be used jointly, and they are relatively simple and cost-effective, but they have in any case limitations in accurately counting large crowds or differentiating between individuals in close proximity.

  An example of the usage of infrared sensors systems for people tracking is described in [23]. The paper proposes a bi-directional people counting system using two impulse radio ultra-wideband (IR-UWB) radar sensors. The system creates two electronic layers in a corridor to detect and count people passing through, while also determining the direction of movement. The system was tested and evaluated in a real-world subway station setting and achieved a counting accuracy with errors less than 10%.

- **Manual Counting**: In some cases, people counting may still be performed manually by personnel stationed at entry or exit points. Since is not an automated approach it have high maintenance cost and is potentially prone to human error.

- **WiFi and Bluetooth Tracking**: WiFi and Bluetooth technology can be leveraged to detect mobile devices carried by individuals. By analyzing signals emitted by these devices, crowd sizes and movements can be estimated.

An example of the usage of video-based systems for people tracking is described in [24]. The paper confirms the feasibility of using WiFi and Bluetooth signals from mobile devices for crowd monitoring activities. The authors used data collected at a major German airport, comparing the results against ground truth data from boarding pass scans. Moreover, it shows that WiFi tracking provides a better approximation to crowd densities and pedestrian flows than Bluetooth tracking.

Another example is the research in [25] where the authors proposes a methodology to track people's flow in urban environments using WiFi probe request packets sent by smart devices.

WiFi tracing offers several advantages compared to other methods of people counting and tracking:

- Non-Intrusive: WiFi tracing is non-intrusive and does not require individuals to carry any special devices or interact with sensors directly. It passively collects data from WiFi signals emitted by devices already present in the area, making it less obtrusive.

- Wide Coverage: WiFi signals can cover a relatively large area, making WiFi tracing suitable for tracking people in expansive indoor environments, such as airports, shopping malls, but also for outdoor environments. This wide coverage allows for sufficient monitoring without the need for extensive sensors deployment.

- Real-Time Tracking: WiFi tracing provides real-time tracking, allowing organizations to monitor crowd movements and respond reactively to changing conditions. This is particularly valuable in dynamic environments where crowd dynamics can change fast, such as during events or emergencies.

- High Accuracy: WiFi tracing can achieve high levels of accuracy in counting individuals within a monitored area. By analyzing WiFi signal, sophisticated algorithms can differentiate between unique devices and estimate crowd sizes with precision.

- Scalability: WiFi tracing systems can scale easily. Additional access points can be deployed or existing infrastructure can be leveraged to extend coverage as needed.

- Cost-Effectiveness: Compared to traditional tracking methods that require the installation of specialized hardware or manual counting by personnel, WiFi tracing is more cost-effective to implement and maintain. It can leverages existing WiFi infrastructure reducing investment and expenses.

Overall, WiFi tracing offers a powerful and versatile approach to people counting and tracking, combining the benefits of wide coverage, real-time monitoring, accuracy, and non-intrusiveness and can be used in either indoor and outdoor scenario.

Before the implementation of MAC address randomization techniques, to count the number of devices through WiFi tracing, it was only necessary to count the distinct MAC addresses within a capture, since each device always communicated the same MAC address, that uniquely identified it. This basic Algorithm is summarized in 1. With the spread of MAC address randomization this is no longer valid. Because the same device communicates more than one MAC address.

Algorithm 1 Counting Distinct MAC Addresses

```
Require: macList: List of all MAC addresses captured
Ensure: distinctCount: Number of distinct MAC addresses
1: distinctCount $\leftarrow$ 0
2: macSet $\leftarrow$ empty set
3: for mac $\in$ macList do
4:     if mac is not in macSet then
5:         Add mac to macSet
6:         distinctCount $\leftarrow$ distinctCount + 1
7:     end if
8: end for
9: return distinctCount
```

In the next Chapters we therefore discuss how to use WiFi probes request to perform people counting, taking into account mac randomization.

2.4.2 Flow-monitoring

People counting is a useful and powerful tool for many applications. However, while counting people provides a snapshot of how many individuals are present at any given time, it lacks the dynamic context of how people move and interact within the space. This is the reason why flow monitoring is either an important task.

Flow monitoring is the practice of tracking the paths and movements of people within a defined area. It gives deeper insights into users behaviors and helps in understanding the volume, patterns, duration, and destinations of people movements. This information is crucial for several reasons and extents people counting applications. For example for safety reason: by punctually identifying potential congestion points authorities can optimize emergency evacuation routes and decide in which direction to focus the rescue.

It can also improve operational efficiency in retail environments by optimizing staff deployment, improving customer experiences and retrieving marketing information. In urban planning, it can lead in the design of more effective transportation systems and public spaces in relation to actual usage patterns.

WiFi tracing is an effective tool for flow monitoring, it keeps the same advantages described in 2.4.1.

As for the people counting task, with the use of a sniffer, WiFi probe request from devices within the monitored area are captured. By using captures from multiple monitoring devices, a system could track the position of scanned devices by processing and analysing those captures. Allowing for the monitoring of movement patterns over time, such as common pathways or areas of congestion.

An example of a solution for flow monitoring through the analysis of WiFi probe requests is presented in the research conducted by [13] . This study analyzed data collected from six sensors over a period spanning from October 2019 to April 2020. These sensors captured and stored WiFi probe requests in the city of Turin, specifically in the area included from the campus of Politecnico di Torino to Porta Susa train station. Each sensor recorded the hashed MAC address and the timestamp of each captured probe request. With this methodology, the researchers were able to infer valuable insights about pedestrian flow dynamics and density. Movement patterns were deduced by comparing the hashed MAC addresses across different sensors. When the same hashed MAC address was detected by multiple sensors, analyzing the timestamps of these detections, were possible to calculate movement directions and speeds. Using this methodology, the authors identified preferred traffic directions, in relation with weekday and hour. Additionally, by examining the time intervals between detections, they were able to hypothesize the means of transport used by the device owners. This comprehensive analysis allowed for a detailed understanding of flow dynamics in the monitored area.

2.5 Privacy Concerns

Both people counting and flow monitoring have significant privacy concerns. They handle MAC addresses in probe requests, and as described in 2.3, a MAC address can be uniquely linked to a device and consequently to an individual. Therefore, without taking adequate countermeasures it is possible to retrieve the position and movements of individuals at a specific time.

Privacy is not only a crucial aspect of the user experience, ensuring that individuals feel secure and respected when using services, but it is also subject to legal regulations. Any implementation involving personal data must adhere to local laws and regulations. Compliance with these laws is essential. This Section presents the current regulations regarding data privacy protection and examines how they relate to our research.

2.5.1 GDPR Regulation

Current technological progress in data analyses, has resulted in the increase of data collection, processing and storage. As personal data has increased their importance, the protection of this data has emerged as a critical concern.

The GDPR [1], which came into effect on May 25th 2018, represents the will of the European Union to address these concerns and establish robust data protection standards. The GDPR was introduced to consolidate data privacy laws across Europe, ensuring that all EU citizens enjoy consistent and comprehensive protection of their personal information.

It imposes stringent obligations on organizations that handle personal data, requiring them to implement strong data protection measures and maintain transparency by emphasizing accountability and security. The GDPR applies not only to data collected directly from individuals but also to data collected indirectly or inferred from other sources. The GDPR applies to any organization that processes personal data and monitor the behavior of individuals within the EU.

The GDPR is the most robust and comprehensive data protection policies globally, setting an exemplary standard for individual privacy rights. It is the de facto standard for privacy regulation.

2.5.2 Personal Data

The GDPR is applicable whenever the data being processed can be classified as personal data. This means that if the information being handled relates to an identifiable individual, the regulations of the GDPR must be followed.

The GDPR covers a wide range of personal data, including names, contact information, identification numbers, location data, online identifiers, and factors specific to the physical, physiological, genetic, mental, economic, cultural, or social identity of a person. In the GDPR, personal data is broadly defined as any information relating to an identifiable natural person. This definition covers a wide range of data, including:

- Basic Identifiers: Names, identification numbers, location data.
- Contact Information: Email addresses, postal addresses, phone numbers, and usernames.
- Online Identifiers: IP addresses, MAC addresses, cookies, device IDs, and other unique online identifiers.
- Special Categories of Data (Sensitive Data): Data revealing racial, political, religious, philosophical, genetic, biometric, health, or sexual orientation information.

- Pseudonymized Data: Personal data that has been processed in a way that it is no longer directly linked to a specific individual without additional information.
- Any Information Facilitating Indirect Identification: Data that, when combined with other information, can lead to the identification of an individual.

GDPR's definition of personal data is intentionally very broad to ensure that individuals' privacy is protected comprehensively.

Chapter 3

Machine Learning Techniques

Machine learning (ML) is a branch of artificial intelligence (AI). It is a field that focuses on developing algorithms that enable computers to learn and make decisions based on provided data. In traditional programming, explicit instructions and logic are provided to perform a task. Instead, machine learning allows systems to improve their performance over time as they are exposed to data. The origins of machine learning can be date to the mid-20th century, but in recent years has increased its popularity due to the exponential growth of computational power, to the availability of vast amounts of data in nowadays smart-society, and to the development of sophisticated algorithms. Machine learning has had a profound impact across various sectors, revolutionizing the way to approach to problem-solving and decision-making. Thanks to the capacity to learn from data and to adapt to them, machine learning has driven significant advancements in numerous fields.

At its core, machine learning involves the creation of models that can identify patterns and make predictions or decisions without human intervention. A machine learning model consists of a set of parameters that are refined during training to make accurate predictions. Machine learning models learn from data through a process of adjusting their internal parameters to minimize prediction error based on the ideal expected output. For each input in the data, the model generates a predicted output based on its current parameters. The model's output is compared to the expected output using a loss function, which quantifies the error and the model parameters are then updated to reduce the error. Figure 3.1 schematize this process.

Machine learning models are designed to minimize error on known data, and their true quality is assessed by evaluating their performance on new, unseen data. Although a model can perform exceptionally well on the training set, this does not guarantee that it will perform equally well on new data. This phenomenon is known as *overfitting*, it occurs when the model learns the noise and specific details of the training data rather

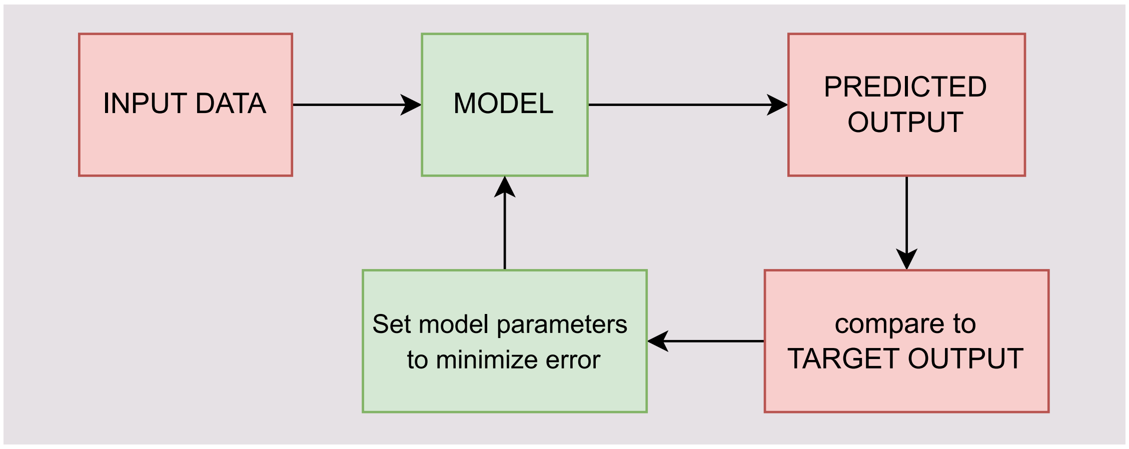


Figure 3.1: Machine learning general procedure.

than the underlying patterns. Overfitting leads to poor performance on new data.

This is why datasets are typically divided into a training set and a test set. The model learns from the training set and then its performance is evaluated using the test set to ensure than it generalizes well to new data.

Based on the task and output type, machine learning models are built using different types of learning methods:

- **Supervised Learning**: The model is trained on labeled data, where the input-output pairs are known. The model learns to map inputs to the correct outputs by minimizing errors in its predictions. Tasks that fall within supervised learning are *regression* (predicting continuous values) and *classification* (predicting discrete labels). An example of regression is predicting car prices, instead an example of classification is distinguish emails as spam or not.

- **Unsupervised Learning**: The model is trained on unlabeled data and must identify patterns or structures within the data. Tasks that fall within unsupervised learning are *clustering* (grouping similar instances together) and *association* (finding relationships between variables). An example of clustering is customer segmentation, instead an example of association is market basket analysis.

- **Reinforcement Learning**: The model learns by interacting with an environment and receiving feedback in the form of rewards or penalties. An example of reinforcement learning is robot training.

- **Semi-Supervised and Self-Supervised Learning**: These approaches mix supervised and unsupervised learning. Semi-supervised learning uses a small amount of labeled data together with a large amount of unlabeled data, while self-supervised learning used the data itself to generate labels, reducing the need for manual annotation.

The choice of algorithm depends on the specific task and the nature of the data.

3.1 Unsupervised Learning

Unsupervised learning is a type of machine learning in which the model is trained on data without explicit labels or pre-defined outcomes. Differently than supervised learning, unsupervised learning does not learn relationship between inputs and outputs, but focuses on identifying patterns, structures, and relationships within the whole data itself. This means that different datasets can produce various outcomes, as the results depend heavily on the specific data and the unsupervised learning algorithm used.

Since there are no labels to validate against, typically there is no reason to apply the traditional train-test split, and the entire dataset is used for the learning process. This allows the model to access all available information to maximize the potential learn.

Although a test set cannot be used for evaluation, other methods can be used to assess the effectiveness of the model. Techniques such as cross-validation, or domain-specific validation can help determine the quality of the discovered patterns. Evaluation metrics can differ much from each other and they could not always be meaningful.

3.1.1 Clustering

Clustering is a unsupervised machine learning technique used to group a set of objects in different groups (called *clusters*) in such a way that objects in the same cluster are more similar to each other than to those in other clusters. In Figure 3.2 is provided a simple example of how clustering works.

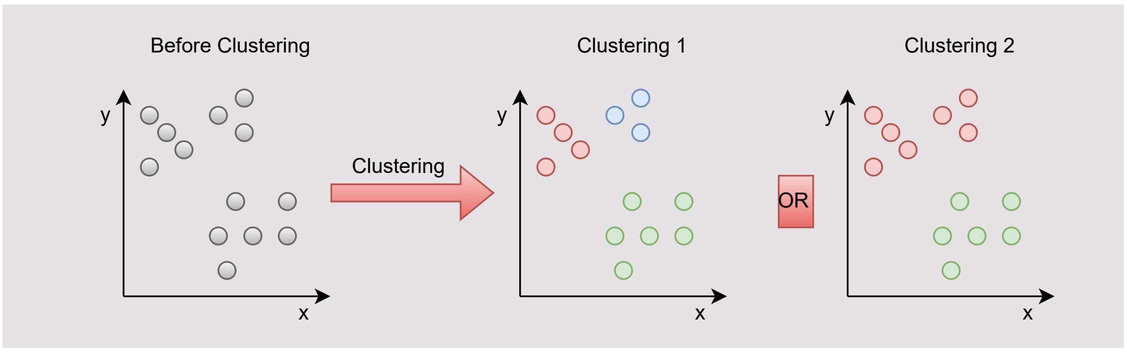


Figure 3.2: Example of labeling data points with clustering.

To measure the similarity or dissimilarity between data points is defined a *distance metric*. A distance metric is a function that measures the distance between two points. Common metrics are Euclidean distance, Manhattan distance, and cosine similarity.

Since there is no ground truth in unsupervised learning, to evaluate clustering results other methods are used. Common methods are *silhouette score* which measures how similar a point is to its own cluster compared to other clusters, and *Davies-Bouldin Index* which evaluates the average similarity ratio of each cluster with the one most similar to it.

Many clustering algorithms are available, each designed to handle different types of data and requirements. The main distinction about clustering algorithms is based on whether they require a prior knowledge of the number of clusters (K) or not. Some algorithms, like *k-means*, require to provide the number of desired clusters as a parameter; while others, such as DBSCAN, do not require this information and can discover the total number of clusters based on the data.

K-means

The k-means algorithm is a popular clustering technique. It aims to partition data points into k distinct, clusters. Each cluster is defined by its centroid, which is the mean position of the points within that cluster.

To apply k-means, is needed to decide the number of clusters to create, denoted as k. Once k is defined, the algorithm begins by randomly selecting k points from the dataset to serve as the initial centroids. These centroids represent the starting centers of the clusters.

With the initial centroids, the algorithm proceeds to the assignment phase. Here, each data point in the dataset is assigned to the nearest centroid, based on a defined distance metric. After the assignment phase, the algorithm moves to the update phase. In this step, the centroids are recalculated. The new centroid is the average position of all the points in the cluster.

These two steps, assignment and update, are repeated iteratively until the centroids stabilize, meaning they no longer change significantly between iterations. This indicates that the algorithm has converged, and the final clusters are defined by the last centroids and the points associated with them. k-means is valued for its simplicity and efficiency, making it suitable for large datasets.

The algorithm is guaranteed to converge, but it is not guaranteed to find the optimal clustering since the converged solution depends on the initial placement of centroids. Moreover, it may not perform well with clusters of varying shapes and sizes or with outliers.

The most critical point is that the number of clusters, k, must be specified in advance, which can be challenging.

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that can identify clusters of varying shapes and sizes in data containing outliers. DBSCAN does not require specifying the number of clusters. Instead, it relies on two parameters ϵ and *MinPts*. ϵ defines the radius, in term of distance metric, of the neighborhood around a point. It determines how close points need to be to each other to be considered part of the same cluster. *MinPts* defines the minimum number of points required, within the ϵ-neighborhood of a point, to form a cluster.

In DBSCAN each data point can be classified as Core Points, Border Points, and Noise. A point is considered a core point if at least *MinPts* points are within its ϵ-neighborhood (including itself). A point is a Border Points if it is within the ϵ-neighborhood of a core point but has fewer than *MinPts* points in its own ϵ-neighborhood. A Point that is neither a core point nor a border point is Noise Points, i.e., an outlier.

Since its introduction, DBSCAN has raised considerable popularity: it is robust to noise data and outliers and can identify clusters of arbitrary shapes and sizes, not just geometric shapes.

The results of DBSCAN can be sensitive to the choice of its parameters ϵ and $MinPts$. For this reason it is necessary to choose their values reasonably. DBSCAN performs well when clusters have similar densities, but it may have problem with clusters of significantly different densities.

OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure) is an advanced clustering algorithm that extends DBSCAN to address its limitations, particularly with varying densities. Like DBSCAN, OPTICS has the same parameters (ϵ and MinPts) and identifies clusters based on density, but OPTICS generates an ordering of the dataset, that captures the clustering structure at different density levels.

OPTICS processes each point in the dataset, producing a reachability plot, which visualizes the structure of the clusters. Clusters are identified by valleys in the reachability plot, where lower values indicate denser regions.

OPTICS is particularly useful when dealing with datasets with varying densities and provides more detailed insights into the data structure through the reachability plot.

