Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

MASTER THESIS

TITLE: Machine Learning for Localization in Narrowband IoT Networks

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Jimmy Xavier Fernández Landivar

SUPERVISORS: Prof. Sofie Pollin - Prof. Carles Gomez Montenegro

DIRECTOR: Dr. ir. Hazem Sallouha

DATE: September 14, 2021

Title: Machine Learning for Localization in Narrowband IoT Networks
Author: Jimmy Xavier Fernández Landivar

Supervisors: Prof. Sofie Pollin - Prof. Carles Gomez Montenegro

Director: Dr. ir. Hazem Sallouha

Date: September 14, 2021

Abstract

Low power wide area networks (LPWANs) are designed for Internet of Things (IoT) applications because of their long-range coverage, low bit rate, and low battery consumption. In the LPWAN networks, Narrow-band IoT (NB-IoT) is a type of network that uses the licensed cellular spectrum, working over the deployed LTE infrastructure. It is rising as a promising technology because of its characteristics and deployment advantages against other LPWAN networks. In NB-IoT networks, localization is an essential service for applications such as smart cities, traffic control, logistics tracking, and others. The outdoor localization is often performed using a Global Navigation Satellite System (GNSS) like Global Positioning System (GPS) to send the current device position with some meters accuracy. However, due to GPS's power and size drawbacks, recent reports focus on alternatives to replace GPS-based localization systems with cost and power efficient solutions.

This work analyses a database collected over an NB-IoT deployed network in the city of Antwerp in Belgium and implements a solution for outdoor localization based on Machine Learning (ML) methods for distance estimation. The data analysis starts in the pre-processing step, where the databases are cleaned and prepared for the ML analysis. The following process merges and debugs the data to obtain an integrated database with classification for urban and rural areas. The localization solution performs a support vector regression, random forest regression, and a multi-layer perceptron regression using as input parameters the received signal strength indicator (RSSI) and the base station (BS) position details in order to predict the distance to the IoT nodes and estimate the current position (latitude and longitude) of them. This implementation includes hyper-parameter tuning, the train and test process, and mathematical calculations to obtain the estimated position with mean and median location estimation errors expressed in meters.

The implementation of the methodology processes results in 280 and 220 meters corresponding to the mean and median location errors for the urban area and 920 and 570 meters for the rural area. The accuracy levels obtained in the results turn this solution suitable for the most common uses of localization in IoT instead of using a GPS device. As a result, this study proposes a new approach for localization in IoT networks. In addition to the implemented solution defines valuable research lines to improve the accuracy levels and generate more contributions to optimize the equipment resources and reduce the IoT device's final cost.

First and foremost, I would like to thank God for all the blessings he has given me to continue with my objectives and goals. I would also like to extend my deepest gratitude to my thesis advisors Sofie Pollin, Hazem Sallouha, and Carles Gomez, for their guidance, experience, and support during the time I conducted this study. Special thanks to my country and the Secretariat of Higher Education, Science, Technology, and Innovation for the opportunity of having a grant scholarship to follow my master's studies in a prestigious international university. Finally, I would like to hugely thank my family because this objective would not have been possible without their support and nurturing. They are who will always be there for me and to whom I dedicate this work.

Jimmy.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

The internet of things (IoT) is an essential emerging technology due to its novel objective of interconnecting all devices to contribute to automating, sophisticating and improving the quality of life of people. As a result of its growing, it is expected to have twenty-two billion connected devices by the end of 2025 [1]. IoT devices are suitable for different applications, starting from home and farming to critical industry measurements [2]. More specific examples involve traffic control and smart cities, supply chain management, logistics tracking, and others. Low power wide area networks (LPWAN) are designed for IoT applications. Within this group of technologies, the 3rd Generation Partnership Project (3GPP) introduced Narrow-band IoT (NB-IoT) networks as a technology that, unlike SigFox and LoRaWAN, uses the licensed spectrum of cellular networks. Hence, NB-IoT can be installed over an LTE infrastructure, offering an IoT network with extended coverage and reducing costs of deployment, as well as optimizing the battery consumption and computational process [3].

In IoT networks, the devices used in smart cities, logistics, autonomous driving, and others, are constantly in motion. This fact makes localization an essential service necessary for meaningful data collection, which has a direct influence on the proper performance of the applications mentioned above [4]. For obtaining the current position in IoT outdoor applications, a Global Navigation Satellite System (GNSS) like Global Positioning System (GPS) is used to send the location with an error of less than ten meters. This approach brings the highest accuracy, but it is the most expensive solution in terms of energy consumption required by GPS devices and for sending the message with the current location [5]. Also, GPS devices involve additional costs to the final IoT module. For this reason, finding alternatives for localization in IoT networks that optimize energy consumption and resources use represent a fundamental challenge for researchers, where innovative techniques and methods are used, including proximity, path loss models and statistical and Machine Learning (ML) algorithms.

Over the past decade, machine learning (ML) has been widely used in wireless networks, offering a powerful alternative to perform complex signal processing tasks efficiently. Therefore, exploiting the advances in ML, many state-of-the-art works are using and combining it with distance estimation methods to predict localization points in IoT networks. The methods for distance estimation between the BS and the devices are based on received signal strength (RSS) and Time Difference of Arrival (TDoA) [6]. In some works on indoor localization, ML methods are combined with channel state information (CSI) and received signal strength indicator (RSSI) [7] [8]. For outdoor localization, there are other contributions using ML, Multilateration and Path Loss estimation techniques, getting prediction results with accurate levels that are suitable for most of the common uses of localization on IoT [9] [10].

In LPWAN time-based techniques are extensively studied. The work performed by [11], [12], and [13] present experiments based on observed time difference of arrival (OTDoA) for distance estimation. At this point, it is worth mentioning that Belgian telecom operators have not implemented OTDoA in the already deployed IoT networks [6]. This fact turns the effort on working in the use of range-based models, and subsequent contrast with Machine Learning techniques, examples of this approach used in LPWAN are present in [9], [14] and [15]. These experiences use ML techniques like Support Vector Regression,

Random Forest Regression, K Nearest Neighbors (KNN), and supervised ML methods to implement localization algorithms. The findings of these investigations include novel proposals for distance estimation [9], error comparison on the applied techniques [15], and solid guidelines for conduct further experiments using ML [10] [14]. However, filtering the works performed over SigFox and LoRa to focus on NB-IoT (released on 2016), the research on localization techniques is lower compared with the other technologies, but it is growing simultaneously with this network use for IoT applications.

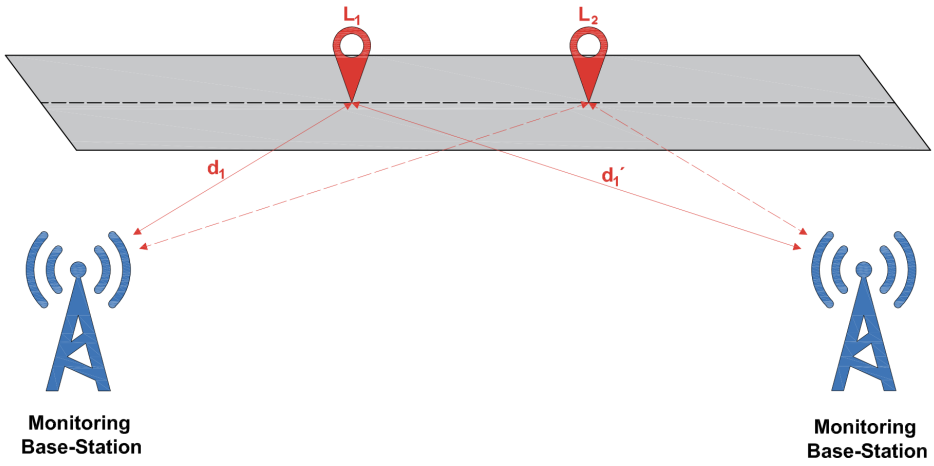


Figure 1.1: NB-IoT nodes positioning for tracking applications.

This master thesis project proposes a new alternative for localization in NB-IoT networks, specifically for outdoor localization that could be applicable in scenarios that use tracking services, as illustrated in figure 1.1. Thus, the analysis is carried out using data presented in [6], which corresponds to an NB-IoT messages database collected in the city of Antwerp in Belgium.

The proposed solution for localization combines ML algorithms to predict the distance between the IoT nodes and the BS, based on the RSSI values of the messages. The ML algorithms used are Support Vector Machines, Random Forest and Neural Networks. These algorithms perform regression models for value predictions, where these predicted values correspond to the mentioned distance between the base station (BS) and the NB-IoT nodes. Finally, the proposed solution predicts the node position (latitude and longitude) from the estimated distance and the BS Azimuth angle. Therefore, this solution can be allocated into multilateration techniques for distance estimation on wireless networks.

The results obtained from this research show a mean and median estimated error in urban areas of 280 meters and 210 meters, respectively. On the other hand, the rural area's mean and median estimation errors are 920 meters and 556 meters, respectively. These results outperform the conventional proximity and ranging methods carried out in the related work. However, the accuracy difference between the fingerprinting algorithms used in the related bibliography and the proposed solution in this work is roughly 80 meters. To conclude, the contribution of this master thesis opens new hypotheses and branches for further research to predict the node's location. Future work will use ML, combined with traditional ranging methods and adding different characteristics as angle predictions for the beaming direction.

This thesis document is organized as follows. Chapter number 2 presents the literature review about LPWAN and NB-IoT networks. Section one explains NB-IoT technology, detailing its core characteristics, including the low device complexity and cost, enhancements for extended coverage and device battery time, and massive deployment and support for these devices. Later, it explains NB-IoT network architecture, its deployment, Uplink (UL) and Downlink (DL) processes, and finally, the localization techniques for NB-IoT. Section two introduces ML theory and its application for regression models, covering the essential concepts of Support Vector Regression, Random Forest Regression, and Multi-layer Perceptron Regression.

Chapter number 3 focuses on the methodology of this project. Sections one and two explain the NB-IoT scenario of Antwerp city and the data collection and storage process. After that, section three shows the whole process carried out to prepare the data for analysis with the different algorithms. This pre-processing includes the merging, debugging, and the calculation and addition of the necessary variables prior to the feature selection and the regression processes. Section 4 covers the algorithm implementation; it includes the feature selection and the entire process of algorithm training, hyper-parameter tuning, regression results, and the calculus of the mean error for the predicted location points. Finally, the algorithm implementation process is explained for each one of the three selected algorithms.

Finally, chapter number 4 presents the results and discussion of this research. First, the results are illustrated according to the Urban and Rural distribution of the database. Then, the implemented algorithms are compared using statistical graphs and performance tables, depicting the main differences. Finally, the discussion includes the most important findings of all the processes performed., including the accuracy results and the established hypothesis for future work. All the used algorithms are attached at the end of this document as annexes.

CHAPTER 2. LITERATURE REVIEW

2.1. Narrowband IoT

Low-Power Wide-Area Networks (LPWANs) represent a substantial contribution to the Internet of Things, allowing the use of a large number of low-cost and low-throughput devices suitable for industrial and home applications [16]. The principal characteristics of these networks are the large area coverage and low energy consumption, which are enabled by small packet application-layer data sizes, that covering the necessities of the required applications [17] [16]. Sigfox, LoRaWan Alliance, and 3GPP lead the standards for the different LPWAN technologies [18]. The principal difference between these brands is the use of licensed and unlicensed frequencies, the bandwidth and data rate, the range of coverage, type of communication, and other characteristics presented in Table 2.1. On one hand, unlicensed spectrum corresponds to the portions of the public frequency space free of licensing costs; Sigfox, LoRaWAN, and similar technologies are part of this frequency standard. On the other hand, the licensed frequencies are part of the public frequency space regulated by national authorities; 3GPP standards such as LTE-M and NB-IoT are part of this group [18].

Characteristic	**Sigfox**	**LoRaWAN**	**NB-IoT**
Modulation	BPSK	CSS	QPSK
Frequency	Unlicensed	Unlicensed	Licensed (LTE)
Bandwidth	100 Hz	250-125 kHz	180 kHz
Max data rate	100 bps	50 kbps	200 kbps
Bidirectional	Limited/HalfDuplex	Yes/HalfDuplex	Yes/HalfDuplex
Range	10 km urban 40 km rural	5 km urban 20 km rural	1 km urban 10 km rural
Max payload length	12/8 bytes	243 bytes	1600 bytes
Auth Encryption	Not supported	AES 128b	LTE Encryption
Localization	RSSI	TDOA	No(Under specification)
Standard	Sigfox-ETSI	LoRa-Alliance	3GPP

Table 2.1: Principal characteristics of LPWAN technologies.

Narrowband IoT (NB-IoT) was introduced in early 2015 by 3GPP in release 13, updating the EGPRS standard for GSM and proposing the new one for LTE networks. This standard presents new physical channel layers and up to 180 kHz bandwidth [18]. This technology offers low cost and low complexity for IoT devices, indoor coverage improvement, and massive low throughput devices. Release number 13 of 3GPP was finished in June 2016 [16], but further work was presented in releases 14 and 15, and finished in June 2018. The new improvements on releases 14 and 15 are positioning enhancements based on Cell-ID (E-CID) and Observed Time Difference on Arrival (OTDOA), higher data rates for the user equipment (UE), and latency and power consumption reduction. These characteristics are achieved because of techniques such as a wake-up channel and data transmission on random access procedures. In this context, NB-IoT has become a good choice for IoT over LTE networks, and it has became the predominant technology compared to other

standards for these applications as LTE-M [19]. Since 2017, in Belgium, telecom operators have deployed the NB-IoT network using the recommendations of release 13 [20] [6]. Release 13 is further explained in the following subsections to emphasize the relevant theoretical parts for the understanding of this project.

2.1.1. Technology Characteristics

The aim of designing NB-IoT is to cover the emerging massive machine type communications necessity, support multiple devices, and improve technical characteristics for low energy and data consumption [18]. However, another essential characteristic to offer is the long-distance covering, which provides facilities on the network deployment. This subsection will introduce these characteristics, including some radio access design principles, to explain why this technology is one of the most popular cellular IoT technologies [17].

2.1.1.1. Low device complexity and cost

The NB-IoT devices improve the baseband processing, memory consumption, and radio frequency (RF) for low cost and complexity devices. On the baseband processing, the improvement is evident when the device only uses a synchronization sequence for establishing primary time and frequency synchronization to the network, improving the cell selection and connection resources [18].

The memory improvement reduces the complexity of the device with a downlink (DL) transport block of 680 bits (Rel 13) and also relaxes the processing time requirements compared with LTE [16]. Regarding RF, a remarkable improvement is the presence of a unique antenna for DL and uplink (UL), achieving the objectives of the technology standard because it is not necessary to transmit and receive messages simultaneously [18]. These technical characteristics significantly reduce the device's complexity, allowing cheap construction and replicability, reaching a cost below 4 € for a target device [19]. Furthermore, the necessity of a large number of devices contributes to cost reduction, becoming suitable for mass production, which directly reduces the final prices per device.

2.1.1.2. Coverage Enhancement

The coverage enhancement on NB-IoT is achieved as a consequence of the trade-off between data range for coverage [18]. Comparing the previous technology standard EC-GSM and the parallel standard LTE-M in which repetitions are used to ensure that the devices are in coverage, NB-IoT has been designed for using a close to constant envelope waveform in the uplink (UL) [17] [18]. This improvement optimizes the coverage in limited power situations, suppressing the necessity of back off the output power to the maximum configurable level [18], and at the same time risks data range, reducing it to ensure the extended coverage.

2.1.1.3. Long device battery time

One keystone of this technology is low energy consumption. It starts minimizing the power back off, resulting in increased PA power efficiency [18]. Nevertheless, the device's battery

life depends on how much energy it uses when it does not have a data session. The significant improvement rises on the listening time, introducing extended Discontinuous reception (eDrx) and Power Saving Mode (PSM). With these innovative power modes, the device can shut down its transceiver and keep running only an oscillator which manages when it turns on the eDrx or PSM. On eRdx the cycle could be up top three hours. However, in PSM the device is a deep sleep state, waking up only for UL transmission, improving battery save to the maximum [21]. In addition, NB-IoT has the connected mode (DRX). The mix of these different modes can reach a significant battery life of up to 10 years in some applications as farming or not real time industry measurements [22].

2.1.1.4. Massive device supporting and deployment flexibility

One single NB-IoT household can support up to 40 devices and achieve up to 55k devices per cell [19]. A massive device supporting is possible due to the efficient spectral transmission in UL for the devices in extreme coverage-limited situations [21]. The different options on NB-IoT bandwidth for UL start on a wide bandwidth which is beneficial for devices with good coverage, and small bandwidths for devices within low coverage. The maximum coverage allowed for this technology is 164dB [16]. Additionally, the operation mode supports a Half Duplex Frequency Division Duplex Operation with a bit rate of 60 kbit/sec for UL and 30 kbit/sec for DL. The maximum transmission unit (MTU) could be up to 1600 bytes [16][19].

In terms of flexibility, NB-IoT supports three modes of operation: Stand-alone, In-band, and Guard band. Switching between these three modes allows it to allocate a large number of NB-IoT channels, resulting in maximum flexibility on deployment. This differentiation enables introducing a significant number of NB-IoT devices in the network, achieving massive low bandwidth communications [21].

2.1.2. Network Architecture

NB-IoT runs under an LTE Network, which means that the topology and the transmission schemes are equal to LTE under some conditions. These conditions are the communication scheme, the network deployment, and the technical characteristics, which reduces the data rate and the battery consumption allowing NB-IoT to be applicable for low-cost IoT [16][18].

Figure 2.1 illustrates the 3GPP network architecture of NB-IoT, where the UE corresponds to the different IoT devices. Next in this topology is the connection to the eNodeBs that are the same base stations used for an LTE network deployed for cellular mobile communications [16]. At the same level, the mobility management entity (MME) is responsible for handling the mobility process for the UE through paging the UEs, managing the sessions, and choosing the correct service gateway (S-GW) for the UE [18][21].

After the eNodeB, the S-GW routes the different user data packets towards the access network. Its function during the handover maintains the connection between eNodeBs and UEs, and between NB-IoT and other 3GPP technologies [19]. Finally, the packet data network gateway is the interface between the network and other external ones. Coexisting at the same level, the home subscribed server (HSS) contains all the user and subscription-related information, helping the mobility management, user authentication, and session

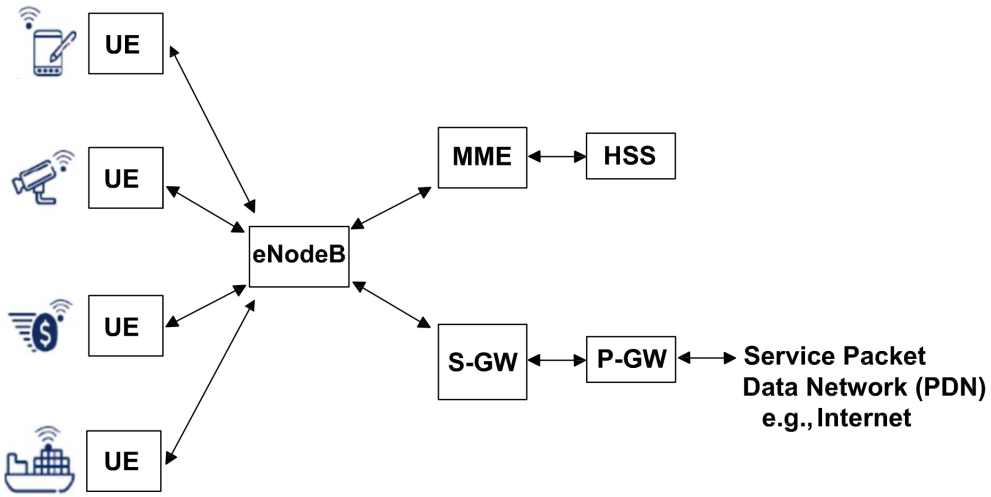


Figure 2.1: Narrowband IoT topology.

establishment [21].

2.1.2.1. Network deployment

According to the technical specifications, NB-IoT uses a frequency band of 180 kHz, which corresponds to one resource block (RB) of LTE [16]. According to this information, the deployment could be performed in three operational modes. The first one is the "Standalone operation," where the NB-IoT RB goes within a GSM carrier, illustrated in Figure 2.2 c). For this operation mode, a recommended guard band of at least 100 kHz has to be inserted between the NB-IoT RB and the GSM carrier, where the new RB is in the space of the refarmed carrier of GSM for this technology [18] [23].

The second and third modes were specially designed for the deployment of NB-IoT into the existing LTE network. First, the "In-Band operation", illustrated in Figure 2.2 a), consists of inserting the NB-IoT RB into the LTE carrier instead of one of the LTE RBs. This in-band deployment could establish the new RB as a different block in the carrier or receive the NB-IoT RB as part of the LTE carrier in one that supports LTE-M features and has a prepared space for the NB-IoT RB. However, it means that the used RB was not designed for a system information block of LTE until LTE-M creates a dedicated one for the NB-IoT RB [18] [19].

Finally, the "Guard-band" operation mode, illustrated in Figure 2.2 b), takes advantage of the fact that LTE carriers have occupied 90% of the channel bandwidth in most cases, assigning the remaining 10% for the Guard-band. The NB-IoT resource block is added to this Guard-band space on each available side of the carrier next to the LTE resource blocks [19] [21].