3.2 Supervised Learning

In supervised learning, each training sample consists of input features (i.e., variables) and an associated output label or target variable. The goal of machine learning methods is to learn a mapping from input features to output labels, enabling the algorithm to make predictions on new data. Features can be either *categorical*, meaning they can have discrete category values, or *quantitative*, meaning they represent quantities that can be expressed as numerical values.

The training procedure typically follows the following process: the dataset is randomly split into a training set (typically 70-80% of the data) and a test set (typically 20-30% of the data). In some cases, a part of the training set is split into a validation subset. The validation set is used to tune hyperparameters of the algorithm and make decisions about the model architecture without affecting the test set. The data split procedure is resumed in Figure 3.3

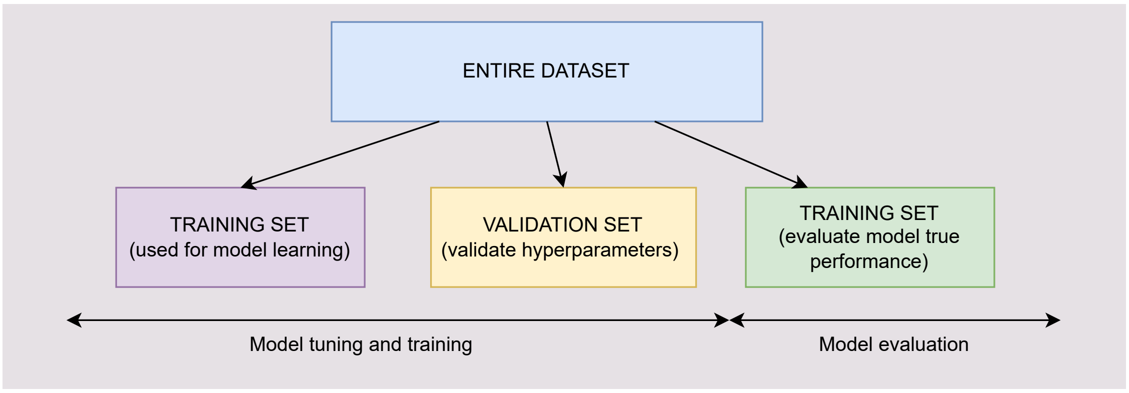

ENTIRE DATASET
TRAINING SET (used for model learning)
VALIDATION SET (validate hyperparameters)
TRAINING SET (evaluate model true performance)
Model tuning and training
Model evaluation

Figure 3.3: Dataset splitting procedure for definition of train-test set.

The model is trained using the training set data to learn the relationship between the input features and the output labels. After training, the model is evaluated on the test set using a performance metric.

3.2.1 Classification

Classification is a supervised learning task in which the goal is to categorize input data points into one of several predefined classes or categories. Labels are the distinct categories into which the data points are classified.

Some examples of classification are email spam detection, image recognition (classifying images into different categories, e.g., if it contains a cat, a dog, or a person), and medical diagnosis (predicting if a patient has a particular disease).

Numerous classification algorithms have been developed. Some of the more popular are: Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVM), and Neural Networks. Each of these algorithms have unique characteristics and parameters.

The chosen model, trained to predict the class label of new data points, is called a *classifier*. While the train set is used to train the classifier, the test set is used to evaluate it. If the classifier predicts a test point to a specific category, it is said to be a *positive prediction* for that category. If the prediction was correct (i.e., equals to the ground truth) it is also said to be a *true positive prediction*. The quality of a classifier can be asserted using different evaluation metric:

- *Accuracy*: the proportion of true positive predictions out of all predictions.
- *Precision* (c): it refers to a specific category (c); is the proportion of true positive predictions out of all positive predictions for that category.
- *Recall* (c): it also refers to a specific category; is the proportion of true positive predictions out of all actual positives for that category.
- *F1 Score*: the harmonic mean of precision and recall, providing a balance between the two.

3.2.2 Regression

Regression is a supervised learning task with the goal to predict continuous numerical values based on input features. Some simple examples of regression are: forecasting stock prices based on historical data and market indicators, or estimating the temperature based on factors like humidity, wind speed, and time of day.

Numerous algorithms are available for regression. Some of the most popular are: Linear Regression, Polynomial Regression, Decision Trees, Random Forest, Support Vector Machines and Neural Networks.

Since dealing with numerical value instead of categories, the evaluation metrics for regression are different from those of classification. The most relevant are:

- *Mean Absolute Error* (MAE): The average absolute difference between predicted and true values.
- *Mean Squared Error* (MSE): The average squared difference between predicted and true values.
- *Root Mean Squared Error* (RMSE): The square root of the MSE, providing a measure of the average magnitude of the error.

3.2.3 Models For Supervised Tasks

This Section presents various models used in supervised tasks, such as classification and regression. It includes a discussion on the most popular ones: random forest, support vector machines, and neural networks, highlighting their applications, advantages, and limitations.

Random Forest

Random Forest is a versatile and widely used machine learning method for unsupervised tasks. Random Forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

A decision tree is a tree-like structure where internal nodes represent tests on features, branches represent the outcomes of these tests, and leaf nodes represent class labels (for classification) or continuous values (for regression). This algorithm builds the tree by splitting the training data based on feature values, and predictions are made by following the path from the root to a leaf, using the label or value of the leaf for the final prediction. Decision trees are interpretable and can handle non-linear relationships but they often suffer from overfitting and instability.

From the original dataset, multiple subsets are created using a technique called bootstrapping: a random sampling with replacement. For each subset, a decision tree is built, that use a random subset of features at each split, which helps to reduce overfitting and improve generalization.

For classification tasks, each tree votes for a class, and the class with the most votes is the final prediction. For regression tasks, the average of the predictions from all the trees is taken as the final prediction.

Random Forest is a powerful and flexible machine learning algorithm that by combining multiple decision trees improves predictive accuracy, robustness to noisy data and reduce the risk of overfitting.

Support Vector Machine

Support Vector Machines (SVMs) are versatile supervised learning models used for classification, regression, and outlier detection. SVMs find the optimal hyperplane that separates data points with different labels. Support Vectors, the data points closest to the hyperplane, influence the hyperplane position and only those points are used to define the hyperplane, making SVMs memory efficient. The goal of SVM is to maximize the margin of the hyperplane: the distance between the hyperplane and support vectors. SVMs can handle non-linear hyperplanes using kernel functions, transforming input space into higher dimensions where linear separation becomes feasible. SVM is particularly effective in complex data scenarios with high-dimensional spaces.

Neural Network

Neural networks are inspired by the human brain's structure and function. These networks consist of interconnected nodes, also called **neurons**, organized in layers, which can process and learn from data.

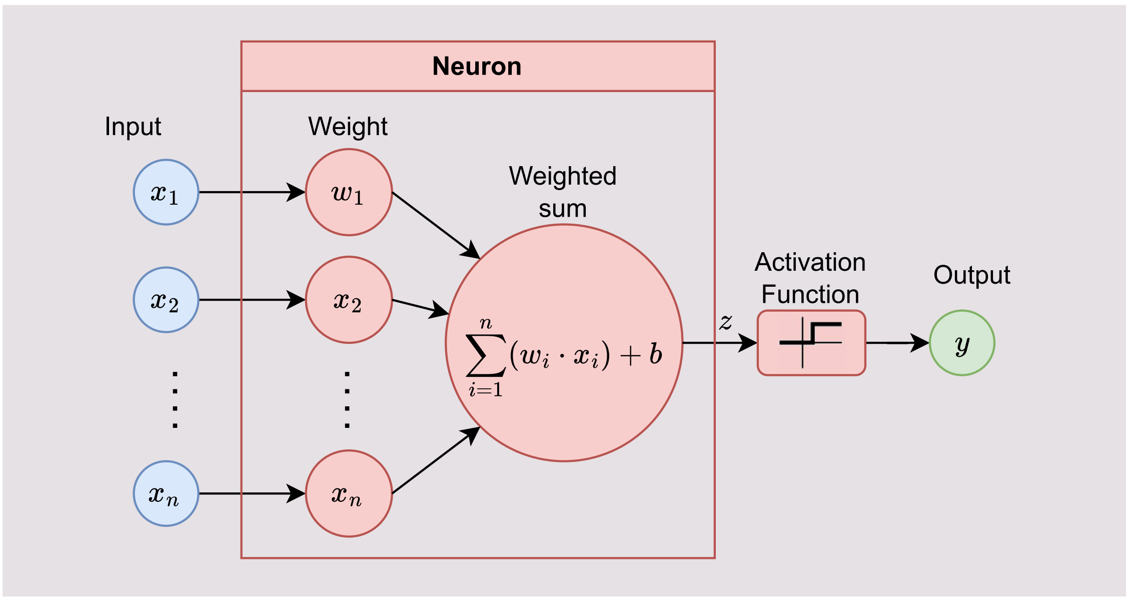


Figure 3.4: Reproduction of the structure of a Neuron.

A neuron (Figure 3.4) is composed of several components:

- *Inputs* (x): is a vector of values of a fixed length. It is received from the previous layer or from the input data.
- *Weights* (w): a neuron has a weight associated to each input value. It determines the importance of that input.
- *Bias* (b): In addition to the weights, a neuron has a bias term. The bias allows the neuron to shift the activation function, providing additional flexibility in learning.
- *Weighted Sum* (z): The neuron computes a weighted sum of its inputs and the bias:

$$z = \sum_{i=1}^{n}(w_i \cdot x_i) + b$$

- *Activation Function* (ϕ): The weighted sum z is passed through an activation function, which introduces non-linearity into the model. The output of the activation function is the neuron's output, $y = \phi(z)$. Activation functions introduce non-linearity that permit to model complex, non-linear relationships in data. Without activation functions only linear transformations would be performed on the inputs, limiting the ability to capture complex patterns. Furthermore, activation can functions regulate the output of neurons to a specific range. This is useful in various contexts. Popular activation functions (represented in Figure 3.5) are *sigmoid*, *tanh*, *ReLu*.

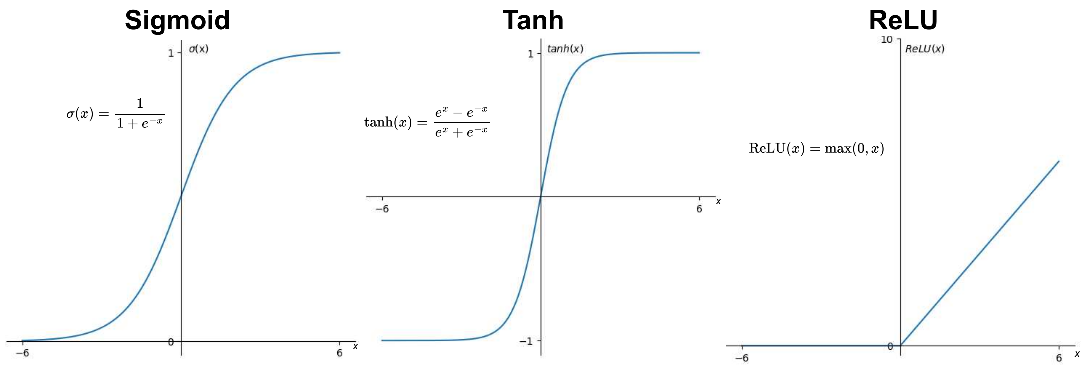


Figure 3.5: Popular activation function.

In neural network, neurons are organized in layers. There are many type of layer from simple *Fully Connected* (or linear) layers (Figure 3.6) to more advanced layers like convolutional layers, recurrent layers, and others. The layers are interconnected to form more complex networks (Figure 3.7). The layer connected to the input is called *input layer*, the layer producing the output is called *output layer*, while the intermediate layers are called *hidden layers*.

The weights and bias of all neurons of the network are parameters that the network updates during training, based on the errors of its predictions compared to actual outcomes. Effective learning occurs when their value is optimal for error minimization.

This process enables neural networks to recognize patterns, make predictions, and even generate new content. This is the reason why neural network outperform in many fields: from image and speech recognition to natural language processing and autonomous systems.

The training of neural networks is based on several key concepts. **Backpropagation** is the core algorithm used to train neural networks. It involves two main phases. The *forward pass* where the input data is passed through the network, layer by layer, until the

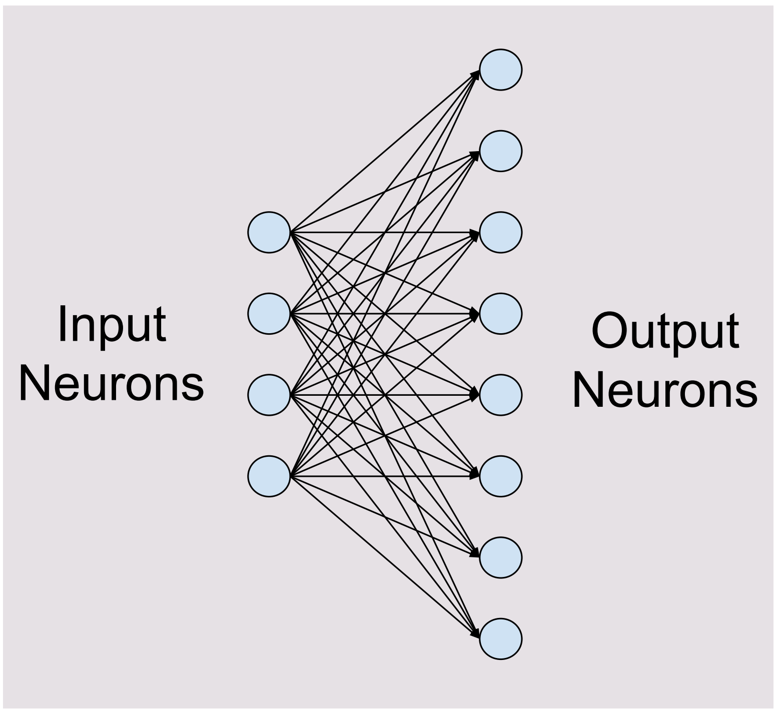


Figure 3.6: An example of a fully connected layer.

output is generated; and the *backward pass* where the error between the predicted output and the actual target is calculated using a loss function. This error is then propagated backward through the network, from the output layer to the input layer, to update the weights.

The **gradient** represents the rate of change of the loss function with respect to the weights of the network. The backpropagation algorithm calculates the gradient of the loss function with respect to each weight. The gradients are then used to update the weights to reduce the error: the direction of the gradient indicates how the weights should be adjusted to reduce the loss.

The weights are updated with the use of an **optimizer**. An optimizer is an algorithm that updates the weights of the neural network based on the gradients. Optimizer perform these updates efficiently and effectively. Different optimizers are available. Common optimizers are *Stochastic Gradient Descent* (SGD) and *Adam* (Adaptive Moment Estimation).

Training is typically done with batches of data. A **batch** refers to a subset of a fixed size (the *batch size*) of the training data, passed together through the network to

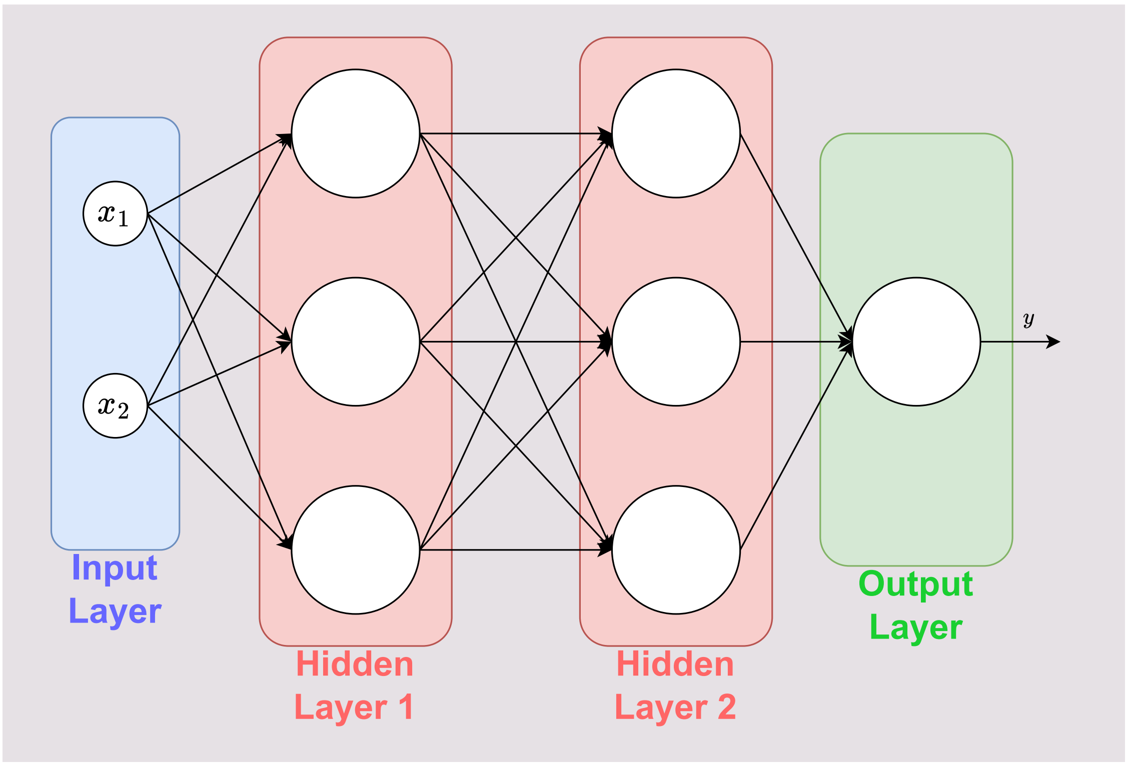


Figure 3.7: Example of complex network with many sequential fully connected layer. The network has input size = 2, two hidden layer with size = 3 and output size = 1.

compute the gradient and update the weights. Using batches allows for faster training. This approach is called *Mini-Batch Learning*. If the batch size is equal to the training set size (i.e., the entire dataset is used to compute the gradient and update the weights), it is called *Batch Learning*.

An **epoch** is defined as a complete pass through the entire training dataset. In neural networks, training is typically done on multiple epochs which allow the network to learn deeply, analyzing the training data multiple times.

The update of the weights is also defined by the **learning rate**, that is an hyperparameter that controls the step size of the update. A higher learning rate can lead to faster convergence but may cause the model to overtake the optimal solution. In contrast, a lower learning rate results in slower convergence but can lead to higher accuracy. For this reason, what is typically done is regulate the learning rate through a decay schedule: the learning rate decay is higher at the beginning and decreases over epochs. This approach improves convergence stability to the optimal solution. The entire training procedure is resumed in Algorithm 2

Computational power is a key factor in neural network: the more complex the network

Algorithm 2 Neural Network Training Process

```
1:  Initialize weights $W$ randomly
2:  Set learning rate $\eta$
3:  for each epoch do
4:      for each batch $B$ in training data do
5:          Forward Pass:
6:              Compute predicted output $\hat{y}$ for inputs $X$ in batch $B$
7:          Compute Loss:
8:              Calculate error $L(\hat{y}, y)$ using the loss function
9:          Backward Pass:
10:             Compute gradients $\nabla_W L$ of the loss with respect to weights $W$
11:         Update Weights (Optimization):
12:             Update weights $W \leftarrow W - \eta \nabla_W L$
13:     end for
14: end for
```

is, the more computations are required to perform backpropagation and weights update. Deep neural networks are characterized by multiple layers and have achieved exceptional accuracy in multiple tasks. This progress is attributed to increased computational power, the availability of large datasets, and advancements in algorithmic techniques.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural networks able to recognize patterns in sequences of data, such as time series, spoken language, or written text. They are powerful machine learning models, designed for tasks where data points are dependent on each other and the sequence in which data points appear is important.