2.1.2.2. Uplink process in NB-IoT

In the UL operation process, NB-IoT uses Single-Carrier Frequency-Division Multiple-Access (SC-FDMA) with a subcarrier spacing of 15 kHz or 3.75 kHz depending whether

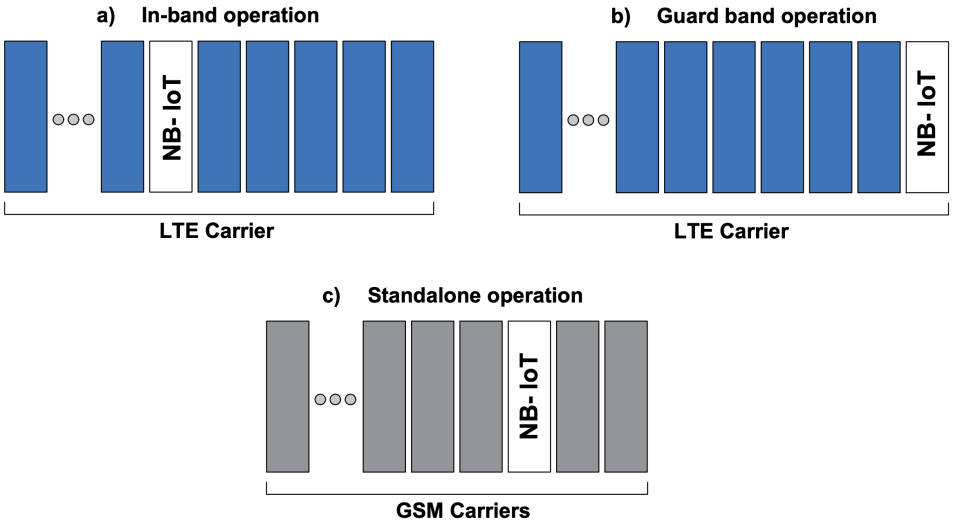


Figure 2.2: Narrowband IoT operational modes.

it is for multi-tone or single-tone transmissions [21]. It is noteworthy that SC-FDMA with a single tone is identical to OFDM, used for DL operation. Also, the User Equipment could have 1, 3, 6, or 12 tones out the whole RB [16].

There are two physical channels in the UL process: Narrowband Physical Uplink Shared Channel (NPUSCH) and the Narrowband Physical Random-Access Channel (NPRACH). NPUSCH is used in the same principle as LTE PUSCH, sending the data symbols as a set of frequency-domain complex symbols [23]. NPRACH as the random access channel establishes the request of communication between the NB-IoT device and the eNodeB [18]. Figure 2.3 illustrates the simplified procedure for UL in NB-IoT.

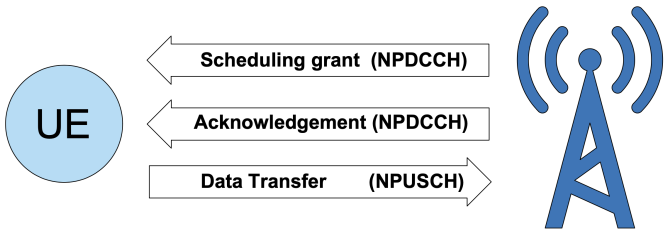


Figure 2.3: Uplink process scheme in NB-IoT.

2.1.2.3. Downlink Process in NB-IoT

For DL operation, NB-IoT employs Orthogonal Frequency Division Multiple Access (OFDMA) with the same characteristics as LTE in terms of subcarrier spacing, slot, sub- frame, among others [22]. The DL waveform has 12 subcarriers and is the same for the three operating modes (stand-alone, in-band, and guard-band) [18]. Also, the DL supports one or two antenna ports with modulated symbols based on transmit diversity using Space-Frequency Block [19].

The physical channels for DL in NB-IoT are time-multiplexed, except for Narrow- band Reference Signal(NRS) [18]. The three physical channels for DL are Physical Broadcast Channel (NPBCH), Narrowband Physical Downlink Control Channel (NPDCCH), and Narrowband Physical Downlink Shared Channel (NPDSCH) [21]. Figure 2.4 illustrates the simplified procedure for DL in NB-IoT.

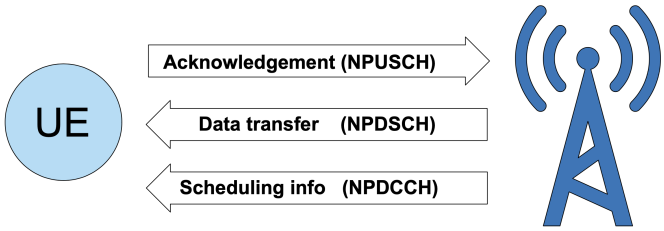


Figure 2.4: Downlink process scheme in NB-IoT.

Figure 2.5 shows all the UL and DL potential channels for NB-IoT. It is important to highlight there is no UL control channel in NB-IoT, instead,the NB-PUSCH transmits the UL acknowledgment. The NBPUSCH channel can occupy less than 1 RB of an LTE system, supporting the single tone and multi-tone transmission, which denotes the space-saving for this technology.

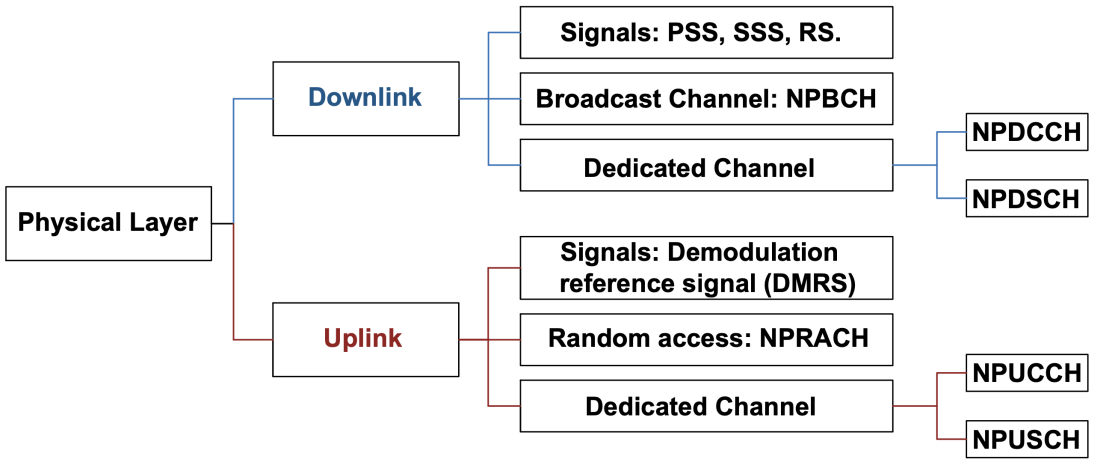


Figure 2.5: Up-link and Down-link channels in NB-IoT.

2.1.3. Localization in NB-IoT

The localization process in NB-IoT depends on the application for the different devices and, therefore, the required level of accuracy. The most reliable localization technique uses GPS signals sent by the device in a simple message to the eNodeBs. Although considering a GPS device to be used with NB-IoT is the best option in terms of accuracy, it is the last option in terms of energy consumption since it uses a message just for sending the location, reducing the power of the battery [19]. The second option for localization

in NB-IoT is the estimation using multilateration techniques such as the Observed Time Difference of Arrival (OTDOA) measurements, available since the release 14 of 3GPP [23]. Another technique uses the Received Signal Strength Indication (RSSI) that represents the transmitted power to the BS to estimate the possible location. Based on the transmitted power, the path loss estimation process is a technique that calculates the distance from the link budget, based on the power received, determining the distance between the device and the base station eNodeB [6]. Other RSSI-based techniques are triangulation and distance estimation which are not the most accurate but are the most efficient in terms of energy consumption and results suitable for the most of the outdoor NB-IoT applications [6].

2.1.3.1. GPS Signals

According to the study realized by Ahmed Elhaddad in [22], an estimate of the battery consumption sending a GPS location from a NB-IoT device could be around 500 mW, assuming a 44% PA efficiency and without counting the GPS Battery consumption. Therefore, sending the GPS Location each time for an NB-IoT device represents a significant increment in energy consumption. Also, GPS is not accurate for indoor estimations [11]. Concluding, GPS is the most suitable solution for estimate location outdoor with precision, but with a high energy consumption cost [11]. On the other side, the OTDOA approach needs accuracy and synchronization of the eNodeBs, which turns into more complexity and higher costs for the implementation in terms of the telecom operators [6].

2.1.3.2. Path Loss Models

The path loss estimation models consider several characteristics and variables including the altitude of the base station, the place location (it is crucial if it is urban or rural), the transmission power, and the frequency. For example, in NB-IoT, the frequency could be considered between 800 - 900 MHz [6]. Therefore, it is possible to use several path loss models on this specific objective; some of these models are The Hata Model, The Cost 231 model, The AH-macro, and the AH micro model, among others [6].

2.1.3.3. Multilateration Techniques

Release 14 of 3GPP presents location services based on cell-ID and OTDOA. The first solution estimates the device's location based on the geographical coordinates of the serving cell [23] [11]. On the other hand, OTDOA measures the arrival time on the reference signals in some points to estimate the difference between these times and then the location [6]. These solutions represent an advance in terms of energy reduction and optimization.

Currently, in Belgium, the telecom operators have not yet implemented OTDOA to increase the technical costs that it represents. Also, for the Belgium network configuration, the nodes establish a connection and data sending with only one base station simultaneously. This configuration is generated to reduce battery consumption and hardware costs, optimizing the resources [6]. The last reasons represent difficulties in implementing the triangulation-based distance estimation because it is impossible to obtain information for more than one database simultaneously and triangulate the position [6] [23]. Instead, it is possible to obtain the RSSI value from the NB-IoT device, allowing the implementation

of a distance estimation based on the power intensity of the sent message. Starting with the location of the base station (eNodeB) as the experiment performed and proposed on [6]. Also, having an extensive database with the locations and the RSSI values of the sent messages from the IoT devices, it is possible to estimate the distance using regression techniques in a 120 degrees sector from the BS.

The Figure 2.6 illustrates how to calculate the distance from the RSSI value and allocated it into the 120 degrees sector of the eNodeB to generate a possible location of the IoT device. The following subsection will explain how the Machine Learning techniques for regression would be suitable for this distance estimation process.

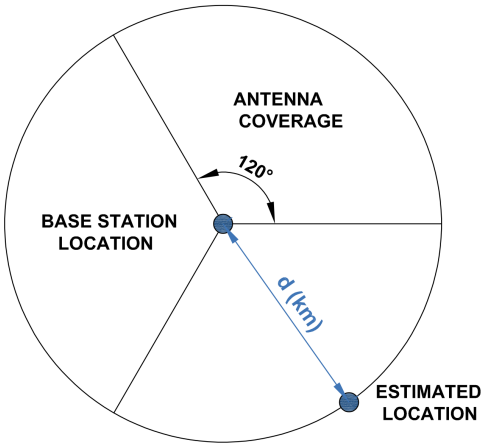


Figure 2.6: Distance estimation between UE and eNodeB based on RSSI.

2.2. Machine Learning Regression Models

Machine Learning (ML) is the terminology used to describe the group of techniques and methods to use the computer and machine capacities to solve problems based on the use of data. ML becomes part of Artificial Intelligence (AI), which principal objective is to provide solutions for different applications as prediction, data mining, optimization, generalization, and statistics problems [24].

Into the ML solutions for statistic problems, there is a significant group of techniques for regression analysis. ML regression techniques aim to provide different value predictions based on a training data set with specific characteristics that depend on each problem where these are used. The regression analysis with ML starts from basic statistical techniques like linear and multi-variant regression up to more complex algorithms such as Neural Networks (NN) [25] [26].

The objective of this master thesis is to use the different tools and techniques of ML for statistic regression analysis and apply them to the prediction of the distance between a base station (eNodeB) and an NB-IoT device, using the technique explained in the previous section and illustrated in the figure 2.6. To accomplish these objectives for this work have been selected three of the most relevant regression models in ML: Support Vector Regression, Random Forest Regression, and Multilayer perceptron regression. The fol-

lowing subsections will cover the main characteristics and a general overview of these techniques.

2.2.1. Support Vector Regressor

Among ML methods, the Support Vector Machines (SVM) is a supervised method that represents the construction of a hyper-plane in an infinite-dimensional space, depending on the complexity of the problem, where some predictions are achieved based on the border of these hyper-planes [27]. The gap established by these hyper-planes and the margin vectors defines the generalization error of the classifier or the regressor. The shortest gap means the lowest generalization error in the different value predictions [28].

SVM can be used for regression methods when the problem aims not to classify the features. Instead, the objective is to predict a value between the hyper-planes and the support vectors according to a set maximum error value, epsilon ε [27]. As a result, a Support Vector Regressor (SVR) represents the solution for some statistic regression problems. For example, equation 2.1 represents the primary math expression for a linear hyper-plane, which explains the support vectors used in a regression problem [29] [30].

$$f(x) = w * x + b \tag{2.1}$$

Next, equations 2.2 and 2.3 represent the constraints $C1$ and $C2$ for the support vectors, limited by the epsilon ε value. These constraints control the accuracy of the model. If the epsilon value is set near the hyper-plane, it represents high accuracy for the regression model [27] [31].

$$C1(x) = w * x + b + \varepsilon \tag{2.2}$$

$$C2(x) = w * x + b - \varepsilon \tag{2.3}$$

Finally, Figure 2.7 illustrates the different parts of the support vector regression, showing in red the different data points, the constraints $C1$ and $C2$, and the hyper-plane $f(x)$. Here the two side vectors are adjusted according to the accuracy level for the solution, and the phi value (ϕ) represents the tolerance between the vectors and the outsider data values to be predicted [27] [31].

For complex solutions, the hyper-plane finding represents a challenge in adding more dimensions to the SVR. A kernel transforms the different dimensions of the problem from non-linear to linear [31]. The complexity of the SVR depends on the number and the type of features in the input data. These characteristics of the input data could need extra processing before the input of the ML model. The management of the data for this project will include some supplemental processes described in detail in chapter number 3.

2.2.2. Random Forest Regressor

A decision tree is a powerful tool for supervised methods for classification on ML, which uses the tree shape to model possible consequences and classes into a solution. This

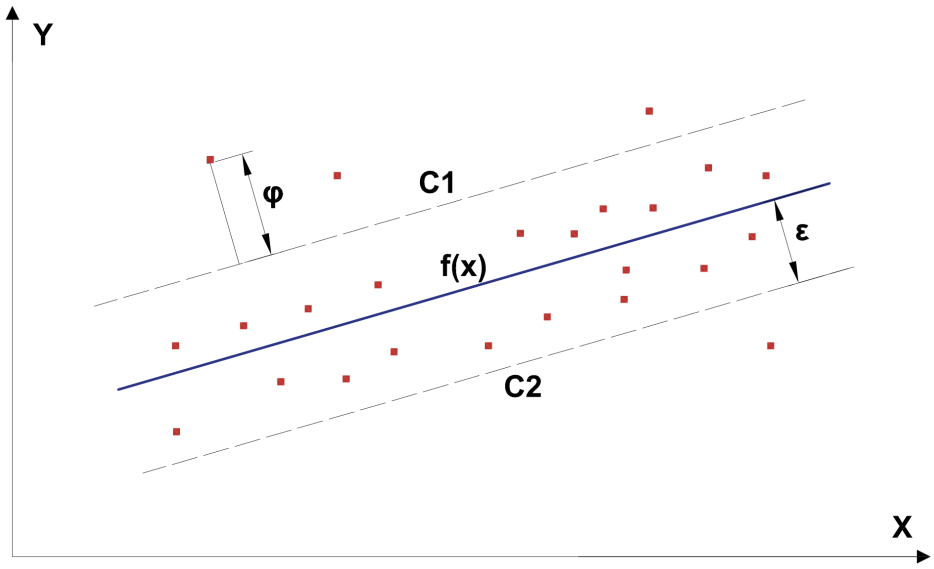


Figure 2.7: Representation of a linear Support Vector Regressor (SVR).

method finds the best solution between the different branches and leaves according to the characteristics of the input parameters [32]. Consequently, the Random Forest (RF) is a method that combines the power of decision trees to generate a classification based on various trees where uses sub-samples of the input data set, selected the best choice of each tree according to the established conditions [9].

The components of the RF are the number of trees, the number of leaves for each tree, and the tolerance to the final desired output value [27]. Inside the model, the algorithm which performs the classification is the Classification And Regression Tree (CART) [29]. The function of this algorithm for every tree is to split the training set into two parts to classify the values using a threshold. The next step is to search and split the subsets, looking for the purest ones, defined by the sizes and characteristics [30].

$$Y(x,x_k) = (\frac{m_{left}}{m}) * G_{left} + (\frac{m_{right}}{m}) * G_{right} \tag{2.4}$$

The equation 2.4 represents the cost function to perform the tree classification, where x represents the use of a single feature and x_k represents the established threshold. $G_{left/right}$ and $m_{left/right}$ represents the impurity and the number of instances of the subset, respectively [30].

The Random Forest Regression (RFR) method involves using the RF method with the difference in the output, where the predicted value represents the average of the obtained predictions by each tree [33]. As a result, in every tree, the CART algorithm is applied in the same way as before; the difference is that the algorithm does not split the training set to minimize the impurity, instead of split the training set to minimize the Mean Squared Error (MSE) [30].

$$Y(x,x_k) = (\frac{m_{left}}{m}) * MSE_{left} + (\frac{m_{right}}{m}) * MSE_{right} \tag{2.5}$$

Equation 2.5 represents the cost function to perform the tree regression, where x represents the use of a single feature and x_k represents the established threshold. $MSE_{left/right}$ and $m_{left/right}$ represents the mean squared error and the number of instances of the subset, respectively [30].

As previously mentioned, RFR combines the power of the tree classification method, adding the necessary number of trees and parameters like the solution needs. Figure 2.8 explains the configuration of a random forest where each tree predicts according to the set parameters. After that, specifically for the regression analysis, the average of these predicted values represents the output predicted value $Y(x)$.

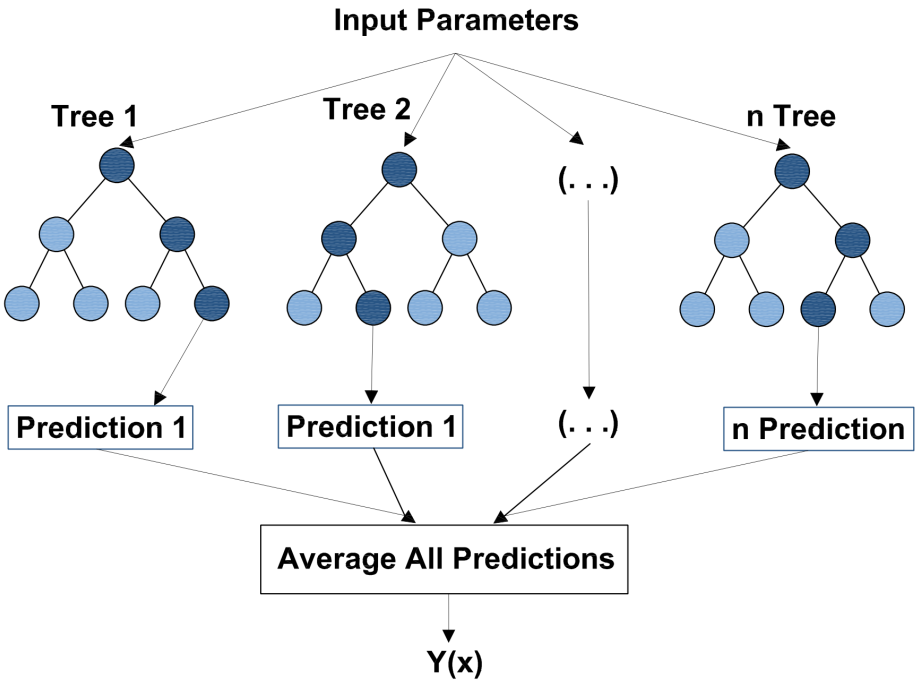


Figure 2.8: Representation of a Random Forest Regressor (RFR).

2.2.3. Multi-layer Perceptron Regressor

An Artificial Neural Network (ANN) is a computer model designed to simulate the human brain to process information and provide solutions related to predictions or classify the inputs of a data set. The artificial neurons, which are the principal components of an ANN, connect and construct an artificial neural system [25]. The perceptron is the most straightforward representation of an artificial neuron into the mentioned system. It has self-learning capabilities, enabling different configurations to generate better results based on the characteristics of the input data [30].

Multi-Layer Perceptron (MLP) is an ANN system with interconnected neurons in at least three different layers: the input layer, one or more than one hidden layer, and the output layer; this type of network is called Feedforward network [25] [34]. First, the MLP aims to predict or classify some data based on the parameters entered in the input layer. Then, each neuron into the hidden layers performs a simple non-linear transformation according to an activation function, producing an output value on the output of each layer [30] [35].

2.2.3.1. *The Backpropagation Algorithm*

The training of this neural network is through the Backpropagation (BP) algorithm. BP trains the network in two phases: one forward and one backward. First, the algorithm makes a prediction using the interconnected network based on the input values in the forward pass. After that, in the backward part, the algorithm changes the connection weights to reduce the model's error. This training process repeats until reducing the overall error according to an established threshold and loss parameter [30] [34].

2.2.3.2. *MLP for Regression Analysis*

For regression analysis, the MLP turns into a regressor adding a single neuron in the output, and this prediction will be the result of the regression [30]. Figure 2.9 illustrates a two hidden layer MLP regressor, where X_n represents the input parameters, $H1$ and $H2$ represent the hidden layers with n neurons each layer, and $Y(x)$ represents the output after the final layer of the algorithm.

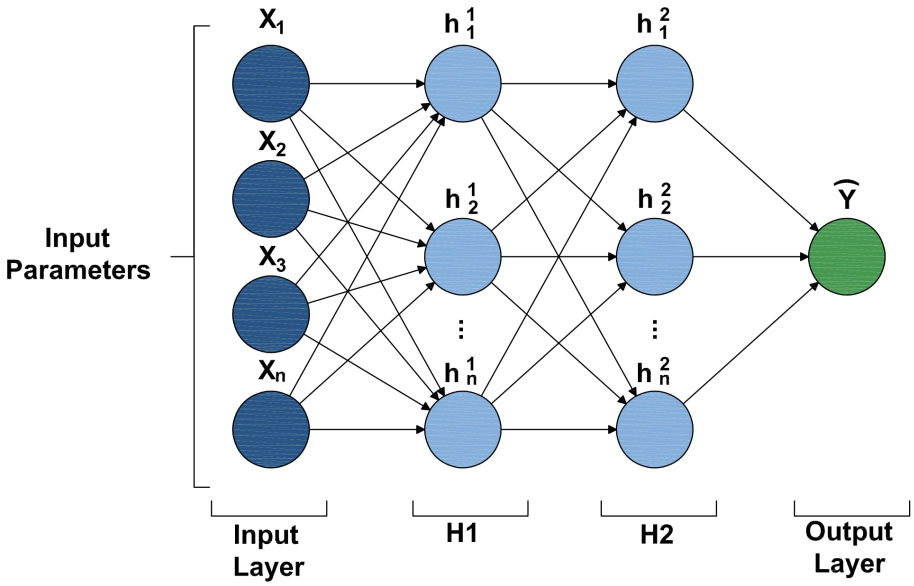
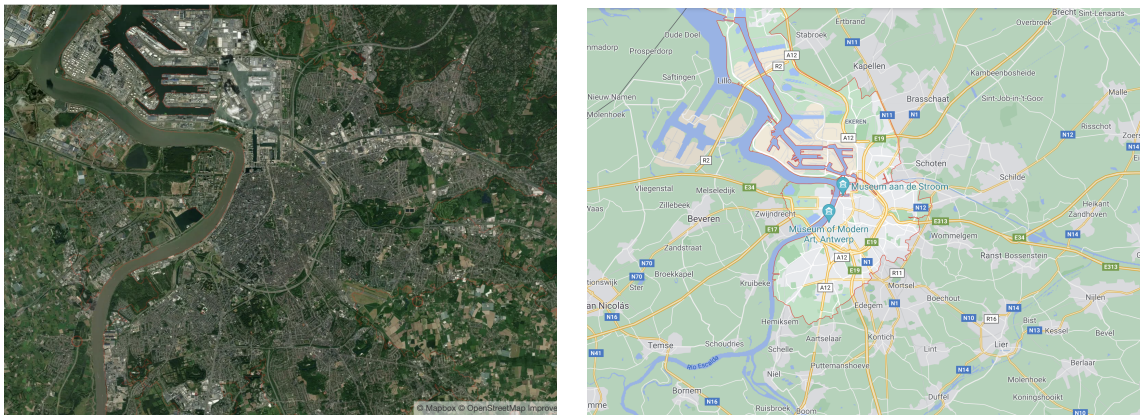


Figure 2.9: Representation of a 2 Hidden-Layer Perceptron Regressor (MLPR).