The key feature of RNNs is their ability to maintain a "memory" of previous inputs, through their unique architectural network structure with internal loops. This is achieved by using **hidden states** that get updated at each step of the sequence, allowing information to persist over time. An illustration of the basic structure of a RNN is shown in Figure 3.8.

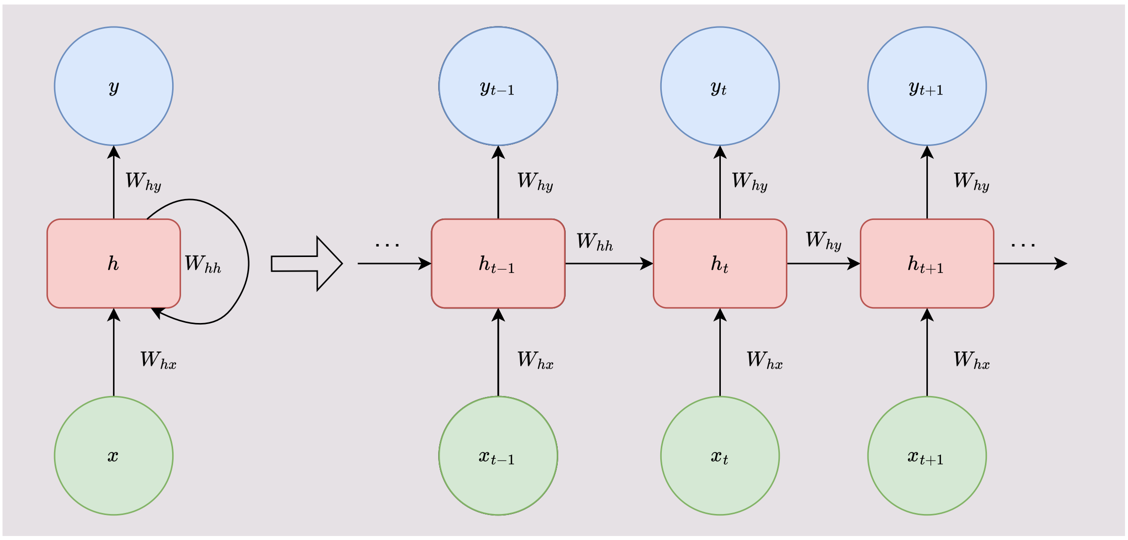


Figure 3.8: Representation of the loop structure of a basic Recurrent Neural Network.

The hidden state is the core of an RNN, which keeps information from the previous time steps. At each time step t, the hidden state h_t is updated based on the input x_t and the previous hidden state h_{t-1}.

Mathematically, this can be represented as: $h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$, where

- f is an activation function (such as tanh or ReLU).
- W_{hx} is the weight matrix for the input.

- W_{hh} is the weight matrix for the hidden state.
- b_h is a bias term.

For each time step an output y_t is produced using the current hidden state:

$$y_t = g(W_{hy}h_t + b_y)$$

where g is another activation function, used to map the output in the correct domain of the interested task; and W_{hy} is the weight matrix for the output.

In the last few years, variety of RNNs architectures have been developed. Among these, one of the most widely utilized variants is the Long-Short-Term Memory (LSTM) network.

LSTM networks are particularly popular due to their ability to retain long-range dependencies in sequential data, addressing the vanishing gradient problem that often limits the performance of standard RNNs. This makes LSTMs highly effective for data series. An LSTM network consists of a series of LSTM cells, each of which processes an input sequence one step at a time. Each LSTM cell, beside the classic hidden state h_t, has an additional state: the cell state c_t.

The output of the LSTM cell is the hidden state h_t itself and a part from the input size, the only hyperparameter of a LSTM cell is the size of the hidden state.

In the cell, those states are used in three special gates (input, forget, and output gates) to regulate the flow of information and they are particularly effective in capturing long-term dependencies.

- **Forget Gate**: Decides what information from the previous cell state should be discarded. It takes the current input (x_t) and the previous hidden state (h_{t-1}) and outputs a vector F_t with values between 0 and 1. F_t is used for an element-wise multiplication ($\otimes$) with the previous the cell state c_{t-1} to module its contribution. A value close to 1 will keep information of an element in the cell state, while a value close to 0 will cancel that information.
- **Input Gate**: Determines what new information will be stored in the cell state c_t. Form x_t and h_{t-1}: before a *tanh* function create a vector $\tilde{c}_t$; then a *sigmoid* function produce a vector I_t, that decide, through an element-wise multiplication with $\tilde{c}_t$, which values will be added to update the cell state.
- **Output Gate**: Determines the output ($y_t = h_t$) by updating the hidden state h_t based on the cell state c_t passed through a *tanh* function and multiplied with a vector O_t (a *sigmoid* function form x_t and h_{t-1}).

The structure and the operation of a LSTM cell are resumed in Figure 3.9.

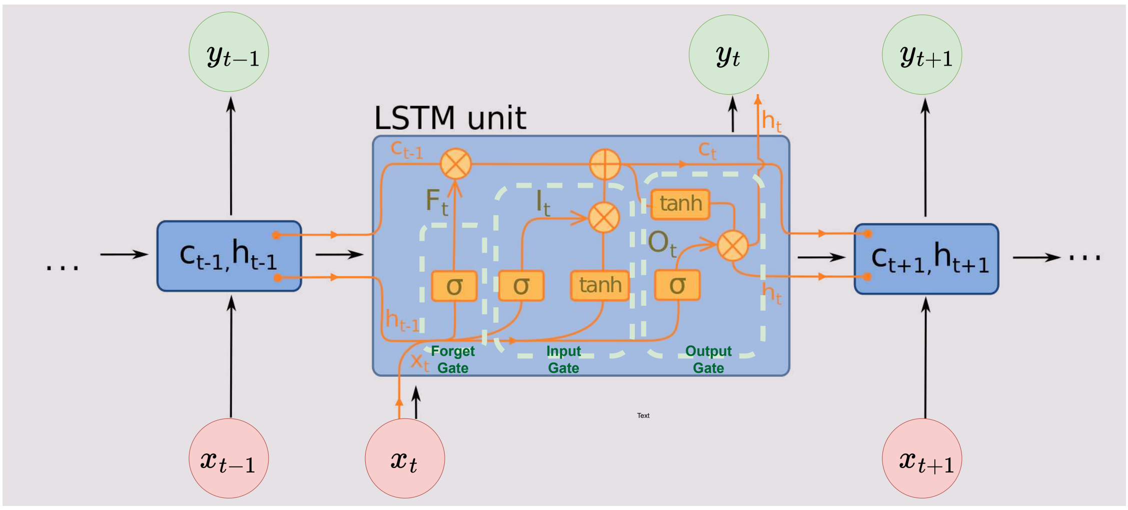


Figure 3.9: Representation of the structure of a LSTM cell.

3.3 ML Techniques For People Counting

This Section discusses the actual state of the art for people counting systems based on machine learning techniques. It presents how classification and clustering models leverage WiFi probe requests to provide prediction about the number of people in an area.

3.3.1 Classification

People counting through analysis of WiFi probe requests can be performed using a classification system. In this system, each packet is treated as a data point, and the class labels are the nominative identifier of a device model (e.g., iPhone 7, Samsung S10, etc.).

During preprocessing, features are extracted from each packet to be used as input for the classifier. The classifier is trained with appropriate data from packets generated by different devices of all classes.

Once the classifier has been trained, it can classify the packets from a test capture. At the end of the classification process, the estimated number of devices in the capture can be determined by counting the number of distinct device classes predicted by the classifier. This type of process is described in Figure 3.10.

In [26], the authors present a method for identifying devices in a WiFi network based on the information contained in probe request frames. They used a neural network to create features to be used as device fingerprint. Although the primary motivation for their research is related to security (focusing on device identification attacks and potential defense mechanisms) rather than crowd monitoring: their work demonstrates that a probe request classification system can achieve excellent performance.

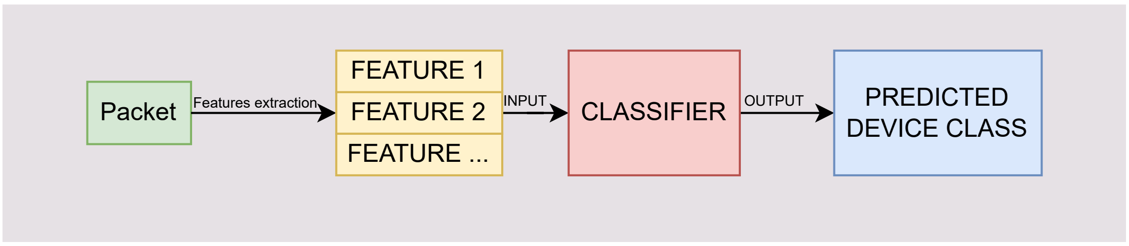


Figure 3.10: Classification based on probe request.

The authors developed a deep learning-based device identification method that automatically extracts features from all the preprocessed probe request fields. This method uses a Multilayer Perceptron neural network, purely consisting of a sequence of fully connected layers. The network architecture includes three hidden layers, each with 128 neurons.

The model is trained on a dataset of probe requests collected from 20 different types of WiFi devices and achieves an average F1 score of over 99%. Their work proves that an approach based on classification can achieve very high accuracy. Unfortunately, this is only feasible in a controlled scenario with a limited and known set of devices.

The main issue with a classification-based solution is that it cannot be applied in a public space scenario, because in perform classification, the labeled devices must be known in advantage. Given the vast and continuously growing number of wireless device models, it would be necessary to analyze every possible device model currently in existence. Adding a new class to the classifier every time a new model is released, which happens essentially daily. This is in practice infeasible and leads to underperformance, because increasing the number of labels to such a large set significantly reduces the model's accuracy.

Furthermore, distinct devices of the same model cannot be distinguished from one another, making it impossible to correctly count the devices separately.

3.3.2 Clustering

Due to MAC address randomization, for estimating different devices in a capture is not sufficient to count the number of distinct MAC addresses within the capture. However, if a method could identify which probe requests were sent by the same device and group these probes accordingly, counting these groups would provide an estimate of the number of devices present. This concept has been explored in many research studies.

The main goal of this type of technique is to perform a clustering on the WiFi probe requests, in order to achieve the desired result of having groups of probes sent by the same device.

Packets in a burst have the same content, including the same MAC address. Each burst is considered a data point. A preprocessing step is performed to the capture, and features are extracted from each packet based on the probe request content fields. This process maps the captured data into a feature space, allowing the application of a clustering algorithm. The basic mechanism of a clustering approach is described in Figure 3.11.

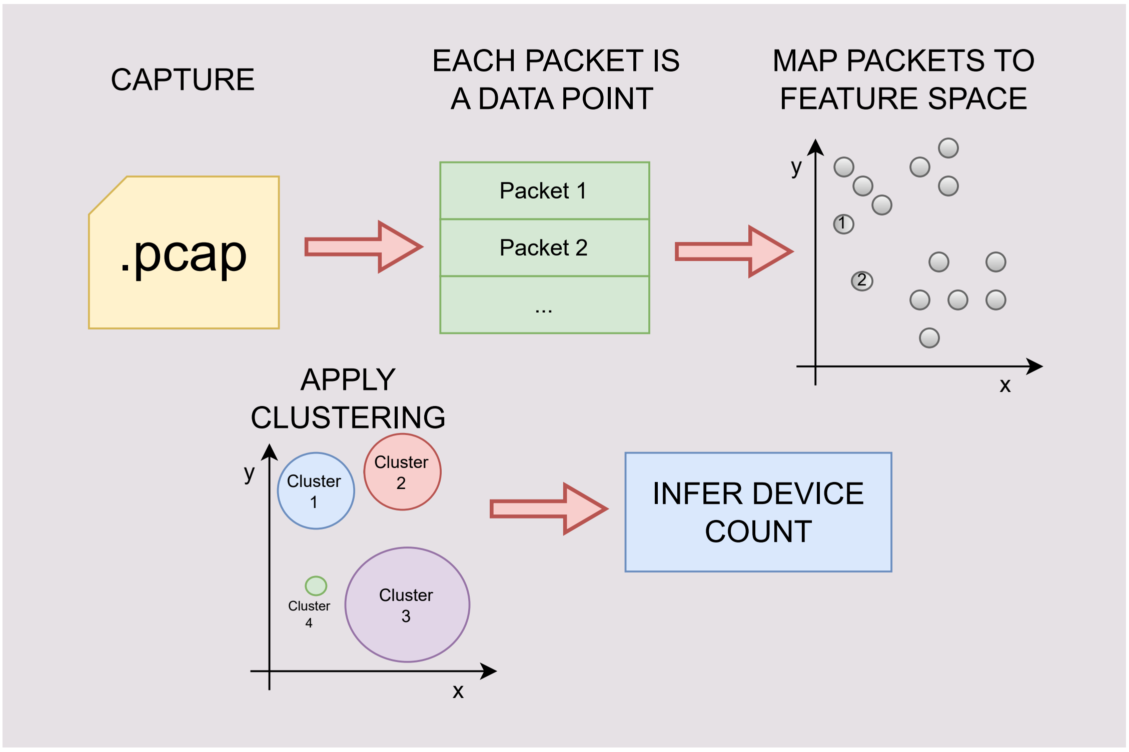


Figure 3.11: Clustering based on probe request.

Many studies have followed this approach, with DBSCAN variants being the preferred algorithms in the literature due to their high performance and native ability to recognize noise packets coming from outside the area of interest.

One example of a clustering application for crowd monitoring is the work in [27]. The authors have investigated the most relevant fields of a WiFi probe request for identifying a device, in order to use them to perform a clustering. The paper presents an analysis of WiFi probe requests to identify unique device fingerprints using information elements (IEs). This is done analysing a labeled dataset of probe requests from individual devices in isolated environments. Then with the use of a Random Forest algorithm they determined the importance of different IEs for recognizing the device related to the probe request. The research shows that three IEs (Extended Capabilities, HT Capabilities, and Vendor

Specific values) are most valuable for device fingerprinting. Based on this, the authors implemented a clustering algorithm based on DBSCAN and achieved an average accuracy of 92% in correctly clustering probe requests by device.

However, this methodology has a drawback: packets of different devices of the same vendor or model are very likely to be considered in the same cluster. Consequently, these devices are counted together as a single device and not recognized as separate devices. This is because different devices of the same or similar model have high probability to have identical fields in probe request body. To address this issue, additional handling is required.

Another example of cluster framework, where this issue is managed, is [4]. The work presents a framework for people counting based on clustering techniques. It first applies a DBSCAN algorithm, to cluster packets from capture of WiFi probe request, and then based on the size of the clusters provide a value for device counting in the capture.

Initially, before the clustering, probes are filtered based on the received signal strength indicator (RSSI), retaining only probes above a specified power threshold. Packets with globally unique MAC addresses are excluded from clustering, since they can already be considered as unique devices. For this reason, they are only used to compute the count of distinct global MAC addresses within the capture (T_g). Duplicate packets (from the same burst i.e., with the same MAC address) are also discarded, considering only one packet from each burst.

After the preprocessing, for each packet, three features are extracted to perform clustering. The features are the sum of all HT Capabilities values of the probe, the sum of all VHT capabilities values of the probe, and the sum of all extended capabilities values in the packet.

The DBSCAN algorithm is then applied to the processed data with the extracted features. To address the issue of multiple devices' probes potentially belonging to the same cluster, the framework estimates the number of devices per each cluster.

The estimation is based on the cluster nearness from probes of known tested devices. For each of the devices tested, is available the average burst rate: the expected bursts that the device sends in a unit of time, called L. The average burst rate (L) is calculated based on experiments performed on [28]. Each device was tested in an isolated environment and in different condition of usage. Based on the probes captured during the tests, the researchers were able to calculate the average rate of burst sent from that device.

The framework proposed in [4] estimate the number of devices (K) in each cluster based on the value L of the nearest (in term of clustering features) device in the database of known devices:

$$K = \frac{N}{L \cdot T}$$

Where N is the total number of elements in the cluster and T is the time duration of the capture. The final predicted number of devices (y) is the sum of all estimated devices across clusters plus the count of distinct global MAC addresses (T_g): $y = \sum K + T_g$.

Clustering approaches for people counting have demonstrated high accuracy and is currently the preferred solution for people counting based on WiFi probe requests.

Chapter 4

ML-Based Solutions For People Counting

This Chapter presents our implemented approach for a crowd monitoring system based on the analysis of probe request captures. More in detail, it presents the hardware and software used and the capture methodology implemented.

Finally, it is presented how capture traces are used within a regression machine learning algorithms to address the problem of people counting. Defining the preprocessing phase and the choice of the model.

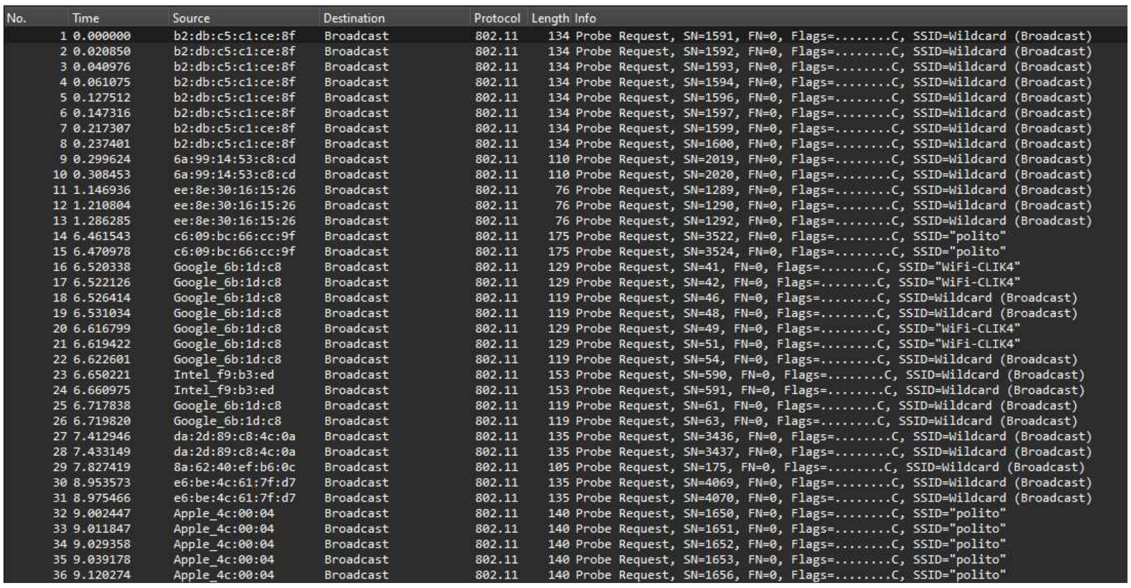
4.1 Capture Methodology

The numerous advantages, presented in Section 2.4.1, of WiFi tracking-based systems, make it the ideal approach for implementing a monitoring system: our research focused on analyzing WiFi traces, specifically WiFi probe requests. The analysis of WiFi probe request packets is used to implement crowd monitoring systems, specifically for people counting.