CHAPTER 3. METHODOLOGY

3.1. Antwerp City NB IoT Scenario

The city of Antwerp is the second biggest city in Belgium with a population of around one million residents [36]. It is located in the Flemish region with an extension of 204.5 km2 and a urban density value of 2,600 hab/km2. According to "The atlas of urban expansion" the percentage of urban distribution of the city is around 70%. increasing 1.1% each year[37]. Sitting Antwerp as a city with one of the highest growth levels compared with similar cities [36] [37]. Figure 3.1 shows the build up and street maps of Antwerp [38].



(a) Built Up Antwerp (b) Street Map Antwerp

Figure 3.1: Antwerp City [38].

Telecom operators have deployed a NB-IoT network in Belgium, concentrating the deployment in Antwerp with the aim of embrace mobile IoT as a catalyst for new companies and technology advances [20]. NB-IoT and LTE-M uses the same bases stations as LTE. This deployment represents the setting of around four thousand LTE base stations around Belgium. In Antwerp there are deployed more than four hundred base stations that are capable to support NB-IoT.

The database used for this study contains information of around three thousand NB-IoT messages and the location of the different base stations around urban and rural zones in the city of Antwerp [6]. The following subsections of this chapter will explain the process for data collection and the variable details. Also, how this data is debugged and prepared for the next processes in order to run ML algorithms to estimate the distance between the nodes and the Base Stations (BS).

3.2. Data Collection and Storage

The first database contains the sent messages from the nodes to each selected BS, becoming the updated version of the data in the work realized in [6]. Besides this messages database, a second database which contains a list of the characteristics and location of the IoT capable BS has been shared for the purpose of this investigation. These two

databases are described in detail in order to understand the next pre-processing steps. The exploratory analysis and all the data management is realized using algorithms implemented in Python 3 language and the set of tools provided by Pandas library [39].

3.2.1. NB-IoT Messages

The messages database has been collected around the city of Antwerp over the NB-IoT deployed network. Each message has been sent from a mobile vehicle using Ublox Sara N211 System, shown in Figure 3.2, over a Release 13 NB-IoT [40]. There is a total amount of 3093 samples which covers rural and urban areas of the city. The variables collected contains information from the nodes, which includes the ID, the GPS coordinates of the transmitter device and the received signal strength indicator (RSSI) [6]. Table 3.1 shows the first 10 rows of the database and figure 3.3 show the distribution of the samples around the city of Antwerp.

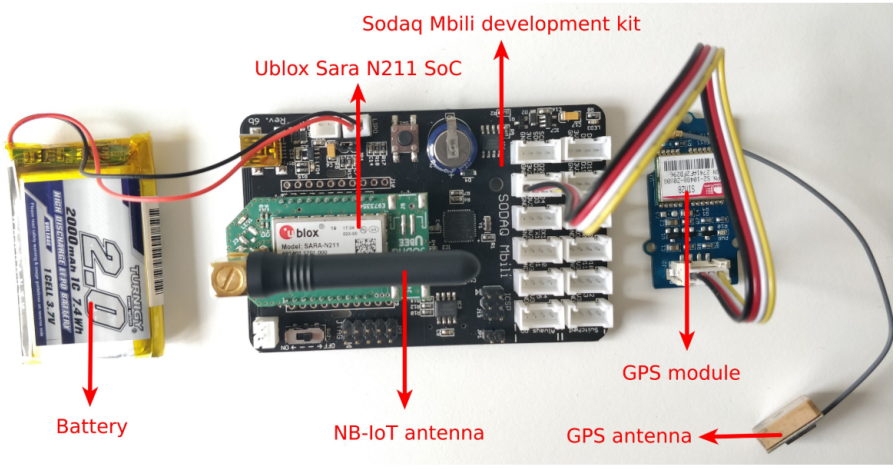


Figure 3.2: NB-Iot + GPS Sensor [15].

timestamp	id	rssi	latitude	longitude	altitude
2019-02-12T14:55:51.028	17270374	-63	51.2257	4.4011	-18.5
2019-02-12T14:55:54.082	17270374	-61	51.2257	4.4011	-18.5
2019-02-12T14:56:05.724	17270374	-61	51.2260	4.4010	-2.0
2019-02-12T14:56:28.672	17270374	-61	51.2258	4.4008	-2.0
2019-02-12T14:57:28.934	17270374	-65	51.2257	4.4009	-1.8999
2019-02-12T14:58:07.479	16909416	-61	51.2257	4.4008	-2.2000
2019-02-12T14:58:41.572	16909416	-63	51.2258	4.4005	-2.6000
2019-02-12T14:58:53.499	16909416	-63	51.2255	4.4000	6.2000
2019-02-12T14:59:16.805	16910440	-63	51.2252	4.3995	15.6000
2019-02-12T14:59:39.789	16910440	-65	51.2252	4.3994	17.8999

Table 3.1: NB-IoT messages data base.

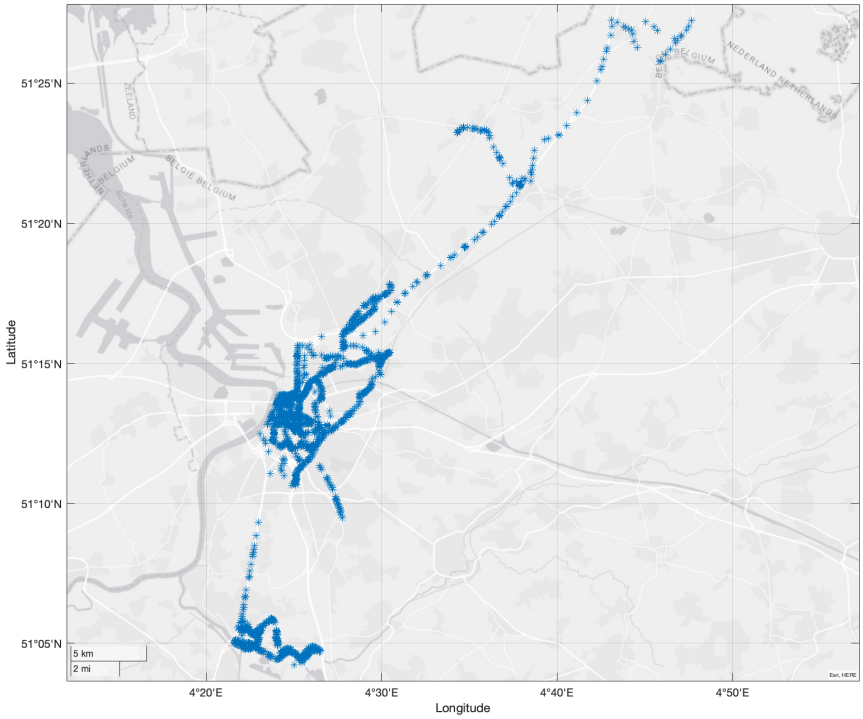


Figure 3.3: NB-IoT messages.

3.2.2. NB-IoT Base Stations

The NB-IoT Base Stations database belongs to the telecom operator in Belgium. There is a total amount of 10096 samples which covers all the base stations deployed in the country. The variables contains information about the base stations (BS), their ID, the GPS coordinates of the location and the azimuth angle for each antenna. Table 3.2 shows the first 10 rows of the database. Figure 3.4 Illustrates the location of all the BS in Belgium. In order to use the Antwerp city BS, a further process to merge this information is performed and explained in the next subsection.

id	latitude	longitude	altitude	azimuth
17226342	50.890784	4.307356	23.0	30
17226343	50.890784	4.307356	23.0	150
17226344	50.890784	4.307356	23.0	270
17647974	50.562789	4.303821	36.3	0
17647975	50.562789	4.303821	36.3	130
17647976	50.562789	4.303821	36.3	270
17330278	50.869985	4.547755	26.0	0
17330279	50.869985	4.547755	26.0	105
17330280	50.869985	4.547755	26.0	270

Table 3.2: NB-IoT BS data base.

description of variables

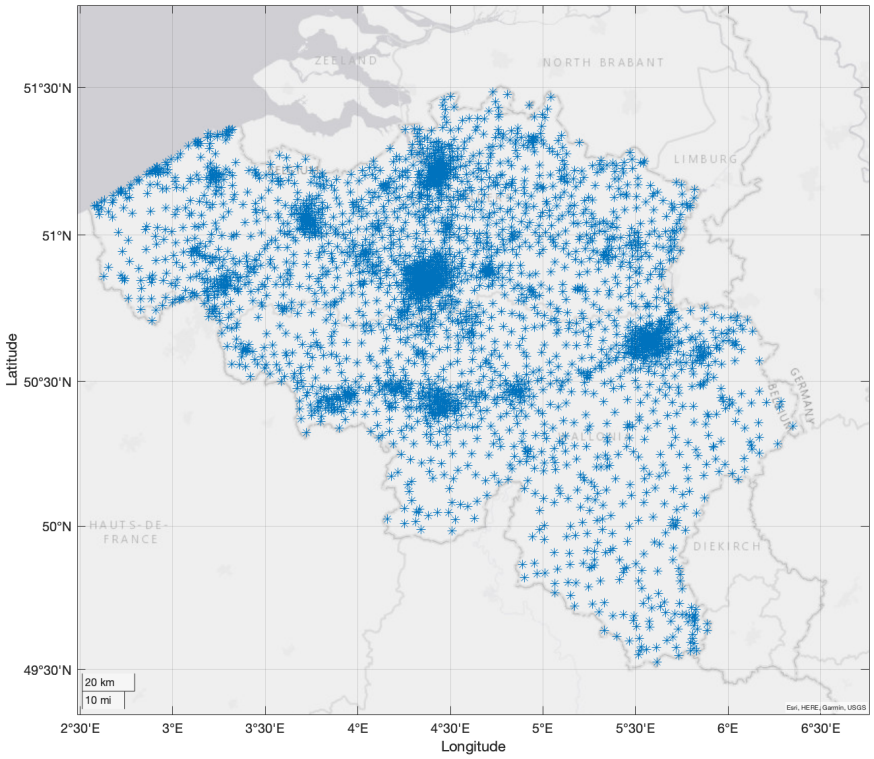


Figure 3.4: NB-IoT BS.