As presented in Section 2.2.1, to capture network traffic within an area, the use of a sniffer is necessary. In our research, we use Raspberry Pi 4b as sniffer due to its flexibility and affordability, making it ideal in a research scenario. The devices were equipped with a USB WiFi dongle to enable monitor mode on the network.

At the device startup, a bash script is triggered to set the monitor mode and intermittently capture all WiFi traffic, filtering only on probe request packets and saving them in a *.pcap* file. To capture network traffic and manage the probe request traces, the script uses tshark, a CLI application derived from Wireshark, as it is one of the best and most complete tools for detailed capture analysis.

To avoid data loss and facilitate easier management and organization of captures, the script uses a temporal parameter to define the duration of a single capture. After this period, the capture is saved in a *.pcap* file, and a new capture starts subsequently. A *.pcap* file is a standardized data file format used to capture and store network packet data. It is the most used file extension for packet captures as it permits to log network traffic for future network analysis and investigations. In Figure 4.1 an example of a *.pcap* file is shown.



```
No.   Time      Source             Destination  Protocol Length Info
    1 0.000000  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1591, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    2 0.020850  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1592, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    3 0.040976  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1593, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    4 0.061075  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1594, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    5 0.127512  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1596, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    6 0.147316  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1597, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    7 0.217307  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1599, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    8 0.237401  b2:db:c5:c1:ce:8f  Broadcast    802.11    134 Probe Request, SN=1600, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
    9 0.299624  6a:99:14:53:c8:cd  Broadcast    802.11    110 Probe Request, SN=2019, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   10 0.308453  6a:99:14:53:c8:cd  Broadcast    802.11    110 Probe Request, SN=2020, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   11 1.146936  ee:8e:30:16:15:26  Broadcast    802.11     76 Probe Request, SN=1289, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   12 1.210804  ee:8e:30:16:15:26  Broadcast    802.11     76 Probe Request, SN=1290, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   13 1.286285  ee:8e:30:16:15:26  Broadcast    802.11     76 Probe Request, SN=1292, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   14 6.461543  c6:09:bc:66:cc:9f  Broadcast    802.11    175 Probe Request, SN=3522, FN=0, Flags=........C, SSID="polito"
   15 6.470978  c6:09:bc:66:cc:9f  Broadcast    802.11    175 Probe Request, SN=3524, FN=0, Flags=........C, SSID="polito"
   16 6.520338  Google_6b:1d:c8    Broadcast    802.11    129 Probe Request, SN=41, FN=0, Flags=........C, SSID="WiFi-CLIK4"
   17 6.522126  Google_6b:1d:c8    Broadcast    802.11    129 Probe Request, SN=42, FN=0, Flags=........C, SSID="WiFi-CLIK4"
   18 6.526414  Google_6b:1d:c8    Broadcast    802.11    119 Probe Request, SN=46, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   19 6.531034  Google_6b:1d:c8    Broadcast    802.11    119 Probe Request, SN=48, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   20 6.616799  Google_6b:1d:c8    Broadcast    802.11    129 Probe Request, SN=49, FN=0, Flags=........C, SSID="WiFi-CLIK4"
   21 6.619422  Google_6b:1d:c8    Broadcast    802.11    129 Probe Request, SN=51, FN=0, Flags=........C, SSID="WiFi-CLIK4"
   22 6.622601  Google_6b:1d:c8    Broadcast    802.11    119 Probe Request, SN=54, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   23 6.650221  Intel_f9:b3:ed     Broadcast    802.11    153 Probe Request, SN=590, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   24 6.660975  Intel_f9:b3:ed     Broadcast    802.11    153 Probe Request, SN=591, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   25 6.717838  Google_6b:1d:c8    Broadcast    802.11    119 Probe Request, SN=61, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   26 6.719820  Google_6b:1d:c8    Broadcast    802.11    119 Probe Request, SN=63, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   27 7.412946  da:2d:89:c8:4c:0a  Broadcast    802.11    135 Probe Request, SN=3436, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   28 7.433149  da:2d:89:c8:4c:0a  Broadcast    802.11    135 Probe Request, SN=3437, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   29 7.827419  8a:62:40:ef:b6:0c  Broadcast    802.11    105 Probe Request, SN=175, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   30 8.953573  e6:be:4c:61:7f:d7  Broadcast    802.11    135 Probe Request, SN=4069, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   31 8.975466  e6:be:4c:61:7f:d7  Broadcast    802.11    135 Probe Request, SN=4070, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   32 9.002447  Apple_4c:00:04     Broadcast    802.11    140 Probe Request, SN=1650, FN=0, Flags=........C, SSID="polito"
   33 9.011847  Apple_4c:00:04     Broadcast    802.11    140 Probe Request, SN=1651, FN=0, Flags=........C, SSID="polito"
   34 9.029358  Apple_4c:00:04     Broadcast    802.11    140 Probe Request, SN=1652, FN=0, Flags=........C, SSID="polito"
   35 9.039178  Apple_4c:00:04     Broadcast    802.11    140 Probe Request, SN=1653, FN=0, Flags=........C, SSID="polito"
   36 9.120274  Apple_4c:00:04     Broadcast    802.11    140 Probe Request, SN=1656, FN=0, Flags=........C, SSID="polito"
```

Figure 4.1: Visualization of a *.pcap* file content.

Within a capture, each packet is associated with a specific timestamp (the time with microsecond precision) of its capture by the sniffer device, as well as the Received Signal Strength Indicator (RSSI).

The RSSI, in the context of WiFi networks, is a metric that measures the power level of the received radio signal by the antenna of the receiver from a wireless sender device. The value of RSSI contributes to determine the signal strength and quality of a WiFi connection. RSSI values are typically expressed in *dBm* (decibels relative to one milliwatt). These values are negative, with values closer to 0 indicating stronger signal strength. Common RSSI values range from -30 dBm (excellent signal strength) to -100 dBm (very poor signal strength). A higher RSSI generally indicates a stronger signal, leading to better performance and higher data rates. RSSI is affected by many factors:

- Distance: The farther the device is from the AP, the weaker the signal strength.

- Obstacles: Physical obstructions can attenuate the signal, reducing RSSI.
- Interference: Interference from other electromagnetic signals can impact RSSI.
- Antenna Orientation: The positioning and orientation of sender and receiver can influence the RSSI.

Due to all the factors mentioned, RSSI is highly variable, even from one packet to another. Therefore, RSSI has to be used with caution.

4.2 Regression For People Counting

Considering that the desired output for a people counting task is a numerical value, the most intuitive machine learning model to use is regression, as it inherently produce as output a numerical value based on a given input. In this approach, features are extracted from each individual capture and used as input for the regression model. Unlike clustering and regression tasks, where features are extracted from individual packets, in regression they are extracted from the entire capture. The selection of these features is non-trivial and crucial for the model's performance. The operations of a regression model applied to people counting are illustrated in Figure 4.2.

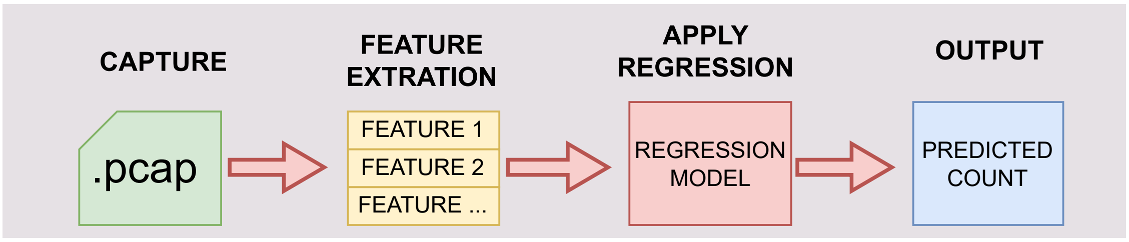


Figure 4.2: Regression based on probe request captures.

To train the regression model is necessary that the captures used in the training set are associated with the corresponding numerical count label. For this reason, another essential aspect is retrieving a large training set with accurate ground truth data.

This methodology has not been attempted in the literature before, and the primary contribution of this research is to develop this approach, while highlighting and proposing solutions to the most critical implementation challenges. All aspects of the model creation are considered, starting from preprocessing to extract meaningful and distinctive features, to defining appropriate datasets for training and testing, and finally with the selection of the model and fine-tuning of its hyperparameters. Once this process is implemented, the model can be trained and used to make predictions in real-time.

4.3 Feature Selection

The initial focus of the research is to determine the appropriate inputs for the regression model. As specified before, the capture serves as input for the regression model. Since the *.pcap* capture is highly unstructured data, it would not be possible to pass it directly to the model as input. Instead, the capture should be preprocessed to produce the model input features. It is crucial that the extracted features accurately reflect the number of captured devices, ensuring their quality and relevance. Alone or in combination with other features, they should provide insights about the capture, enabling the model to relate them to the ground truth. We started by defining measures that are intuitively linked to the number of devices present in the capture area, and progress to more refined features.

4.3.1 Naive Features

It is immediately clear that the more devices there are, the more packets they are likely to generate. Therefore, the first characteristic of a capture naturally associated with the number of devices in the area is the "size" of the capture, i.e., the total number of captured packets. With this in mind, additional features defined in our research, that can intuitively be associated with the number of devices in the capture are:

- Total number of packets
- Total count of distinct MAC addresses
- Time duration of the capture: all measurements must obviously be related to the capture time. With the same number of devices, the longer the capture duration is, the more packets are likely to be present in the capture.

The total number of packets and the total count of distinct MAC addresses can be further divided to better explore the differences between packets with locally administered MAC address and globally unique MAC address. The selected features thus become:

- Total number of local packets ($\#Packet\_local$)
- Total number of global packets ($\#Packet\_global$)
- Total count of distinct locally administered MAC addresses ($\#MAC\_local$)
- Total count of distinct globally unique MAC addresses ($\#MAC\_global$)
- Time Duration of the capture (T)

4.3.2 RSSI

This Section analyzed the possibility of using RSSI as a fingerprint for a device. To conduct this evaluation, we examined the probe request traces available from the work of [28]. These traces, collected in an isolated environment, contain only packets generated by a single device. Both the monitoring device and the test device were stationary at a fixed distance from each other with no obstacles in between.

From these traces, it is possible to observe how the RSSI value of probe requests varies. To reduce the variability of the RSSI value, the maximum RSSI within a burst is considered, and its stability over time is evaluated. If it is sufficiently stable, it would be possible to determine the probability that two bursts belong to the same device based on the RSSI value.

In Figure 4.3 and 4.4 the test results of two smartphones are shown. Tests were conducted on all devices examined in the research, and the results were consistent across all. The maximum RSSI value within a burst, even under conditions of absolute isolation and stationarity, remains too variable. The Figures also demonstrate that even when taking the maximum value within a burst, the RSSI remains similarly distributed.

For this reason, it would not be useful to use RSSI as a feature in a machine learning models.

4.3.3 Other Features

In general, for a neural network, the more high-quality features available, the easier it will be to identify patterns for the model, leading to better performance. For this reason, we have added the following features to better explore the characteristics of the capture.

Average Intra-Burst Time

Packets within the same burst in a capture can be identified by their same MAC address. For each burst, it is saved the first and last timestamps of the burst packets in the capture. The burst is considered active during this interval. The intra-burst time, which is the time duration of the burst, is calculated by the difference between these timestamps. The average intra-burst time features (*avg_ibt*) is then derived by averaging these durations across all bursts.

Average Number Of Collisions

Using the burst time information from the *avg_ibt* calculation, another feature can be determined: the average number of collisions. A time instant is defined as the minimum timestamp precision considered: for our research is a millisecond. For each time instant

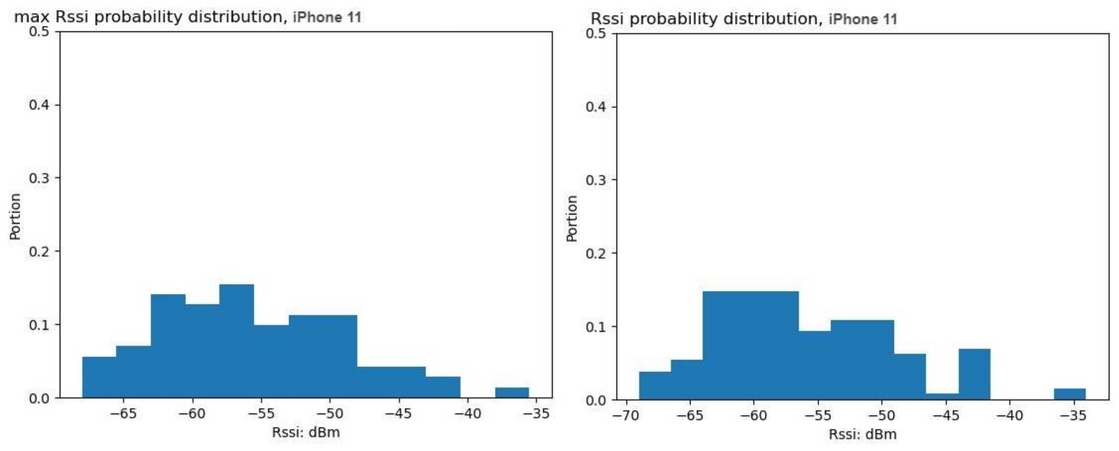


Figure 4.3: Probability distribution of the RSSI value for Apple iPhone 11.

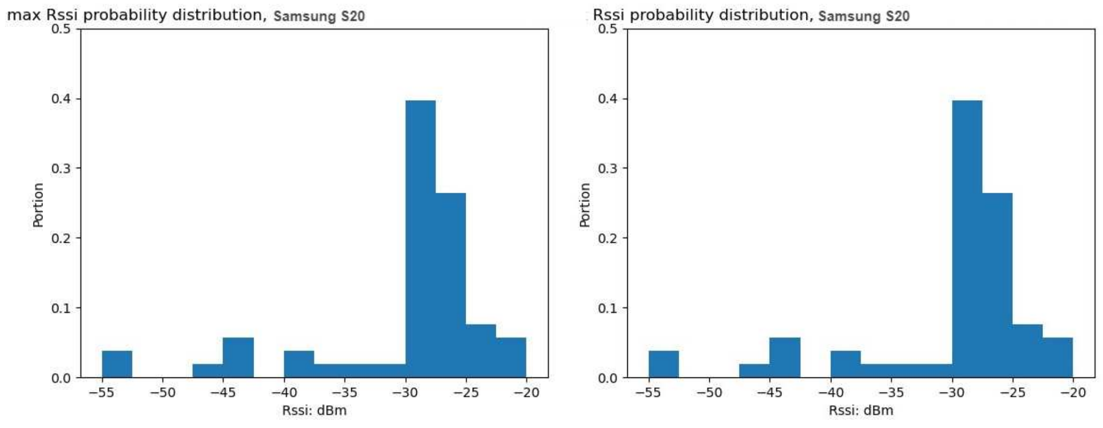


Figure 4.4: Probability distribution of the RSSI value for Samsung S20.

is calculated the number of collisions in that instant. The number of collisions is the total number of active bursts in a specific time instant. The feature, average number of collisions (*avg_colls*), is obtained by averaging the collision counts across all time instants.

Clustering Count

The model can also include outputs from other models as features. To incorporate the positive results and performance of the clustering approach for people counting discussed in Section 3.3.2, our model uses the count predicted by the clustering method defined in [4] as a feature (*y_clust*). This allows the model to use this value as a reference, enriching and enhancing it.

Number Of Different Information Elements

Built on the concept of incorporating positive results from clustering approaches, the model also includes the number of different data points produced by the clustering feature extraction performed in [4]. To do this, each probe request is processed to extract the coded Information Elements. At the end of the processing, the count of distinct data points extracted (*n_diff_IE*) is used as a feature.

Summarizing, the features that the model uses are: *#Packet_global*, *#Packet_local*, *#MAC_local*, *#MAC_global*, *T*, *avg_ibt*, *avg_colls*, *y_clust* and *n_diff_IE*.

4.4 Capture Preprocessing

In this Section, we provide a more detailed illustration of the preprocessing phase, explaining how it is implemented in our framework. As shown in the previous Sections, the preprocessing phase is responsible for feature extraction. Besides producing the defined features for the model, preprocessing allows for the analysis of the input captures, filtering and cleaning noisy packets.

The preprocessing process involves iterating over each WiFi probe request in the capture, updating the variables for the final features calculation based on the values of WiFi probe request fields. Preprocessing can also include data modification systems, such as data augmentation, by applying transformations and perturbations to increase the availability and variety of data. How this is used in our framework will be detailed in Section 5.1.3.

4.4.1 Power filter

During capture preprocessing, data cleaning can also be performed. In our case, for instance, it is possible to discard from the analysis packets, based on specific values, such as the RSSI field.

Although, as shown in Section 4.3.2, RSSI cannot be used as a feature, it remains a useful piece of information that can be used to filter the captured packets, eliminating noise and improving the overall quality of the capture.

Real-world captures often face challenges such as packet loss due to interference or the inclusion of packets from devices outside the target area. To mitigate these issues, we process the captures by passing them through a power filter based on the RSSI values. This filtering step aims to reduce the impact of noise and enhance the quality of the data used for feature extraction.

The operation of the filter is as follows: a threshold value in dBm is chosen. Since the threshold is expressed in dBm, it can only have negative values: the closer the threshold is to zero, the stronger the filter becomes, resulting in more packets being discarded. As the packets in the capture are cycled through, their RSSI value in dBm is checked. If it is below the threshold, the packet is discarded and not considered for feature calculation.

In our analysis, we considered various filtering values, ranging from no filter applied to an aggressive filter of -55 dBm. The values considered are summarized in Table 4.1.

RSSI threshold values (dBm)	-55	-65	-70	-85	-100	$-\infty$ (without filter)

Table 4.1: Possible RSSI threshold values for the power filter.

4.5 Neural Network Models

In this Section, we present the machine learning models selected for processing the input features and performing the regression. To ensure a comprehensive approach, we evaluate two different models, comparing and contrasting their differences. The chosen models are a FCNN and a LSTM network.

We describe the architecture of both models and outline the hyperparameters that need to be tuned to achieve optimal performances.

4.5.1 Fully-Connected Neural Network

The initial model proposed is a Fully Connected Neural Network. This network architecture takes as input a feature vector x of size *input_size* equals to the number of features used. The network produces a single positive number as output (output_size = 1) using a ReLU activation function (which ensures that the value is always positive), representing the predicted count.

In addition to specifying the input and output sizes, the architecture includes defining the number of layers, their respective sizes, and the activation functions used between these layers. Figure 4.5 provides a summary of the FCNN architecture.

During training and evaluation, we investigated the optimal combination of the number of layers, their sizes, and the activation functions to determine the best-performing model configuration.

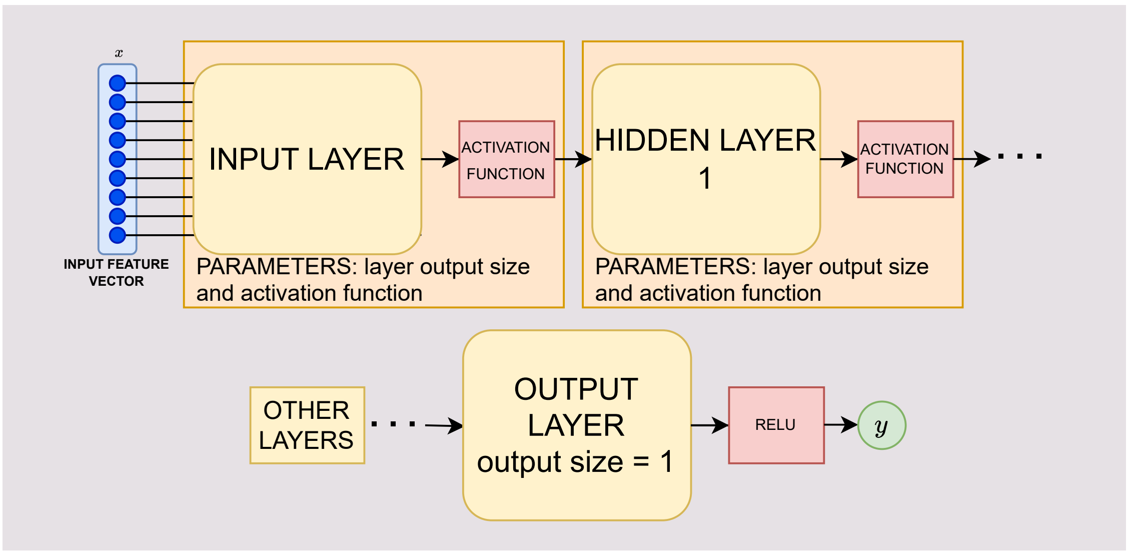

x
INPUT FEATURE VECTOR
INPUT LAYER
ACTIVATION FUNCTION
PARAMETERS: layer output size and activation function
HIDDEN LAYER 1
ACTIVATION FUNCTION
PARAMETERS: layer output size and activation function
OTHER LAYERS
OUTPUT LAYER output size = 1
RELU
y

Figure 4.5: Architecture of the proposed Fully Connected Neural Network.

4.5.2 Short-Long Term Memory Neural Network

The second proposed model is an LSTM network, where multiple LSTM cells are stacked to produce the desired output. The input and output sizes are the same as those for the FCNN, but adjustments, due to the recurrent nature of the network, are necessary to adapt the input and output data.

An LSTM processes sequential inputs: the input entry of the model consists of a series of x feature vectors, stacked to form a matrix X. To enable the LSTM cells to identify temporal relationships between captures, it is essential that the features extracted from the captures are in temporal sequence. The number of inputs taken is defined by a parameter T, it represents the time window during which the model can retain memory of previous captures. The ability to analyze more data and to learn temporal patterns likely improves performance, but it has as a drawback the fact that requires T consecutive captures for making a prediction.

The size of the model input X is therefore: *(input_size, T)*. For this reason, the features data need to be expanded before being passed as input to the network. The data expansion operation is illustrated in Figure 4.6.

As shown in Section 3.2.3, the output of an LSTM is the hidden state, which is a vector with length equal to the hidden size of the LSTM. Therefore, to obtain the desired count value as output, a linear layer with size = 1 (i.e., a single neuron) and ReLu activation function is connected to the last LSTM output. Figure 4.7 summarizes the architecture of the LSTM network.

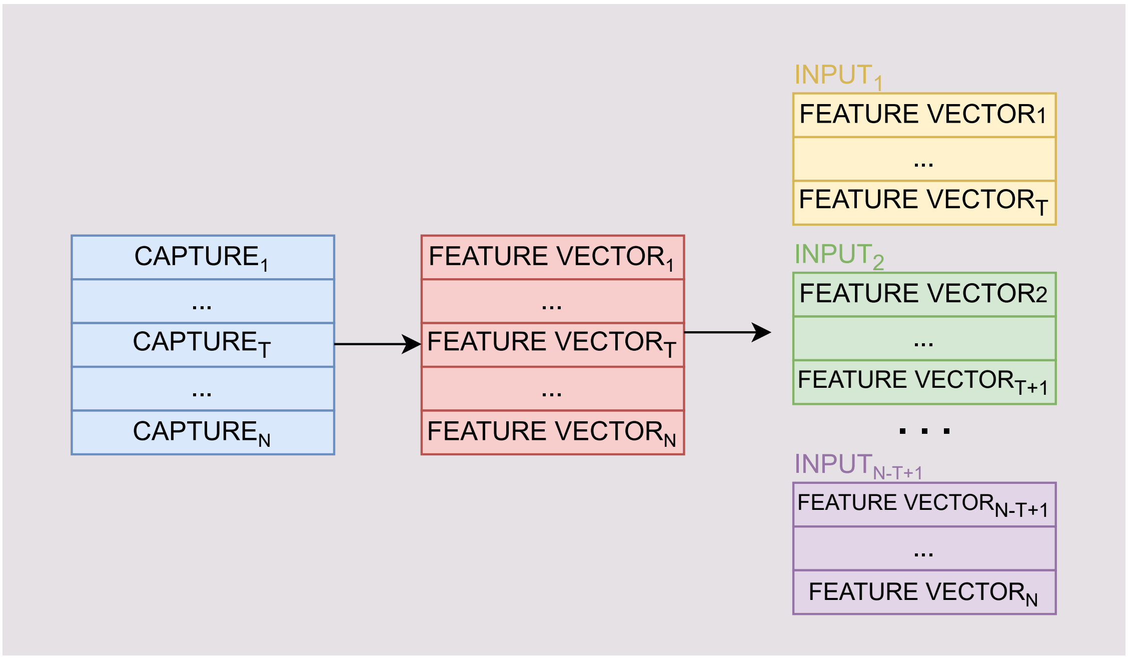


Figure 4.6: Data expansion process, for passing T sequential features as input to the LSTM network.

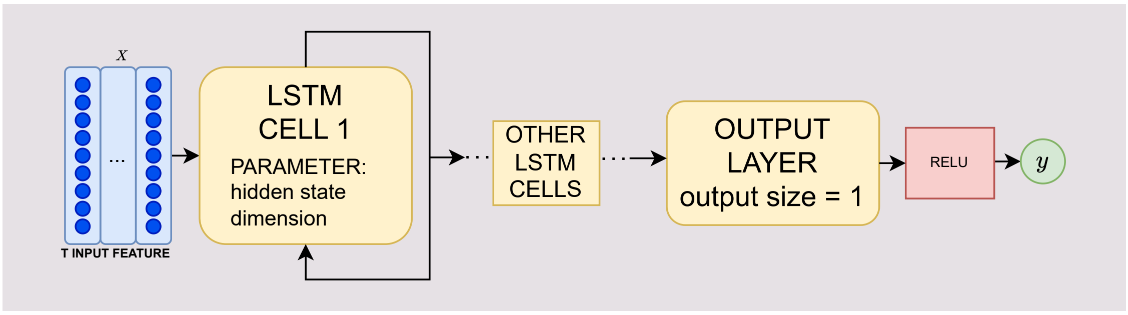


Figure 4.7: Architecture of the proposed LSTM Network.

The optimal number of LSTM cells and their respective sizes, as well as the influence of the time window T, will be investigated during training and evaluation tests.

Chapter 5

Dataset Selection

This Chapter addresses the selection of dataset to use for training and testing phases in our proposed regression framework. It discusses the challenges of obtaining accurate labeled data, i.e., captures with associated device counts. It presents our approach to overcoming this issue, proposing solutions for obtaining such extensive and representative datasets.

Proper training of a neural network requires a large amount of high-quality data, that accurately represents the various cases and capture scenarios the model may encounter. In the Chapter we discuss the methodologies and strategies used to ensure data accuracy, diversity, and volume of labeled captures.

The proposed methodologies for retrieving labeled data are based on two approaches. The first one is based on the generation of synthetic data, through the use of a software able to generate WiFi probe request captures from a predefined number of devices, emulating the MAC address randomization behavior of real devices. The second in based on the sniffing procedure described in Section 4.1, with ground truth provided by manual counting the people in the capture area.

Finally, a detailed overview of the various datasets collected is provided.

5.1 Establishing The Ground Truth

To train regression models, sample data are essential to extract knowledge. This sample captures must be associated with the device counting ground truth.

In literature, there are available many datasets of probe requests, collected from various public and private environments. The research in [29] present a summary of the most popular datasets of WiFi probe requests available. The dataset presented are: *Sapienza2013* (captured in various locations in Rome), *Glimps2015* (collected during a music festival in Ghent, Belgium), *Nile2021* (captured during nighttime in a shopping center), *IPIN2021* (collected during the IPIN 2021 conference in Spain) and *Pintor2021* (achieved by capturing probe requests in an anechoic chamber to suppress environmental signals). Among these, the only dataset associated with ground truth labels is *Pintor2021*. However, it is a small dataset with captures of a single devices for only 20 minutes.

Due to the lack of labeled counting data, a significant challenge arises when attempting to utilize these datasets for training machine learning models. This absence hampers their utility for training models, that require precise labels to learn effectively.

The few datasets that do include labeled counting data, such as Pintor2021, cannot be used due to the critical limitation of their size. These datasets are too small to provide a robust training for machine learning models. Small datasets do not capture the variability and complexity of real-world environments, leading to models that are overfitted: they perform well only in contexts similar to the training ones but they become very poorly when applied to broader contexts.

Overcoming these challenges requires laborious approaches for labeling large datasets, which is considerable time-consuming and prone to error.

Another alternative is to rely on synthetic data. Although this approach has limited applicability, because the data may not accurately reflect real-world conditions, it can still illustrate and demonstrate the validity of the adopted framework.

The last and best alternative is to personally collect numerous data from various situations (such as either indoor and outdoor environments, or stationary and transitory areas, etc.) while ensuring the ability to count people during the capture process. However, this method requires intensive and elaborate monitoring.

The simplest method for counting people while capturing data is manual counting, but this presents several difficulties. Manual counting is time-consuming, prone to human error, and challenging to scale. In crowded or highly dynamic spaces, people frequently move, interact, and change positions, making it hard to count individuals accurately. Moreover, in outdoor scenarios, it is challenging to determine the distance within which a person should be included in the count or not. Despite these challenges, manual counting can provide accurate ground truth data in many situations.

Another solution for providing counting labels is camera-based systems. Camera-based counting would be the best option because it is both accurate and scalable. It can be completely automated, able to handle large data and various environments, providing precise counts without the need for manual intervention. However, these systems require specific configuration of hardware and software; and above all they have significant privacy concerns. They require capturing and storing video, which raise issues related to the privacy of individuals recorded. Additionally, camera-based counting may have limitations such as a restricted field of view, which may result in inaccurate counts, especially in crowded or large areas. Variations in lighting and physical obstructions can also limit the effectiveness of camera-based counting systems.

Our research investigates two of the approaches proposed above, one based on generating synthetic data and the other based on real-world manual capturing and counting.

5.1.1 Probe Request Generator

Even though the synthetic data approach may not fully capture the complexities and variability of real-world scenarios, it allows us to create controlled environments for testing the validity of our models under various conditions.

For this purpose, we utilize a Python script called *Probe Request Generator* [30], which can be easily executed through the command line. It allows us to generate *.pcap* captures from a predefined configured number of devices and conditions. The development of the Probe Request Generator is detailed in [31] and is also used in [4]. It is built upon a solid understanding of probe request messages with MAC address randomization and the IEEE 802.11 standard.

Using the Probe Request Generator, we can produce as many datasets as needed in the form of *.pcap* captures, along with their associated ground truth labels. This capability allows us to train the regression model effectively.

Although this is a synthetic approach, the generator is designed to emulate the real behavior of the devices examined in [31] with exceptional precision. Consequently, a model trained on this generated data can also be used to accurately predict real capture data.

Input Parameters

The probe request generator offers significant flexibility by allowing customization of the following parameters:

- **Script time**: This parameter defines the duration of the capturing window for the final *.pcap* capture.
- **Average device number**: This parameter lets specify the average number of devices present at any time.
- **Average permanence time**: This parameter indicates the average time a device is considered to remains within the antenna's coverage area.

Furthermore, with minor modifications to the probe request generator code, there is the option to create captures with devices in a specific states (i.e., active, awake, or locked) to simulate different contexts. This allows to replicate environments where users are primarily using their devices (active state), such as at transport hubs; environments where people use their devices in locked mode to check time and notifications (awake state); and environments where devices are kept locked (locked state), such as in a university class during lessons. This is important because, as demonstrated in the research [32], these different states influence the probe request send rate.

With small modifications to the generator code, it is also possible to define the device models used in the simulation within those available.

Output Generated

The probe request generator produces two key outputs. Firstly, the *.pcap* trace file, that includes all the probe request packets transmitted by the devices in the simulation. Secondly, it generates the crucial ground truth value, that precisely indicates the number of devices that have sent at least one probe request.

Dataset Generation

A dataset can be generated by simply running multiple time the generator with the desired number of devices, the desired capture time, and the average permanence time as parameters.

To create an effective and varied dataset, we simulated a scenario in which each capture in the dataset can have devices count from 0 to 100. To ensure an even distribution of values, the number of devices increases or decreases gradually. Initially, the count predominantly increases from 0 until reaching 100, with random fluctuations between -3 and +3; with positive fluctuations being more likely, creating an overall upward trend. From this point, the trend predominantly decreases, using the opposite method, until it returns to 0. This cycle repeats until reaching 1,000 captures: a sufficient number for training the network.

This method provides a realistic temporal pattern in the captures, where people increase and decrease continuously over time, allowing the LSTM-based model to learn and predict temporal patterns more effectively. An example of the ground truth generated using this approach is shown in Figure 5.1.

The capture time is random, ranging between 30 seconds and 5 minutes. The average permanence time is defined to be equal to the capture time. We have defined the device models used in the simulation to be randomly chosen in proportion to the vendor market

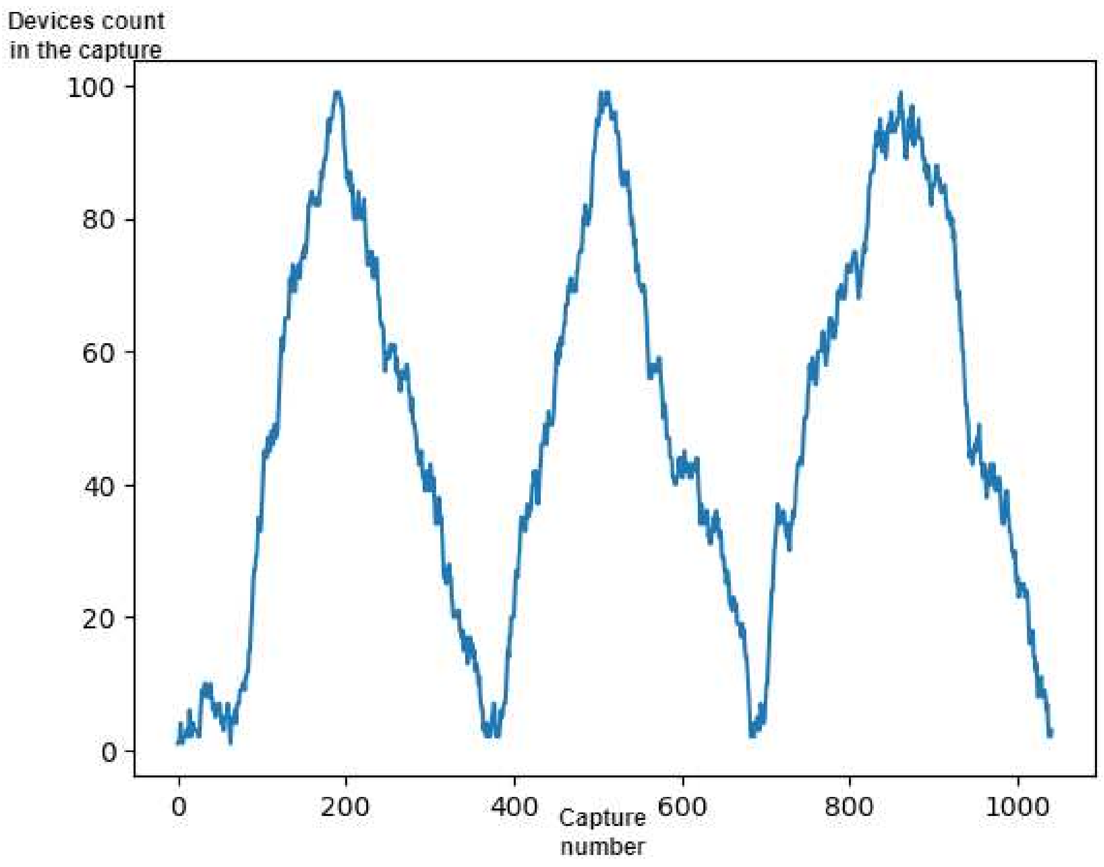


Figure 5.1: Sequential captures generated with the generator along with their respective device counts.

share [33], and the device states (active, awake, locked) to be varied. The parameters used and their possible values are summarized in Table 5.1.

Parameter	**Configured values**
Script time	Random, between 30 seconds and 5 minutes
Average device number	0 to 100
Average permanence time	Equal to script time
State	Mixed states
Generated device models	Models chosen randomly in proportion to market share

Table 5.1: Probe Request Generator configured parameters.

Features Used In Generator-Based Model

The generator is designed to simulate devices that use MAC address randomization. Therefore, the model trained on the generator data cannot make predictions based on probe requests with globally unique MAC addresses, but only on those with locally administered MAC addresses. Consequently, when testing real captures with generator-trained models, the input features are computed by excluding global probe requests from the capture. For the same reason, features based on global probe requests (*#Packet_global* and *#MAC_global*) are also excluded.

Furthermore, features related to clustering models (*y_clust* and *n_diff_IE*) cannot be applied because the generator produces data based on only a few device models. In contrast, a real capture may contain probe requests from many different device models, resulting in features that are not representative and could lead to misleading results.

Therefore, models trained on generator data use only five features (*#Packet_local*, *#MAC_local*, *T*, *avg_ibt*, *avg_colls*) instead of the initials nine.

For these reasons, when trained with generated data, the model's output is not the total number of devices but rather the total number of devices that use locally unique MAC addresses. To handle devices that use globally unique MAC addresses, each distinct global MAC address in the test capture is considered a unique device. Therefore, by summing the count of distinct global MAC addresses to the model's predicted output, the estimated total number of devices is computed.

5.1.2 Manual Counting

The second approach used for dataset acquisition is manual counting: with the Raspberry Pi configured as described in Section 4.1. Once selected the capturing area, the sniffer was powered on and it automatically started capturing data. The sniffer captures for 2 minutes before saving the *.pcap* on the hard-drive. During this time, we manually recorded the average number of people present in the area in those 2 minutes. At the end of the capture process, the device was turned off, and the *.pcap* files extracted and analyzed.

This process is quite labor-intensive, requiring care, attention, and rules for counting or not counting a person. For these reasons, captures were conducted in indoor environments with minimal foot traffic. Specifically, two capture sessions were conducted, both inside classrooms of the Politecnico di Torino. One session took place in a small underground classroom, offering better shielding from external signals, while the other was conducted in a larger, standard classroom. In summary, at the end of a monitoring session lasting T minutes, we produce $T/2$ *.pcap* captures, each paired with a corresponding manual count label.