3.3. Data Processing

The NB-IoT messages and BS databases combined gather the enough information for the distance estimation process. Using the GPS coordinates of the NB-IoT nodes and the different base stations, it is possible to calculate the distance between these two points, and use the obtained values and the RSSI values for training the different ML regression algorithms and generate prediction models for distance estimation. The first process in order accomplish this calculus is unify the two databases, merging the necessary information for the Antwerp NB-IoT studied network. After that, there is needed to realize data preprocessing in order to obtain the maximum benefit adding new values, new data categories and debugging wrong and missing cells.

3.3.1. Data Merging and Debugging

The merging process for the two databases is crucial for the distance estimation process because it brings the information to calculate the distance between the nodes and BS. Also, merging the data is the best filter between the two databases for obtaining the Antwerp base stations involved in the data collection process. All the used base stations for sending the messages from the nodes are identified in the BS database, and this merge will combine the information of precisely the used BS for this process in the city of Antwerp.

First, the data merging algorithm uses the common variable ”ID” on the two databases, identifying the rows to be merged. After, it copies the location details (latitude, longitude, altitude) in newly added columns in the messages database. This process creates a unified database with enough characteristics to calculate the distance between the nodes and the antenna locations.

Finally, the result of merging the two databases is a 10 column database containing the information of the nodes and the located BS. The number of BS antennas where the nodes sent the messages is 428. Figure 3.5 shows the plot of the merged results of the 3093 nodes and their respective BS in the city of Antwerp, where the BS is represented in blue color and NB-IoT messages in cyan color. For further steps, this merged database will be called the ”NB-IoT database.”

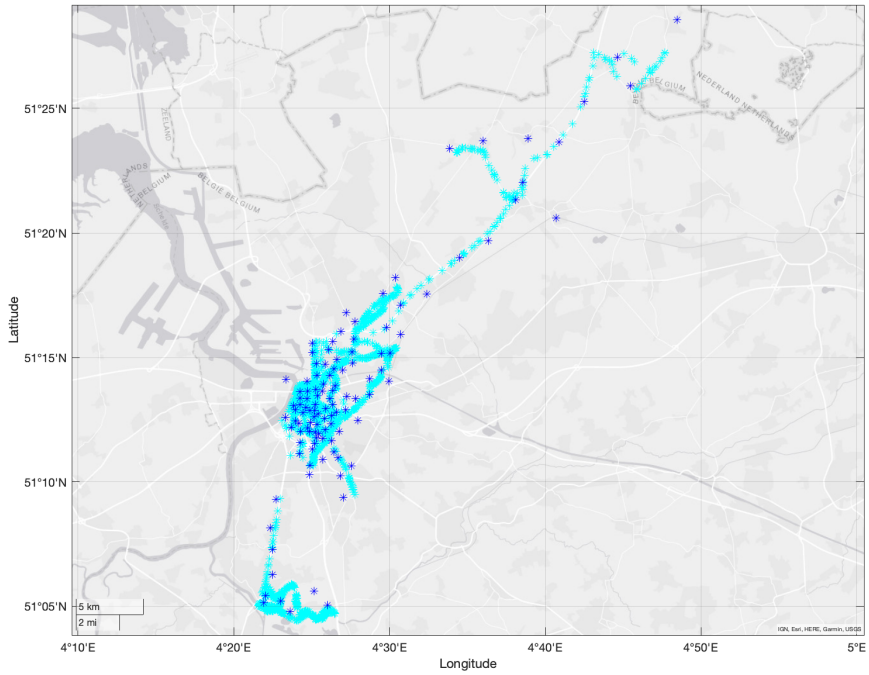


Figure 3.5: BS and NB-IoT messages in Antwerp City (NB-IoT database).

3.3.2. Data Pre-processing

Into Pre-processing group of actions, the first one is cleaning the database, and in order to do it, a profile of the database has been generated using the tool Profiling of Pandas Library [39]. Figure 3.6 shows the profile of the database, highlighting the essential characteristics. First, the missing and corrupted values on the NB-IoT database are identified and corrected. Second, the duplicate rows and non-numeric values are analyzed. Finally, all the variable types are revised to get the NB-IoT database ready to perform the mathematical calculus and the ML models.

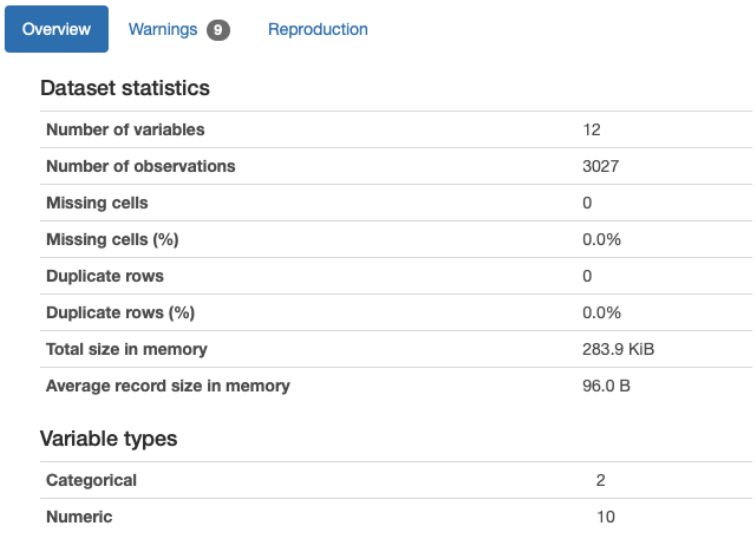


Figure 3.6: Profiling results of NB-IoT database [40].

3.3.2.1. Distance calculation

The distance between the BS's and the nodes becomes the first added variable to the database and the most important for the analysis because it will be the dependent variable of the ML regression models. The process use an algorithm for calculate the distance between 2 GPS points and it is illustrated in the equations 3.1, 3.2 and 3.3 representing the haversine formula, where φ is the latitude, λ is the longitude, R is the earth's radius and the angles are expressed in radians.

$$a = sin^2(\frac{\Delta\varphi}{2}) * cos(\varphi_1) * cos(\varphi_2) * sin^2(\frac{\Delta\lambda}{2}) \tag{3.1}$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{1-a}) \tag{3.2}$$

$$d(km) = R * c \tag{3.3}$$

The distance in kilometers is stored in a new column of the NB-IoT database. It is important to mention that in this process the rows 1611 to 1617 were deleted for being incorrect location values, obtaining finally a number of 3088 samples and its respective distance to the BS's stored in the NB-IoT database. Figure 3.7 shows the histogram of value distribution of the calculated distances between the nodes and the BS.

3.3.2.2. Urban and Rural classification

Perform a rural and urban classification and run the algorithms separated is important for the final distance estimation process. It is because the estimation error is higher for urban areas where the covered distances are higher than urban areas where on the contrary, there are short covered distances but a different building configuration, being a urban/rural classification so important on studying wireless networks [41].

The classification between the urban and rural zones of Antwerp has been carried out according to the urban and rural distribution shown in [36] [37]. Figure 3.8 illustrates this

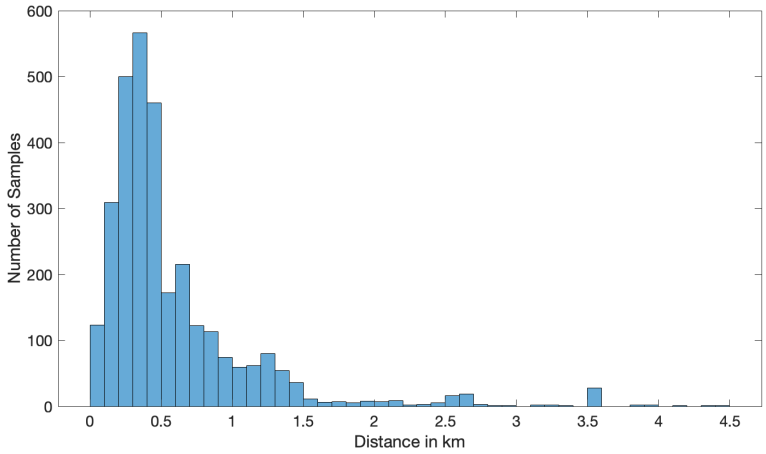


Figure 3.7: Histogram of the calculated distances between nodes and BS.

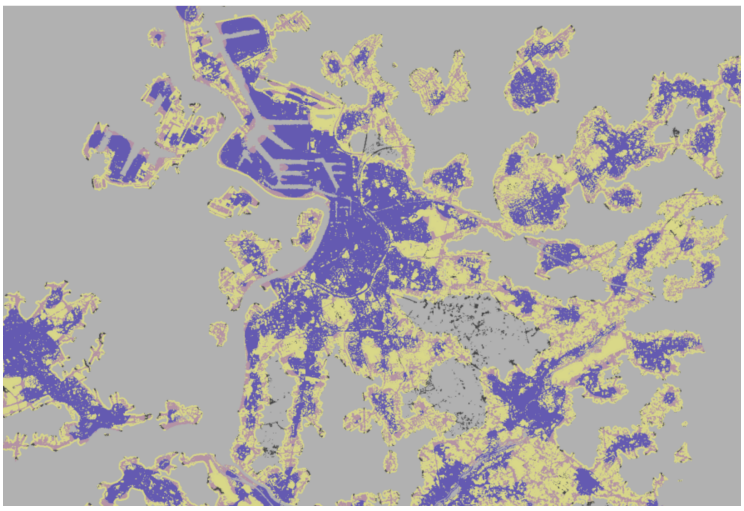
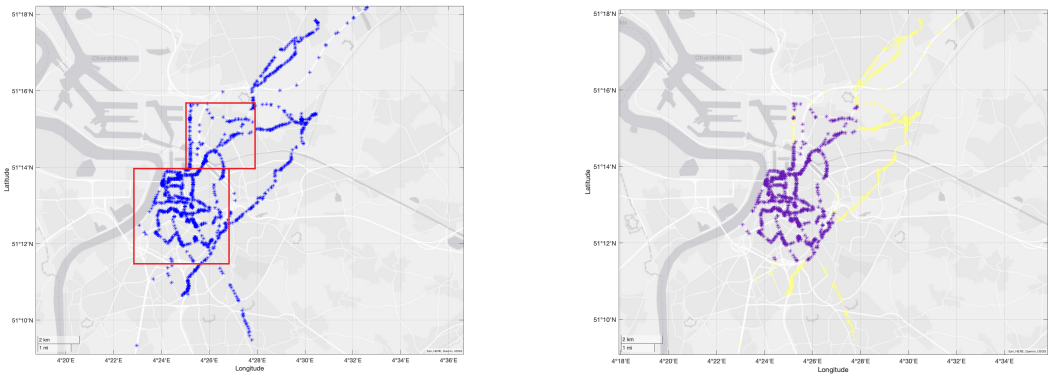


Figure 3.8: Urban and Rural distribution of Antwerp [37].

distribution where the yellow part represents the rural zones and the purple part represent the urban areas. The algorithm for classification creates a new column in the NB-IoT database where if the location of the node is between a set sector the value is true and if it not below to the sector the value is false. Figure 3.9 a) represents the two selected sectors for classify the zones in the database and 3.9 b) represent the final urban and rural distribution of the nodes in the NB-IoT database.