Lesson Capture

The first capture session took place in the underground classroom 14 of the Politecnico di Torino. This classroom was chosen for its advantageous position regarding external electromagnetic isolation, allowing for cleaner captures with less external noise.

We arrived in the classroom about 15 minutes before the start of the lecture, to be able to capture the variation in the number of people as students gradually entered. Upon our arrival, there were already some people present in the class. To obtain more data, we used two sniffers in this session. They were turned on and off simultaneously but positioned differently: one in the center of the classroom and the other at the back in the last row of desks.

The monitoring session lasted 52 minutes, producing 26 captures per sniffer. The count values corresponding to each capture are shown in Table 5.2.

Exam Capture

The second capture session took place in classroom 8 of the Politecnico di Torino, during a computer-based written exam. In this case, only one sniffer was used. It was turned on after the exam had started, with all the students already placed. The sniffer was positioned on the teacher's desk and was turned off when all the students had left the room.

The monitoring session lasted 54 minutes, producing 27 captures. The count values corresponding to each capture are shown in Table 5.3.

5.1.3 Data Augmentation

The real captures acquired are too few to be used to train a machine learning model effectively. To make them usable, data augmentation is necessary to increase the dataset size.

The data augmentation process works as follows. Each capture is divided into N chunks of equal temporal duration. Since the captures are temporally contiguous, the last chunk of a capture i is directly connected to the first chunk of capture $i+1$. Therefore, these chunks can be merged, maintaining their temporal dependence. This allows for the creation of new captures by combining all possible groups of $M < N$ contiguous chunks. By merging these chunks, new augmented captures can be generated.

Finally, a device count label must be generated for the newly created data: if the chunks of the generated capture come from the same original capture, the label remains

Lesson captures

Capture	**People counted**	**Capture**	**People counted**
1	5	**15**	21
2	5	**16**	21
3	7	**17**	21
4	6	**18**	21
5	6	**19**	21
6	7	**20**	23
7	9	**21**	22
8	9	**22**	22
9	12	**23**	22
10	15	**24**	22
11	14	**25**	22
12	18	**26**	23
13	20	**27**	23
14	21	**28**	23

Table 5.2: Class captures ground truth.

Exam captures

Capture	**People counted**	**Capture**	**People counted**
1	79	**15**	75
2	79	**16**	73
3	79	**17**	70
4	79	**18**	65
5	79	**19**	65
6	79	**20**	65
7	79	**21**	50
8	79	**22**	35
9	79	**23**	25
10	79	**24**	15
11	79	**25**	8
12	78	**26**	4
13	77	**27**	2
14	75		

Table 5.3: Exam captures ground truth.

the same as the original; else, if the chunks come from two consecutive original captures, the label is the mean of those two.

With this procedure, starting with C initial captures, a total of $N \cdot (C - 1) + M$ captures can be obtained. To avoid significantly reducing the capture duration, we have used $N = 3$ and $M = 2$. The data augmentation process is illustrated in Figure 5.2.

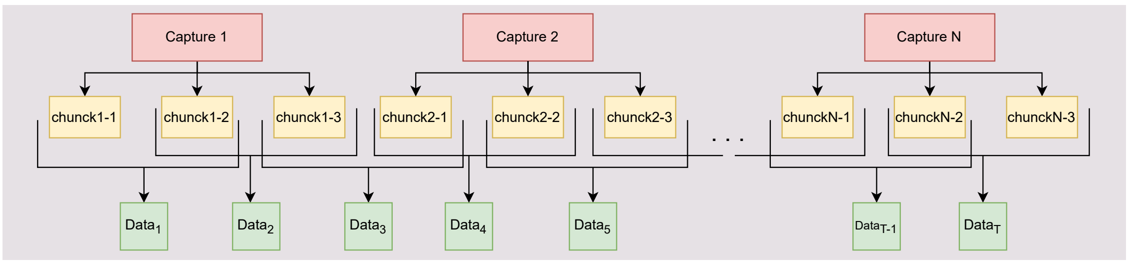


Figure 5.2: Example of data augmentation, with $N = 3$ and $M = 2$. $T = 3 \cdot (C - 1) + 2$.

The augmentation process offers several advantages:

- Increased Dataset Size: by artificially expanding the dataset, time and effort required for additional manual data collection is saved.
- Enhanced Generalization: the diversity introduced through augmentation techniques helps the model generalize better, improving its adaptability and robustness.
- Reduced Overfitting: by mixing various captures, the model is less likely to memorize specific details of the training data.

Overall, these benefits lead to significant improvements in the performance of the model trained on augmented data.

5.2 Dataset Review

This Section provides an overview of the datasets used in our research, including their source, size, and how they are utilized to train and test the model. Table 5.4 summarizes all these information.

Dataset name	**Data Source**	**Dataset size**	**Used for training**	**Testing usage**
Generator	Captures generated from the probe request generator	1000 captures of varying duration	Yes (using 5 features)	Testing generator trained models
Lesson	Captures from manual monitoring from a lesson in class 14 of the Politecnico Di Torino	56 captures of 2 minutes	No	Testing generator trained models
Lesson augmented	*Class* dataset with data augmentation	164 captures of 30 seconds	Yes (using 9 features)	Testing class augmented-trained models
Exam	Captures from manual monitoring from an exam in class 8 of the Politecnico Di Torino	56 captures of 2 minutes	No	Testing generator trained models

Table 5.4: Overview of the dataset used.

Chapter 6

Experimental Results

This Chapter presents the results obtained by our framework for estimating crowd size. The quality of the proposed framework has been evaluated using both synthetic and real data, demonstrating high level of accuracy. All evaluations are based on the Mean Square Error (MSE) metric, which is the most valuable indicator for regression tasks.

Firstly, the Chapter outlines the optimal architectures and parameters for both the LSTM network and Fully Connected Neural Network (FCNN). Hyperparameter tuning was conducted using the *Generator* dataset, created with the Probe Request Generator, and an 80-20 train-test split was applied. Post-training, the MSE values on both the training and test sets were evaluated to assess model quality. By exploring various configurations, the best hyperparameters for each model were determined.

Subsequently, models trained on the *Generator* dataset were utilized to predict crowd sizes in the *Lesson* and *Exam* datasets. An analysis of the power RSSI filter was conducted to evaluate its influence, determining the best threshold value for it.

Additionally, the optimal model architectures were used to train new networks for predicting the number of people in real captures. For this task, the best LSTM network and FCNN were trained and tested on augmented real data from the *Lesson augmented* dataset. The results provided insights into the models' performance with real data.

Following the numerical results, a discussion of the findings is presented. This includes an analysis of the positive outcomes, limitations, and potential areas of improvement.

6.1 Network Hyper-Parameters

This Section, investigate the optimal configurations of the proposed architectures for the LSTM network and FCNN. Each model requires hyperparameter fine-tuning, demonstrating how architectural changes impact performance.

Given the relatively small number of input features, there was no justification for increasing the model complexity with a deeper architecture. Therefore, to avoid overfitting, when similar accuracy is observed between configurations, we selected the configuration with the minimum complexity. Even if a configuration shows lower training accuracy, we prioritize those with better test accuracy, indicating superior generalization capabilities.

The models are trained and tested on the *Generator* dataset. Despite being a synthetic environment, the realistic behavior of the generator allows for an accurate assessment of the network's capacity to learn relevant relationships within the features. Additionally, this approach enables tests to be performed on a sufficiently large dataset. The model uses as input five features, detailed in Section 5.1.1: *#Packet_local*, *#MAC_local*, *T*, *avg_ibt*, *avg_colls*.

Each configuration is trained for 1000 epochs, using *Adam* as optimizer, with a batch size of 32, exponential learning rate decay, and min-max normalization applied to the input. After training, the models are tested, and the best configuration is selected based on both training and testing accuracy.

6.1.1 Fully-Connected Neural Network

For the FCNN, the hyperparameters to be investigated include the number of layers, the output size of each layer, and the activation functions between them.

Number Of Layers

Firstly we focus on the number of layers, while setting the layer sizes and activation functions to reasonable default values, which will be fine-tuned in subsequent investigations. Table 6.1 presents the possible configurations considered, along with their parameters and corresponding values of accuracy.

When evaluating the optimal configuration, we take into account not only the training error but also the model complexity and test error. Based on these criteria, the optimal number of layers for the FCNN is determined to be 2.

Layers	**Train error**	**Test error**
1 layer (size = 6)	64.53	91.36
2 layers (size1 = 7, size2 = 5)	**56.72**	**89.51**
3 layers (size1 = 7, size2 = 5, size3 = 3)	56.95	89.65
4 layers (size1 = 9, size2 = 7, size3 = 5, size4 = 3)	52.11	94.43

Table 6.1: Configurations of numbers of layers for the FCNN, with respective training and testing errors (MSE).

Layers Size

With the layers identified in the previous step, we now investigate their optimal size. Table 6.2 shows the possible configurations considered along with their parameters and accuracy. For the FCNN, the optimal sizes for the layers are determined to be 15 for the first layer and 7 for the second layer.

Size	**Train error**	**Test error**
2 layers (size1 = 30, size2 = 10)	66.19	73.73
2 layers (size1 = 20, size2 = 8)	63.03	71.53
2 layers (**size1 = 15, size2 = 7**)	**63.11**	**65.97**
2 layers (size1 = 10, size2 = 5)	63.95	66.65
2 layers (size1 = 8, size2 = 5)	66.53	70.56
2 layers (size1 = 6, size2 = 3)	74.51	74.52
2 layers (size1 = 4, size2 = 4)	71.57	71.94

Table 6.2: Configurations of output size of FCNN with two layers, with respective train and test errors (MSE).

Activation Functions

Lastly, we searched for the best activation function to use between layers. Table 6.3 shows the configurations considered along with their accuracy. For the FCNN, the optimal activation functions are determined to be ReLU for both layers.

Activation functions	**Train error**	**Test error**
ReLU (both first and second layers)	**67.43**	**71.12**
Sigmoid (first layer) + ReLU (second layer)	77.21	80.62
Tanh (first layer) + ReLU (second layer)	70.38	78.64

Table 6.3: Configurations of FCNN with different activation functions, with respective train and test errors (MSE).

The final architecture for the FCNN with the best hyperparameters found is illustrated in Figure 6.1.

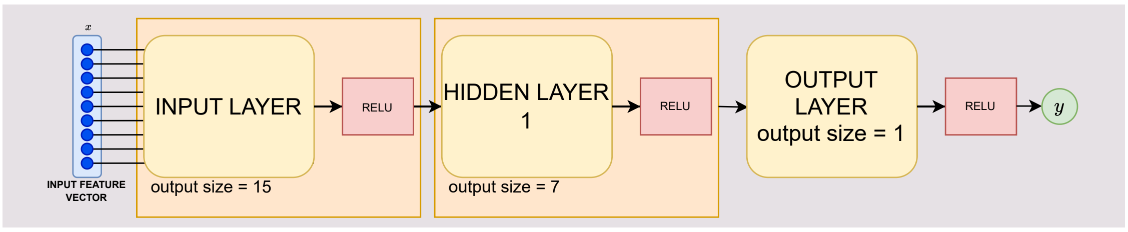


Figure 6.1: The optimal FCNN architecture found after hyper-parameters tuning.

6.1.2 LSTM Network

For the LSTM network, the hyperparameters to investigate include the number of layers and the hidden size of the LSTM cells.

Number Of Layers

Initially, we investigated the number of layers, setting the output sizes to reasonable default values, which will be optimized later. Table 6.4 shows the possible configurations considered, along with their parameters and accuracy. The results indicate that the best accuracy for the LSTM network is achieved with 2 LSTM layers.

Layers	**Train error**	**Test error**
1 LSTM layer (size = 6)	58.18	62.65
2 LSTM layers (size1 = 6, size2 = 3)	**40.49**	**65.29**
3 LSTM layers (size1 = 6, size2 = 4, size3 = 2)	35.26	68.79

Table 6.4: Configurations of LSTM network with different numbers of layers, with respective train and test errors (MSE).

Layers Size

With the optimal number of layers identified, we then investigate their optimal size. Table 6.5 shows the possible configurations considered, along with their parameters and accuracy. The results indicate that the optimal sizes for the LSTM layers are 15 for the first layer and 10 for the second layer.

Size	**Train error**	**Test error**
2 LSTM layers (**size1 = 15, size2 = 10**)	**29.70**	**39.45**
2 LSTM layers (size1 = 13, size2 = 6)	32.32	40.39
2 LSTM layers (size1 = 10, size2 = 5)	44.93	40.35
2 LSTM layers (size1 = 10, size2 = 10)	40.54	40.79

Table 6.5: Configurations of LSTM network with two layers of varying sizes, with respective train and test errors (MSE).

Data Expansion Time Windows

Lastly, we investigate the effect of the time window T used to generate the model input. Table 6.6 shows the possible values for T considered and their corresponding accuracy. The results indicate that a larger T value results in better performance. Therefore, T can be chosen based on the context and data availability of the prediction situation.

Time window	**Train error**	**Test error**
3	29.70	42.85
6	30.32	23.76
10	8.03	15.29
25	2.02	3.73

Table 6.6: Configurations of LSTM network with different time windows, with respective train and test errors (MSE).

The final architecture for the LSTM network with the best hyperparameters found is illustrated in Figure 6.2.

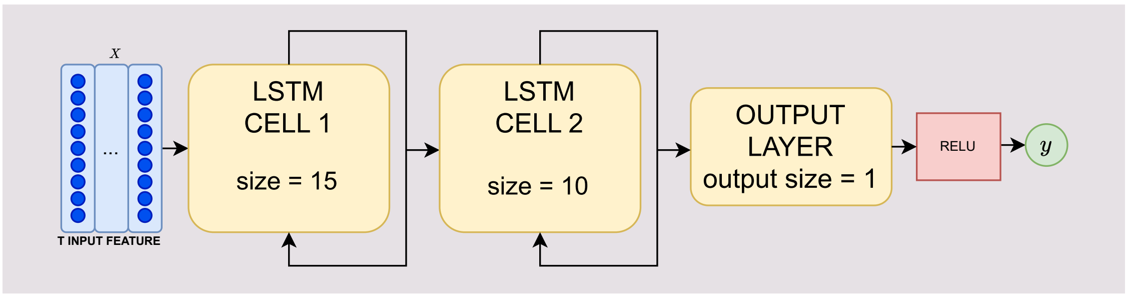


Figure 6.2: The optimal LSTM network architecture found after hyper-parameters tuning.

6.1.3 Other Algorithm Results

To compare the contribution given by the implementation of the model, the data are used, with the same preprocessing and features, in other models. A basic SVM obtained a result of MSE = 106.50 and instead a Random Forest obtained a MSE = 81.72. It can be seen that the FCNN and, especially, the LSTM network outperform both results.

6.2 Generator Based Model For Real Capture Prediction

In this Section, we present the tests performed using the models trained on synthetic datasets from the previous step, applied to real-world captures. The goal of this evaluation is to determine whether the generator can effectively provide training data that improves prediction accuracy in real-world scenarios.

The features extracted from the real captures are the same five features used during the training phase. Since the output of the model, based on the generator data, refers only to devices using MAC address randomization with locally administered MAC addresses, we can predict the final result, by combining the model's prediction (y) with the distinct count of globally unique MAC addresses (*#MAC_global*) observed in the filtered capture. The final prediction is computed as: $y\_pred = y + \#MAC\_global$

Moreover, we further analyze the effect of the power filter, discussed in Section 4.4.1, on real captured data from the *Lesson* and *Exam* datasets, providing results for both the LSTM network and the FCNN models.

For the LSTM network, we used a relatively small value of T. This choice was made because, while predicting, the first T captures cannot be tested as they would require T previous captures to be processed. To avoid significantly limiting the number of predictions, we chose $T = 6$.

6.2.1 Power Filter

The power filter value is a parameter of the preprocessing in our regression framework. For each value in Table 4.1 the effect of applying the power filter to the test captures is analyzed in terms of the model's performance on the *Lesson* and *Exam* datasets.

Lesson Results

Table 6.7 illustrates the MSE obtained by the FCNN and LSTM for the *Lesson* dataset for each considered value of the power filter.

From the results, it can be seen that the optimal filter value is -65 dBm. Relevant results for both the FCNN and LSTM network are illustrated in Figures 6.3 and 6.4. It

	$-\infty$ dBm	-100 dBm	-85 dBm	-70 dBm	-65 dBm	-55 dBm
FCNN	482.84	324.71	75.42	**35.25**	45.78	97.95
LSTM Network	382.42	244.85	68.53	**28.36**	36.93	81.23

Table 6.7: Lesson dataset results for each threshold value of the power filter applied.

is immediately visible from the Figures the importance of applying the filter. Without the filter, the model completely overestimates the prediction. On the other hand, an aggressive filter of -55 dBm reduces the predicted value too much, leading to a consistent underestimation.

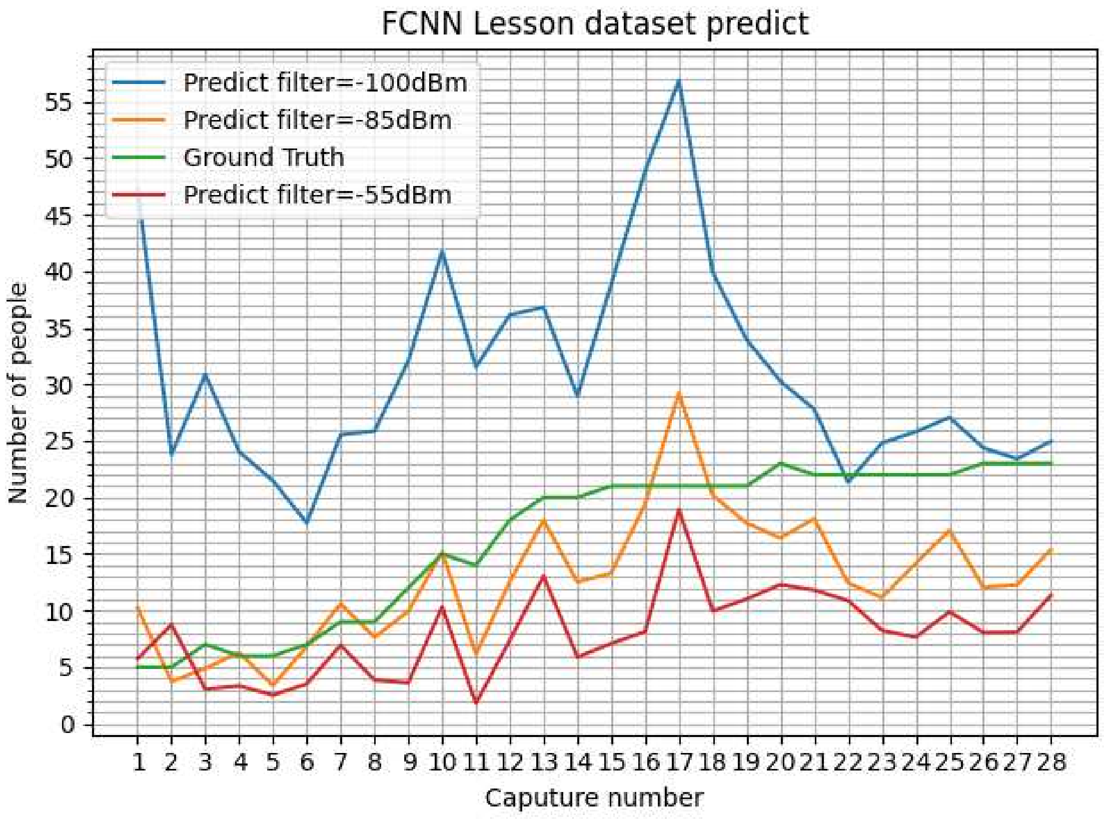


Figure 6.3: Comparison between ground truth and FCNN prediction with different power filter for Lesson dataset.

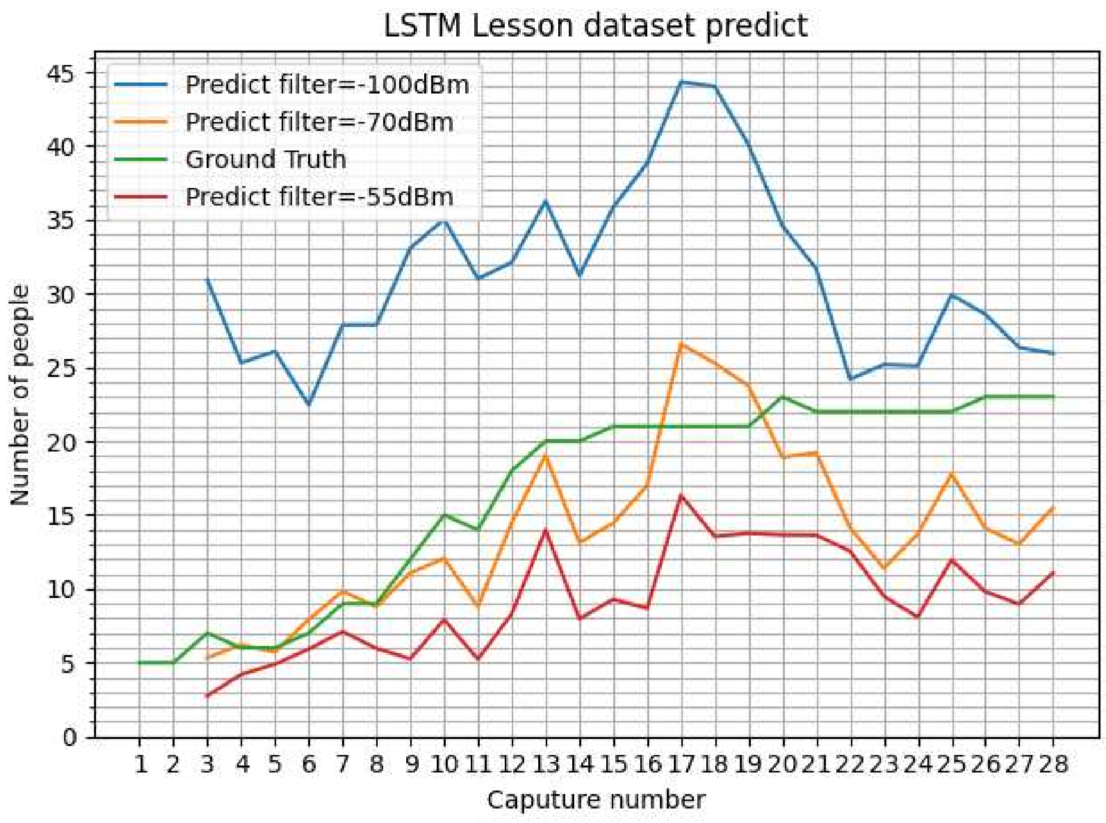


Figure 6.4: Comparison between ground truth and LSTM Network prediction with different power filters for Lesson dataset.

Exam Results

Table 6.8 instead illustrates the MSE obtained by the FCNN and the LSTM network for the *Exam* dataset for each considered value of the power filter.

	$-\infty$ dBm	-100 dBm	-85 dBm	-70 dBm	-65 dBm	-55 dBm
FCNN	2703.59	2610.19	**1560.65**	2216.83	2998.40	3775.77
LSTM Network	1933.08	1833.19	**1244.51**	2355.98	2974.39	4045.70

Table 6.8: Exam dataset results for each threshold value of the power filter applied.

From the results, it can be seen that for this case, the optimal filter value is -85 dBm. Relevant results for both the FCNN and LSTM network are illustrated in Figures 6.5 and 6.6.

In the latest Exam dataset captures, for both models, are evident limitations in predicting the reducing numbers of people, despite the application of filters. This issue may be linked to the fact that students, after leaving the room, are remaining in proximity zones and their devices are continuing to emit packets that are still being collected.

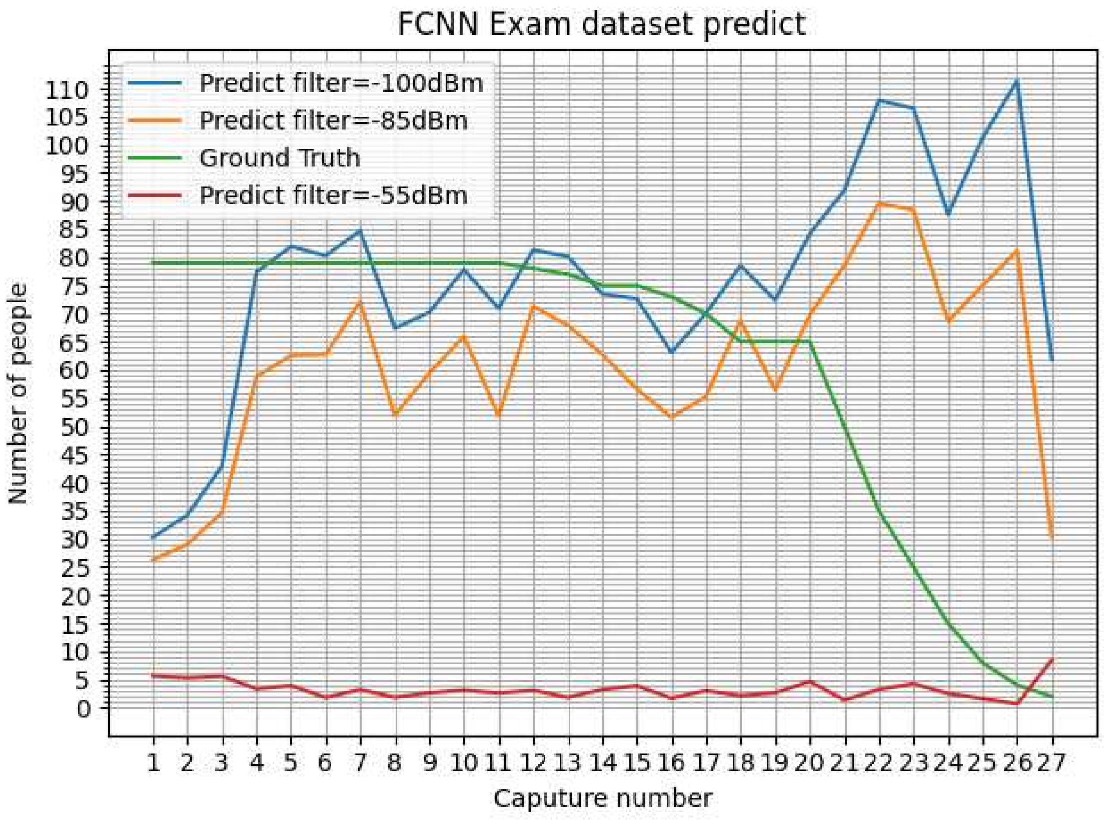


Figure 6.5: Comparison between ground truth and FCNN prediction with different power filter for Exam dataset.

6.3 Real Captures Training

The last test performed involves using real traces as training set. In this scenario, we utilized all nine defined features, including *y_clust* and *n_diff_IE* extracted from the clustering framework. For the LSTM network, we used $T = 3$.

For this task, we used the *Lesson Augmented* dataset. Due to its relatively small size (164 captures), the testing was performed as follows: for each initial 2-minute capture, the augmented data related to at least one of its chunks were used as the test set, while all other augmented data were used for training the models. The average of the predictions

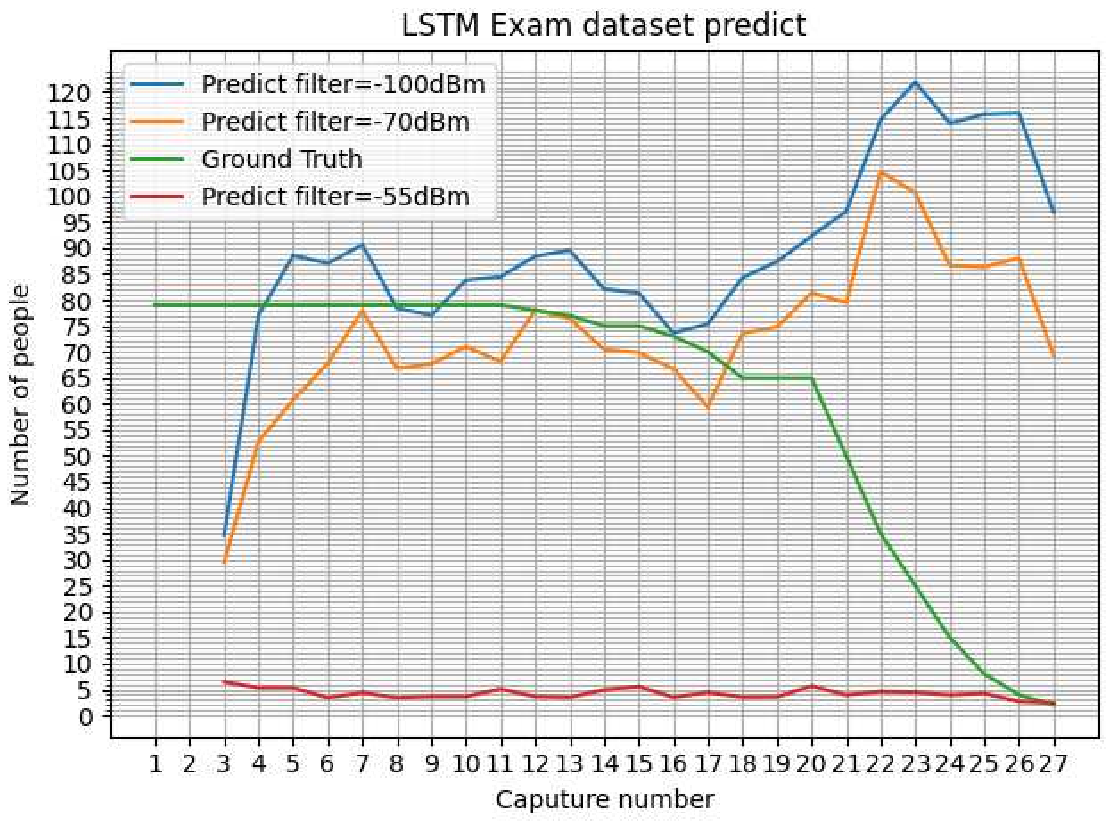


Figure 6.6: Comparison between ground truth and LSTM Network prediction with different power filter for Exam dataset.

from the test set was used as the estimation for the initial 2-minute capture, for which the ground truth is available.

In Figure 6.7, the results for both the LSTM network and FCNN are shown. The results reveal one of the most significant findings of this research: the LSTM network achieved an MSE of only 0.89, outperforming the FCNN.

However, it is important to note that the dataset was limited in size, and more extensive testing on larger datasets should be conducted to better assess the quality of the framework. Despite this limitation, the LSTM network demonstrates very promising results.

6.4 Main Takeaways

In this Section, the obtained results are discussed and analyzed to effectively present both the potential strengths and limitations of the framework.

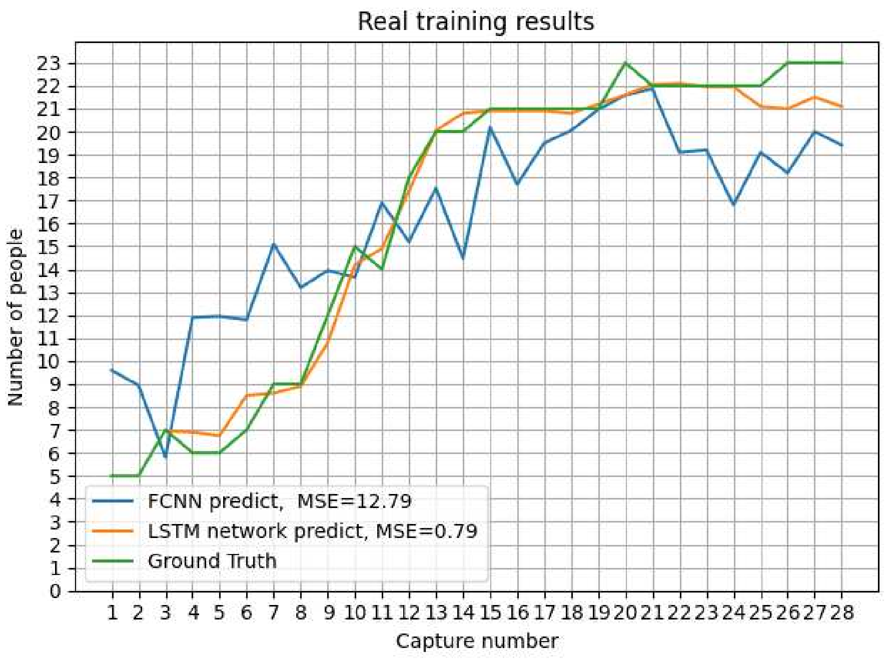


Figure 6.7: Prediction results of the regression model trained on real data from Lesson dataset.

6.4.1 FCNN Vs. LSTM network

In all the tests presented, the LSTM network has shown superior performance in respect to the FCNN. However, in tests based on real captures, using data from generator for training, the difference was not as pronounced. The most decisive results were instead observed in synthetic environment and real capture training tests.

This is certainly due to the greater complexity of the LSTM network and its ability to analyze multiple captures together, examining temporal correlations among them. This demonstrates the greater potential of this model for people counting task.

On the other hand, the FCNN has demonstrated good predictive capabilities, making it a viable alternative when temporal sequences of captures are not available.

6.4.2 Filter Effect

The filter had a positive effect on the regression and is crucial to use. This was especially evident in the Lesson Dataset where the models achieved very high accuracy. The excellent result in this dataset is also attributed to the fact that the capture location was well isolated from external interference.

The poor prediction results on the Exam Dataset demonstrate that relying solely on the power filter is not always sufficient. The inadequate prediction can be explained by the fact that in not isolated environments, such as the exam room, the power filter alone is insufficient to effectively filter captures. In this manner the regression framework analyzes packets outside the area of interest, resulting in undesirable predictions.

This suggests that there is still room for improvement in adapting the generator to real-world captures.

6.4.3 Real vs. Simulated Scenario

One of the main points highlighted is the difference in results between training with generated captures and training with real captures. A model trained on real captures offers the opportunity to learn better relationships among data, learning complex patterns that are not reproducible in a simulated environment. Additionally, real captures allow the model to utilize a broader range of more relevant features, favoring a deeper understanding of data relationships.

Although with a limited dataset, the research has demonstrated the potential of a model trained solely on real captures. Despite being highly challenging, the possibility of utilizing vast amounts of ground truth data, ideally from cameras, could enhance the quality of the proposed framework in many contexts. results.

6.4.4 Data Limitations

The datasets analyzed do not cover all possible scenarios and variations. Therefore, the results should be interpreted with caution and considered within the context of these limitations. The datasets may not fully represent all real-world conditions, which can affect the generalizability of the findings.

Nevertheless, these results provide a valuable overview of the framework's performance across various settings, offering insights into both its capabilities and limitations.

Moreover, the framework is not limited to the datasets and tests analyzed in this study. It remains adaptable and open to new datasets and testing scenarios. This flexibility ensures that the framework can continue to evolve and be applied effectively in diverse and evolving environments.

6.4.5 Enhancing Features

The features analyzed in this study represent only a limited subset of the possibilities available. The framework can be enhanced by incorporating numerous other relevant features, as the ones selected may not be sufficiently relevant. Introducing new features has the potential to significantly improve the framework's quality. In addition to information derived from WiFi probe requests in the captures, the framework could incorporate results from other people counting models. This integration could leverage the strengths of multiple models to enhance the framework's overall performance. For instance, the current implementation focuses solely on features of a specific clustering model, but it can be expanded to include a broader set of people counting models.

Enhancing the diversity and relevance of features is a critical aspect of improving the framework.

Chapter 7

Privacy

With the advent of the GDPR [1], the MAC address is considered personal data, as stated in Section 2.5. Consequently, any processing or storage of MAC addresses must comply with GDPR regulations. The GDPR states that all necessary measures should be taken to properly protect personal data, such as MAC addresses.

Crowd monitoring tasks based on the analysis of WiFi probe requests, inevitably use MAC addresses. Therefore, our research is deeply committed to achieving full compliance with the GDPR. Additionally, to deploy our research in Europe, compliance with the GDPR is mandatory, as it falls within the established terms.

In our people counting framework, MAC addresses are not directly stored in the features or used in the model. However, they are saved in the *.pcap* capture files. To avoid retaining the capture files, and therefore the MAC addresses, they should not be saved to the hard disk. Instead, the captures should be processed immediately after collection to extract the features for the model, storing only these features in memory. These features do not contain MAC addresses or other personal data but only numerical measurements from the capture. This approach ensures we do not fall within the scope defined by the GDPR.

However, for some specific scenarios such as a flow monitoring framework, saving the MAC address is needed, and strict compliance with the GDPR is essential. For this reason, an analysis was conducted on how to manage MAC addresses in accordance with GDPR regulations.

This Chapter illustrates the requirements imposed by the GDPR and the privacy issues related to the processing of MAC addresses in current solutions. Furthermore, it presents more efficient alternatives based on advanced privacy paradigms, such as Bloom filters and differential privacy.

7.1 Problems Of Current Solutions

The primary area of interest in our research is flow monitoring based on the analysis of WiFi probe requests. Most of flow monitoring systems implemented in the literature, track individuals' movements by analyzing MAC addresses and timestamps extracted from probe requests. When the same MAC address is detected at different locations and times, it is possible to trace the path of a specific device. By aggregating data from various monitoring devices, it is possible to retrieve the volume, patterns, and duration of people's movements. Since the storage and processing of this information include MAC addresses, this process must comply with GDPR regulations. The main solutions currently used for flow monitoring involve hashing and encrypting, but still present several issues.

7.1.1 Hashed MAC Address

The most common method implemented to protect MAC addresses is MAC hashing. *Hash* is a one-way function that transforms input data into a fixed-size string of characters, which appears random. The key property of an hash function is that it is infeasible to reverse the original input from the hashed output.

In this way, the data becomes pseudo-anonymized: it can be linked to a device but not reasonably to an individual. The original MAC address information is replaced with a representative token that identifies a device without revealing the personal information contained in the MAC address. Figure 7.1 provides an example of this process.

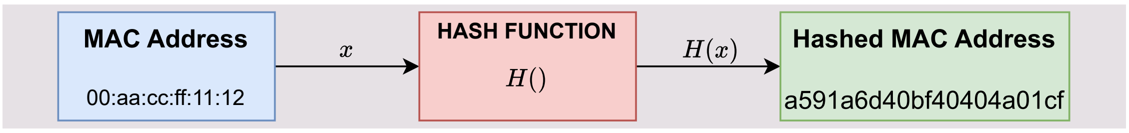


Figure 7.1: Reproduction of MAC address hashing process, with an example.