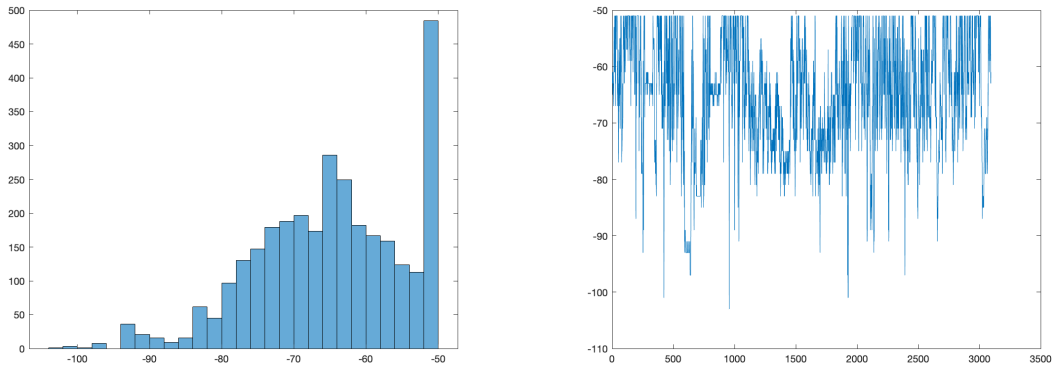
The received signal strength indicator RSSI values represents the power received in the BS in decibels (dB) and these values are originally negative. Analysing the RSSI values distribution in the NB-IoT dataset, the mean of the is 83 dB, and the biggest and smallest values are -103dB and -51dB respectively as illustrated in 3.10. The RSSI values are the only negative values in the database and these values are decreasing when the received power minimizes. This fact allows to change the sign of the values in order to obtain a better result in the further regression ML models, improving the next steps as the normalization and scaling of the input data [42] [6].

The final result of the data processing is a NB-IoT database that contains 14 rows, which



(a) Urban sectors in NB-IoT database (b) Urban/Rural distribution in NB-IoT database

Figure 3.9: Urban/Rural classification process for NB-IoT database.



(a) Histogram of RSSI values Antwerp (b) Plot RSSI values Antwerp

Figure 3.10: Analysis of RSSI values in NB-IoT database

are composed by the Nodes positions, the BS positions, the RSSI values, the distance between the nodes and the BS's and the Urban and rural classification for all the samples in the database. The execution of this process allows the correct implementation of the ML regression algorithms for distance estimation and further location estimation of the IoT nodes. The table 3.3 shows the ten first rows of the NB-IoT database where 2 columns were drooped (distance in meters and rural variable) for the correct illustration of it.

timestamp	id	rssi	latitude	longitude	altitude	Lat_BS	Lon_BS	alt_BS	distance_BS	Urb	Az
2019-02-12T14:55:51.028000	17270374	63	51.2257	4.4011	-18.5	51.2228	4.404134	20.0	0.376031	1	55
2019-02-12T14:55:54.082000	17270374	61	51.2257	4.4011	-18.5	51.2228	4.404134	20.0	0.376031	1	55
2019-02-12T14:56:05.724000	17270374	61	51.2260	4.4010	-2.0	51.2228	4.404134	20.0	0.415551	1	55
2019-02-12T14:56:28.672000	17270374	61	51.2258	4.4008	-2.0	51.2228	4.404134	20.0	0.405252	1	55
2019-02-12T14:57:28.934000	17270374	65	51.2257	4.4009	-1.8999	51.2228	4.404134	20.0	0.394295	1	55
2019-02-12T14:58:07.479000	16909416	61	51.2257	4.4008	-2.2000	51.227258	4.403851	30.2	0.268738	1	240
2019-02-12T14:58:41.572000	16909416	63	51.2258	4.4005	-2.6000	51.227258	4.403851	30.2	0.277654	1	240
2019-02-12T14:58:53.499000	16909416	63	51.2255	4.4000	6.2000	51.227258	4.403851	30.2	0.325773	1	240
2019-02-12T15:02:33.861000	16896870	65	51.2215	4.3970	44.2999	51.21765	4.397330	23.5	0.435371	1	0
2019-02-12T15:02:45.444000	16896870	63	51.2206	4.3963	30.5	51.21765	4.397330	23.5	0.337442	1	0
2019-02-12T15:03:12.046000	16896870	57	51.2198	4.3958	14.8000	51.21765	4.397330	23.5	0.260933	1	0

Table 3.3: 10 first rows of the processed NB-IoT data base.

3.4. Algorithm Implementation

The algorithm implementation is composed by 3 parts. The first one is the Feature selection, where the variables are selected to be part of the input for the regression models and where the distance is set as the output variable to be predicted. The second part explain in detail the executed algorithms for the three ML models. Finally, where the ML models predicts the estimated values for distance between the nodes and BS, the location predicted points are calculated. Based on these results the localization estimation errors and model measures are generated for evaluation and discussion.

3.4.1. Feature Selection

The Feature selection process establishes the designed variables for the ML models. The selection of variables is supported by statistical methods as correlation matrices, descriptive analysis of the data, related work, and others. The algorithm for feature selection process for the NB-IoT database is carried out according to the flow diagram illustrated in figure 3.11, where after read the NB-IoT processed database starts a sequence of steps in order to create the input and output datasets for the train and test of the ML models.

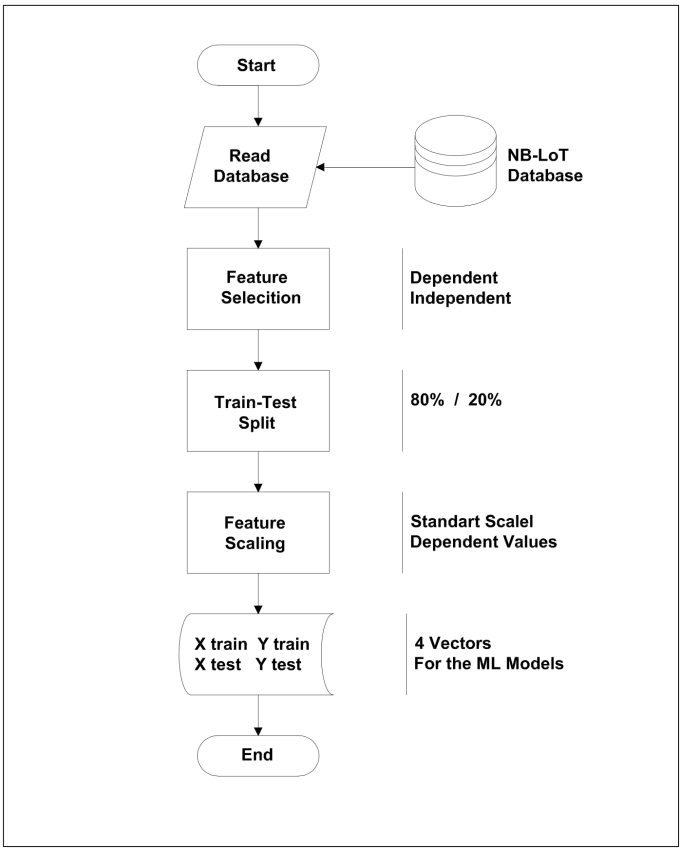


Figure 3.11: Flow diagram for feature selection process.

The dependent variable is defined since the beginning of the project and it is the distance between the nodes and BS. This dependent variable is calculated and it will be the output parameter for the ML regression models. The selection of the independent variables is

based on the information provided by the correlation matrix illustrated in 3.12, where the correlation is analysed in order to find a positive correlation (blue gamma of colors) with the calculated distance. The results of this analysis establishes the RSSI values, the BS location parameters (longitude, latitude and altitude) and the azimuth of the BS antenna as the positive correlated parameters for the distance prediction. Being these group of variables the input parameters of the ML regression models.

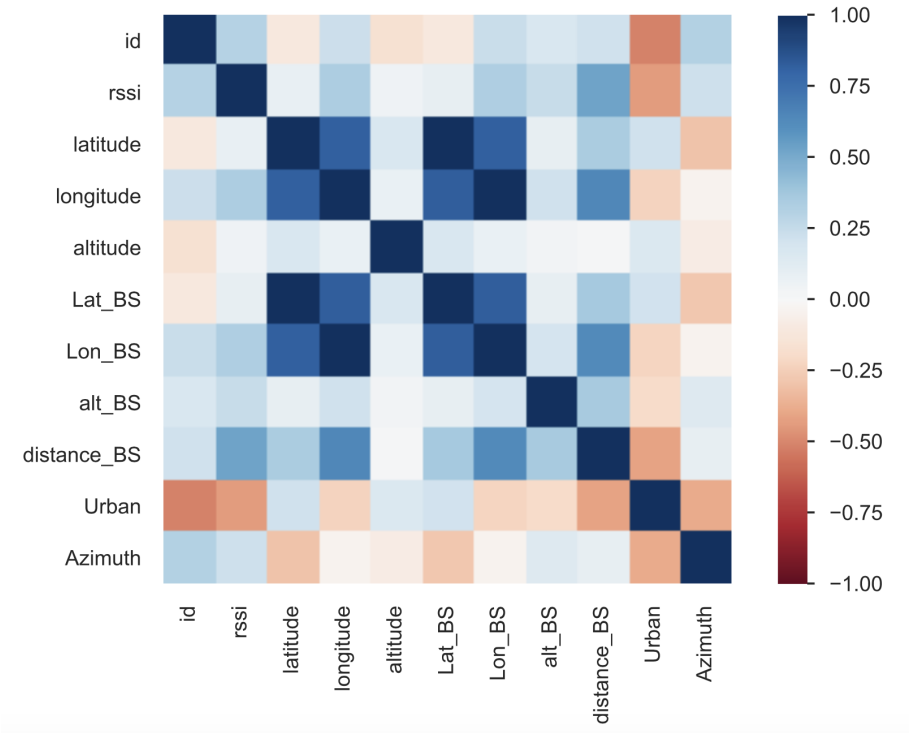


Figure 3.12: NB-IoT database variable correlations.

The next process is to split the NB-IoT database for test and train the models. This process is carried out using the tools provided by the Sci-kit Learn library for Python[43]. The chosen proportion for split the database is 80% for training the model and 20% for testing the model. The results of the train-test-split process are the vectors $X-train, X-Test$ for the input parameters and $Y-train$, $Y-test$ for the output parameter. The final process of the feature selection is to standardize the values in order to obtain the correct output values from the implemented models.

$$z = (\frac{x-u}{s}) \tag{3.4}$$

The equation 3.4 represents the mathematical process for standardizing x features where u is the mean of the training samples, and s is the standard deviation of the training samples.

Once the feature process ends and the input and output parameters are defined, the Support Vector Machine, the Random Forest and the Multi-layer Perceptron are implemented, according the flow diagram illustrated in figure 3.13. The next subsections emphasize the execution details of each model, where the principal tools used for their implementation are Sci-kit Learn, Keras and Tensor-Flow libraries over Python 3 programming language.