A fundamental aspect to ensure true randomization is the practical impossibility of retrieving the original value. To achieve this, very complex algorithms are used to generate the hashed MAC addresses, making the computational effort required to reverse-engineer them infeasible with current technologies.

Although a brute force approach is currently impractical, there is no guarantee that future technologies will not make this possible. This already represent an issue of these solutions.

Identification Of Known MAC Address

The first problem with the hashed MAC address approach is that, knowing a MAC address and the hashing algorithm, no matter how complex, it is possible to verify whether a given hashed MAC is compatible with the hypothesized initial MAC address. For this reason, it could be possible to test suspected MAC addresses. A positive result, if the identity of the device owner is known, would allow for the identification of the person, thus violating the privacy of the person.

A hashed MAC address is not uniquely associated with a single MAC address because two different MAC addresses could produce the same hashed value. This scenario, known as "collision", introduces uncertainty: even if the MAC address is known, it cannot be definitively linked to the hashed value, as there is a small probability it related to a different MAC address. To increase the likelihood of multiple MAC addresses sharing the same hashed value, hashed MAC addresses are often truncated. However, the probability of such collisions remains still very low, thus having minimal practical impact. Therefore, the possibility of verifying the presence of a specific MAC address remains a privacy concern, even with truncation.

To enhance the security of the hashing process and make it even more infeasible to restore the original MAC address, modern solutions use an additional layer of salt and encryption with keys during the creation of the hashed MAC. This means that to verify the presence of a specific MAC, it would also be needed to know the encryption keys and the salts used. This approach limits the presented problem, but in the event of a data breach or if an insider knows the encryption keys, it would still be possible to perform this kind of privacy violation.

Re-Identification Issues

The main problem with the hashed MAC address approach is that it results in pseudo-anonymized data, meaning it is not necessarily impossible to link the hashed MAC address to a physical person. If the hashed MAC address is stored along with other data, it is possible to use that additional information to identify an individual.

This issue was highlighted by the Dutch Data Protection Authority (DPA), regarding a flow monitoring project based on WiFi tracking, started by the municipality of Enschede in the centre of the city [34].

The project anonymized processed MAC addresses using truncated hashed MAC addresses. The Dutch DPA found that truncating the hashed MAC addresses did not adequately anonymize the data, as individuals could still be identified through timestamps and location information. Employees could identify individuals by observing who walked by a sensor, tracking the duration within the sensor range, or analyzing movement patterns across multiple sensors.

7.2 Proposed Privacy Solutions

As expressed by the Future of Privacy Forum [35], those who handle hashed MAC addresses, to overcome the problematics presented, must take measures to ensure the data cannot be re-identified, engage to maintaining data as de-personalized, and prohibit re-identification attempts. However, this approach is not simple and remains risky. Given that more advanced and secure solutions are available, they should be used. This Section presents two alternative approaches based on probabilistic solutions that offer stronger privacy capabilities: Bloom Filters and Differential Privacy.

These solutions are based on the idea of adding noise and uncertainty to the processed MAC addresses to ensure no possibility of re-identification, while not significantly compromising the flow information extracted from them.

7.2.1 Bloom Filter

A Bloom filter is a highly space-efficient probabilistic data structure used to represent a set. Bloom filters allow for testing whether an element is a member of a set or not. As a probabilistic structure, it permits false positive matches but not false negative matches: it can sometimes incorrectly indicate that an element is present in the set, but it will never falsely indicate that an element is not in the set.

A Bloom filter is represented as a bit array of length m bits. Initially, when the Bloom filter is empty, all bits are set to 0. Bloom filters employ k different hash functions, each of which hashes input elements to one of the m positions in the bit array.

Elements can be added to a Bloom filter by hashing the element multiple times using each of the k hash functions. Each hash function produces an index, and the corresponding bit in the bit array is set to 1. If the bit is already 1, it remains 1.

To check if an element y is in the Bloom filter, the element is hashed using the same k hash functions, and the bits at the corresponding positions are checked. If all bits are 1, the element is probably in the set. If at least one of these bits is 0, the element is definitely not present in the set. Figure 7.2 illustrate the process of insertion and presence check of an element in a bloom filter.

An example of research that uses Bloom filters is [36]. The research shows that to minimize false positives, the optimal value of k based on the available memory m can be analytically determined as: $k_{opt} = \frac{m}{n} \ln(2)$. Additionally, the number of elements

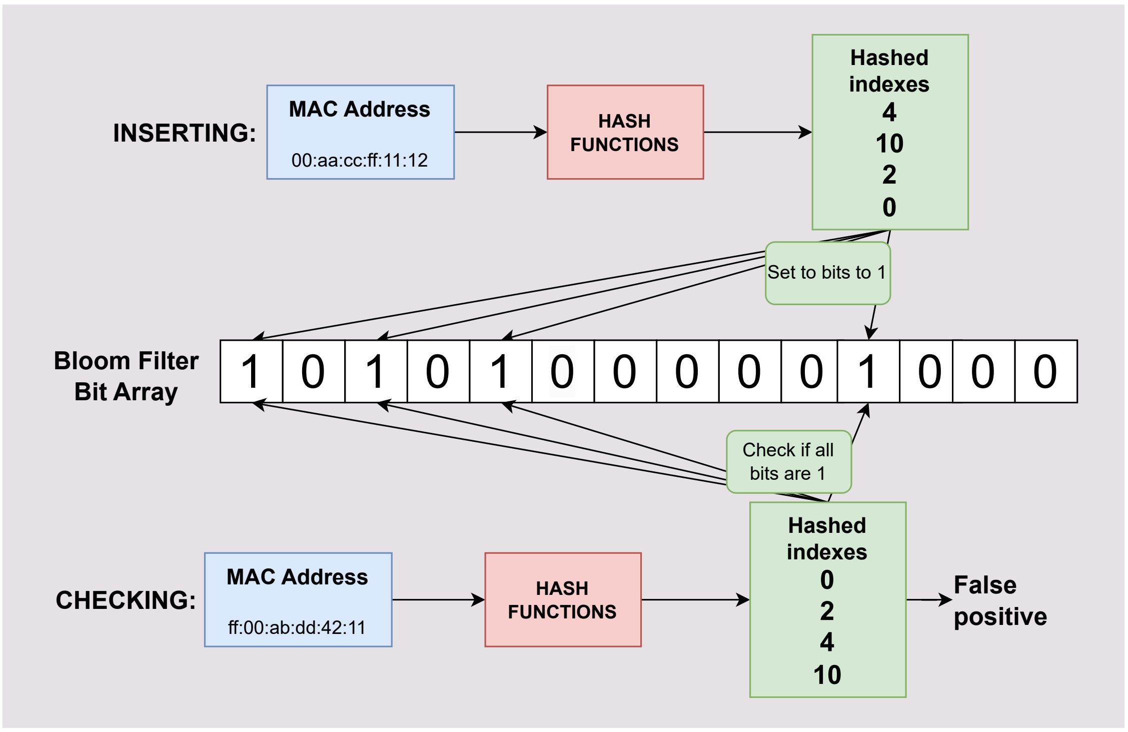


Figure 7.2: Representation of a Bloom filter, with an example of insertion and presence check that produces a false positive.

present within the Bloom filter (c) can be estimated as: $c = -\frac{m}{k} \log\left(1 - \frac{t}{m}\right)$ where t is the number of bits set to 1 in the Bloom filter.

Bloom filters can be used to represent a set of MAC addresses, which can be used for crowd monitoring tasks that require the storage of MAC addresses. To apply Bloom filters to flow monitoring, it is essential to estimate the number of common elements between two Bloom filters, i.e., the count of elements in their intersection.

To find the intersection between these, a bitwise logical AND operation is performed between the two Bloom filters. This operation produce a new Bloom filter: BF3 $=$ BF1 $\wedge$ BF2, which represents the intersection. Using this resultant Bloom filter, the count of elements within the intersection can be determined.

This method enables the calculation of the cardinality of the intersection of the sets represented by these Bloom filters. With this information, movement flow analytics can be computed. Furthermore, this technique can be expanded to determine the intersection of multiple Bloom filters, facilitating the identification of specific paths across a series of WiFi scanners.

Bloom Filter Benefits

The primary privacy advantage of using a Bloom filter-based approach is that it does not store MAC addresses directly but instead stores a probabilistic representation of them. In a Bloom filter, you can test for the presence of a specific MAC address, but due to the possibility of false positives, it could not be definitively associated with a particular MAC address. This concept is known as “deniability”.

γ-Deniability is a privacy measure applied to Bloom filters that allows for plausible deniability regarding the presence of an element in the Bloom filter with a controlled probability γ ($0 < \gamma \leq 1$). This is achieved by introducing randomness during the Bloom filter’s construction, ensuring that when querying an element, there is at least a γ probability that the element is deniable. For complete privacy, $\gamma = 1$ must be reached, meaning every possible element can be denied.

In the study [36], this is accomplished by adding anonymization noise during the Bloom filter’s initialization: n_{min} random MAC addresses are added to the empty Bloom filter. Based on the values of m and k, the sufficient value of n_{min}, to reach $\gamma = 1$, can be calculated.

This ensures that the presence of a specific MAC address cannot be confirmed, and certain re-identification is not possible. Combined with the fact that MAC addresses are not stored, this makes a properly configured Bloom filter fully compliant with GDPR.

7.2.2 Differential Privacy

Differential Privacy [37] is a rigorous mathematical framework designed to provide strong privacy guarantees when performing data analysis on datasets that contain sensitive information. It ensures that the inclusion or exclusion of a single individual’s data in a dataset does not significantly affect the outcome of any analysis, thereby protecting the individual’s privacy.

It defines a parameter ϵ (Privacy Parameter) that quantifies the privacy guarantee. It controls the trade-off between privacy and utility: a smaller ϵ provides stronger privacy but may reduce the accuracy of the analysis.

Mathematically, a randomized algorithm A is ϵ-differentially private if for all datasets D and D' differing in a single element, and for all possible outputs S of the algorithm:

$$\Pr[A(D) \in S] \leq e^{\epsilon} \cdot \Pr[A(D') \in S]$$

. This ensures that the presence or absence of any single individual’s data does not significantly change the probability of any outcome.

In some cases, a second parameter δ (additional parameter) is introduced, providing an additional relaxation to the privacy guarantee. This results in (ϵ, δ) differential privacy, allowing for a small probability δ of the privacy guarantee being violated: For all datasets D and D' differing in one element, and for all possible outputs S:

$$\Pr[A(D) \in S] \leq e^{\epsilon} \cdot \Pr[A(D') \in S] + \delta$$

To achieve differential privacy, mechanisms are employed to add noise to the analyzed data. Common mechanisms include the Laplace Mechanism, Gaussian Mechanism, and Exponential Mechanism. These mechanisms ensure that the noise added to the data is calibrated to reach the desired level of privacy parameters. In this way, differential privacy provides strong, mathematically provable, privacy guarantees without significantly compromising the utility of the data.

Proposed Solutions For People Counting

Given the issues associated with hashed MAC addresses, a flow monitoring solution based on a differential privacy framework could effectively leverage the benefits offered by differential privacy. Such an approach has not been explored in the literature and deserves further investigation.

A straightforward proposal to implement this, would involve adding noise perturbation to hashed MAC addresses to achieve differential privacy. For instance, one approach to perturbation could involve adding small random noise to truncated MAC addresses and introducing additional random MAC addresses into the original dataset.

This approach would reduce the certainty of associating a hashed value with a specific MAC address, while still allowing for flow information calculation based on the differences in hashed MAC addresses.

Implementing such solutions is not trivial, but considering the privacy advantages they offer, they should be thoroughly explored to develop systems with enhanced privacy protections.

Chapter 8

Future work & Conclusion

This Thesis has highlighted both the significant potential and the numerous challenges present in crowd monitoring. The results achieved provide a solid foundation for further research and development. More extensive work can be carried forward by training and testing the regression framework with more comprehensive datasets. Additionally, the proposed framework can be improved through the enhancing of the existing preprocessing phases, or with the inclusion of more advanced preprocessing techniques, such as adding better filtering methods. Another area for improving the proposed framework is the inclusion of additional features. This includes not only the extraction of more distinctive features, but also including as features the results from other crowd monitoring models. In this Thesis, only a clustering approach was incorporated, but many others can be added once implemented, to further enhance the framework. A crucial area is also privacy concerns. Further investigation for addressing privacy issues is related to the practical implementation of differential privacy approaches.

To sum up, this Thesis has demonstrated good results in various contexts, demonstrating the validity of the proposed regression approach. It has shown great results in both training with synthetic environments and real-world captures, demonstrating particularly high accuracy in the latter. Moreover, it has demonstrated the effectiveness and performance of neural networks models based on LSTM for crowd monitoring tasks, thanks to their ability to analyze temporal sequences of T consecutive captures. The results have been particularly excellent when a high number of consecutive captures is available. Further investigation into the benefits of LSTM models remains a promising area for future research. This exploration could improve the robustness and applicability of these models in diverse real-world scenarios.

The findings of this research have practical implications across multiple domains, from business insights to public safety management. There remains substantial room for improvement and enhancement, which can lead to even better results and broader applications in the future.

Bibliography

[1] European union gdpr.
URL https://gdpr.eu/tag/gdpr/

[2] Libelium website.
URL https://www.libelium.com/

[3] G-move services.
URL https://www.g-move.com/prodotti

[4] D. Gasco, Enhancing crowd-monitoring through wifi fingerprint analysis, available at https://webthesis.biblio.polito.it/28445/ (2023).

[5] Libelium meshlium iot gateway, [Online]. URL: https://www.libelium.com/libeliumworld/success-stories/libelium-iot-technology-helps-to-monitor-traffic-volume-in-petrol-stations-to-empower-sales-strategy/.

[6] Gmove device and web site, [Online]. URL: https://www.g-move.com/sensori.

[7] Wireshark website.
URL https://www.wireshark.org/

[8] Tshark manual.
URL https://www.wireshark.org/docs/man-pages/tshark.html

[9] tcpdump website.
URL https://www.tcpdump.org/

[10] Kismet website.
URL https://kismetwireless.net/

[11] aircrack-ng website.
URL https://www.aircrack-ng.org/

[12] M. Cunche, C. Matte, On wi-fi tracking and the pitfalls of mac address randomization, in: Proceedings of IDO 2016, 2016.

[13] R. Rusca, Mobility detection through electromagnetic fingerprints in 5g networks, available at https://webthesis.biblio.polito.it/15904/ (2020).

[14] C. Matte, Wi-fi tracking : Fingerprinting attacks and counter-measures, Phd thesis, UNIVERSITE DE LYON, Lyon, available at https://theses.hal.science/tel

-01921596/file/these.pdf (November 2018).

[15] Apple mac address randomization documentation.
URL https://support.apple.com/guide/security/wi-fi-privacy-secb9cb3140c/web

[16] Windows mac address randomization documentation.
URL https://support.microsoft.com/en-us/windows/how-to-use-random-hardware-addresses-in-windows-ac58de34-35fc-31ff-c650-823fc48eb1bc

[17] Networkmanager tool website.
URL https://networkmanager.dev/

[18] Android mac address randomization documentation.
URL https://source.android.com/docs/core/connect/wifi-mac-randomization-behavior

[19] Andorid 6.0 and mac address randomization.
URL https://developer.android.com/about/versions/marshmallow/android-6.0-changes

[20] Andorid 8.0 and mac address randomization.
URL https://source.android.com/docs/core/connect/wifi-mac-randomization

[21] Apple ios 8 and mac address randomization.
URL https://blogs.cisco.com/networking/apple-ios-8-and-mac-randomization-what-it-means-for-ciscos-connected-mobile-experiences-cmx-solution

[22] Z. Cai, Z. L. Yu, H. Liu, K. Zhang, Counting people in crowded scenes by video analyzing, in: 2014 9th IEEE Conference on Industrial Electronics and Applications, 2014, pp. 1841–1845. doi:10.1109/ICIEA.2014.6931467.

[23] J. W. Choi, X. Quan, S. H. Cho, Bi-directional passing people counting system based on ir-uwb radar sensors, IEEE Internet of Things Journal 5 (2) (2018) 512–522. doi:10.1109/JIOT.2017.2714181.

[24] L. Schauer, M. Werner, P. Marcus, Estimating crowd densities and pedestrian flows using wi-fi and bluetooth, in: Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MOBIQUITOUS '14, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 2014, pp. 171–177. doi:10.4108/icst.mobiquitous.2014.257870.
URL https://doi.org/10.4108/icst.mobiquitous.2014.257870

[25] K. Gebru, M. Rapelli, R. Rusca, C. Casetti, C. F. Chiasserini, P. Giaccone, Edge-based passive crowd monitoring through wifi beacons, Computer Communications

192 (2022) 163–170. doi:https://doi.org/10.1016/j.comcom.2022.06.003.
URL https://www.sciencedirect.com/science/article/pii/S0140366422001980

[26] X. Gu, W. Wu, X. Gu, Z. Ling, M. Yang, A. Song, Probe request based device identification attack and defense, Sensors 20 (16) (2020). doi:10.3390/s20164620.
URL https://www.mdpi.com/1424-8220/20/16/4620

[27] L. Pintor, L. Atzori, Analysis of wi-fi probe requests towards information element fingerprinting, in: GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 2022, pp. 3857–3862. doi:10.1109/GLOBECOM48099.2022.10001618.

[28] R. Rusca, F. Sansoldo, C. Casetti, P. Giaccone, What wifi probe requests can tell you, in: 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), 2023, pp. 1086–1091. doi:10.1109/CCNC51644.2023.10060447.

[29] T. Bravenec, J. Torres-Sospedra, M. Gould, T. Fryza, Uji probes revisited: Deeper dive into the dataset of wi-fi probe requests, IEEE Journal of Indoor and Seamless Positioning and Navigation 1 (2023) 221–230. doi:10.1109/JISPIN.2023.3335882.

[30] Probe request generator.
URL https://github.com/Diegomangasco/ProbeRequestGenerator

[31] R. Rusca, Crowd monitoring and city sensing techniques supported by next generation mobile networks, Phd thesis, Politecnico di Torino, Turin, TO, available at https://iris.polito.it/handle/11583/2987572 (March 2024).

[32] Probe request sniffing.
URL https://github.com/riccardo-rusca/ProbeRequestSniffing

[33] Mobile devices market share.
URL https://gs.statcounter.com/vendor-market-share/mobile

[34] Dutch dpa fines municipality for wi-fi tracking.
URL https://www.edpb.europa.eu/news/national-news/2021/dutch-dpa-fines-municipality-wi-fi-tracking_en

[35] Future of privacy forum on mac address hashing.
URL https://fpf.org/blog/mac-addresses-and-de-identification/

[36] R. Rusca, A. Carluccio, C. Casetti, P. Giaccone, Privacy-preserving wifi-based crowd monitoring, Transactions on Emerging Telecommunications Technologies 35 (3) (2024) e4956. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.4956, doi:https://doi.org/10.1002/ett.4956.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4956

[37] C. Dwork, Differential privacy, in: Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 1–12.