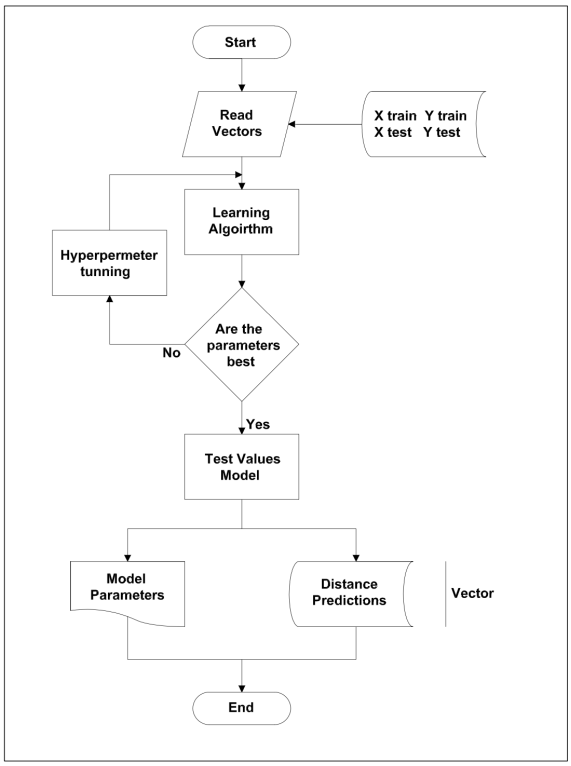


Figure 3.13: Flow diagram of the ML models execution.

3.4.2. Support Vector Regression

3.4.2.1. Algorithm configuration

The Support Vector Machine Regression (SVR) algorithm is set to read the input vectors $X-train$, $Y-train$ for the training process of the algorithm. The first execution of the model is performed using the default set parameters [43], which are polynomial kernel, polynomial degree equal to one, and epsilon value equal to 0.1. The code below shows the execution results of the SVR for training the model. In this algorithm the score measure is the coefficient of determination R squared and the principal measured error values are the Mean Absolute Error (MAE), The Mean Squared Error (MSE) and The Median Error.

```
R^2: 0.5433311633159026
{'C': 1.0, 'cache_size': 200, 'coef0': 0.0, 'degree': 1, 'epsilon':
0.1, 'gamma': 'scale', 'kernel': 'poly', 'max_iter': -1, 'shrinking':
True, 'tol': 0.001, 'verbose': False}
Mean Absolute Error (MAE): 0.23421782618502743
Mean Squared Error (MSE): 0.12167793734695186
Max Error: 1.3430591609600884
Median Absolute Error: 0.14054557441723728
```

3.4.2.2. Hyper-parameter tuning

The Hyper-parameters tuning helps to optimize the algorithm choosing the best para meters for the application. In the case of this regression experiment, the column ”Values” from

the table 3.4 shows the hyper-parameters and its values for searching the best configuration for the SVR learning algorithm, this process is achieved using the tool GridsearchCV from Sci-kit Learn library [43].

Hyperparameter	Values	Selected Values
Regularization C	[0.1, 1, 100]	[1]
Gamma	[scale, auto]	[scale]
Epsilon	[1, 0.1, 0.01]	[0.1]
Kernel	['linear', 'poly', 'sigmoid', 'rbf']	['rbf']
Degree	[1, 2, 3, 4, 5]	[1] for rbf kernel

Table 3.4: Hyper-parameters for RFR model.

The tuning process executes the algorithm with every variation of the selected Hyper-parameters in order to reduce the error of the model, which in this model is the MAE. In every execution of the code, the tuning tool also uses the K-Fold evaluation process where the NB-IoT training database ($X-train$, $Y-train$) is divided into test and train data sets and executed five times with different combinations of 80/20 percent split proportion. The figure 3.14 illustrates the K-Fold process of train test split for five different folds [43].

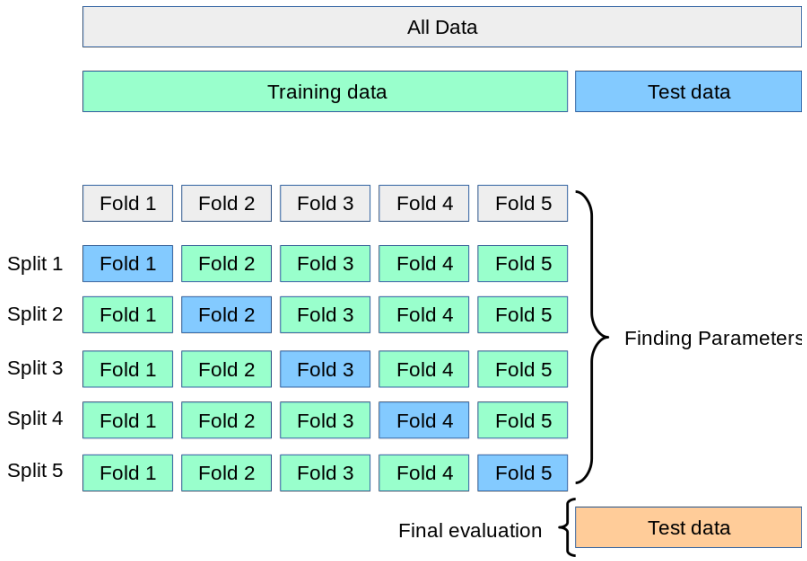


Figure 3.14: K Fold data splitting process [43].

The final selected parameters are showed in the table 3.4, and the final results after select the best parameters for the model applied on the whole NB-IoT database (Urban and Rural) are added below.

```
R^2: 0.7456464541444283
{'C': 1.0, 'cache_size': 200, 'coef0': 0.0, 'degree': 1, 'epsilon':
0.1, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'shrinking':
True, 'tol': 0.001, 'verbose': False}
Mean Absolute Error (MAE): 0.1642226082414969
Mean Squared Error (MSE): 0.06777168120626229
Max Error: 1.5741667411838147
Median Absolute Error: 0.09996635255759578
```

3.4.2.3. Train test and validation

The process of validating the model results in order to avoid overfitting in the ML model is carried out by a K-Fold cross validation process. This process is the same as illustrated in the figure 3.14 with 10 set folds for the validation process [43]. The table 3.5 shows the individual scores for every execution, where the mean score of all the executions is 0.67 accuracy with a standard deviation of 0.05.

K-Fold Cross Validation Algorithm Executions									
1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**
0.508235	0.478158	0.585044	0.531276	0.52924	0.51257	0.51262	0.518432	0.421668	0.54999

Table 3.5: K-Fold Cross validation scores for SVR in urban and rural scenarios.

3.4.3. Random Forest Regression

3.4.3.1. Algorithm configuration

The Random Forest Regression (RFR) algorithm is set to read the input vectors $X-train$, $Y-train$ for the training process of the algorithm. The first execution of the model is performed using the default set parameters [43], which are fifty estimators, and 1 leaf per sample. The code below shows the execution results of the RFR for training the model. In this algorithm the score measure is the coefficient of determination R squared and the principal measured error values are the Mean Absolute Error (MAE), The Mean Squared Error (MSE) and The Median Error.

```
0.8264810464300284
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'mse', 'max_depth'
: None, 'max_features': None, 'max_leaf_nodes': None, 'max_samples':
None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 10, 'min_weight_fraction
_leaf': 0.0, 'n_estimators': 1000, 'n_jobs': None, 'oob_score':
False, 'random_state': 3, 'verbose': 0, 'warm_start': False}
Mean Absolute Error (MAE): 0.14526395064155118
Mean Squared Error (MSE): 0.05099224106490743
Max Error: 1.176582914033302
Median Absolute Error: 0.08674038813552519
```

3.4.3.2. Hyper-parameter tuning

The Hyper-parameters tuning helps to optimize the algorithm choosing the best para meters for the application. In the case of this regression experiment, the column ”Values” from the table 3.6 shows the hyper-parameters and its values for searching the best configuration for the SVR learning algorithm, this process is achieved using the tool GridsearchCV from Sci-kit Learn library [43].

The tuning process executes the algorithm with every variation of the selected Hyper-parameters in order to reduce the error of the model, which in this model is the MAE. In every execution of the code, the tuning tool also uses the K-Fold evaluation process where

Hyperparameter	Values	Selected Values
Number of estimators	[10, 50, 80, 100, 500]	[100]
Max Features	['auto','sqrt']	["sqrt']
Max depth	[2, 4, 8 ,10, None]	[10]
Min samples split	[2, 8, 16]	[8]
Min samples leaf	[1, 2, 4]	[1]
Bootstrap	['True','False']	['True']

Table 3.6: Hyper-parameters for RFR model.

the NB-IoT training database ($X-train, Y-train$) is divided into test and train data sets and executed five times with different combinations of 80/20 percent split proportion. The figure 3.14 illustrates the K-Fold process of train test split for five different folds [43].

The final selected parameters are showed in the table 3.6, and the final results after select the best parameters for the model are added below.

```
R^2: 0.8255853389465615
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'mse', 'max_depth'
: None, 'max_features': None, 'max_leaf_nodes': None, 'max_samples':
None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf':1, 'min_samples_split': 10, 'min_weight_fraction_
leaf': 0.0, 'n_estimators':100, 'n_jobs': None, 'oob_score': False,
'random_state': 3, 'verbose': 0,'warm_start': False}
Mean Absolute Error (MAE): 0.14473533083977178
Mean Squared Error (MSE): 0.0500308897119315
Max Error: 1.1460325477071953
Median Absolute Error: 0.08670440257197358
```

3.4.3.3. Train test and validation

The process of validating the model results in order to avoid overfitting in the ML model is carried out by a K-Fold cross validation process. This process is the same as illustrated in the figure 3.14 with 10 set folds for the validation process. The table 3.7 shows the individual scores for every execution, where the mean score of all the executions is 0.73 accuracy with a standard deviation of 0.05.

K-Fold Cross Validation Algorithm Executions									
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
0.684097	0.686874	0.804652	0.782578	0.758779	0.785445	0.690506	0.743537	0.668995	0.746649

Table 3.7: K-Fold Cross validation scores for RFR in urban and rural scenarios.

3.4.4. Multi-layer Perceptron Regression

3.4.4.1. Algorithm configuration

The Multi-layer perceptron Regressor (MLPR) algorithm is set to read the input vectors $X-train, Y-train$ for training the algorithm. This neural network is configured with 2 hidden

layers, and the first training of the model is performed using the default set parameters which are 4 neurons in the two hidden layers, relu activation function, Adam optimizer and 0.1 dropout rate. The code below shows the execution results of 100 epochs for training the model. In this algorithm the score measure Mean Absolute Error (MAE), The Mean Squared Error (MSE) and The Median Error. The MLPR model has been implemented using the tools provided by Keras and Tensor-flow for Python 3 [44].

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 5
=================================================================
Total params: 45
Trainable params: 45
Non-trainable params: 0
_________________________________________________________________
None
20/20 [==============================] - 0s 934us/step -
loss: 0.1809 - mae: 0.1809 - mse: 0.0806...
[0.1809142529964447, 0.1809142529964447, 0.08056644350290298]
```

3.4.4.2. Hyper-parameter tuning

The Hyper-parameters tuning helps to optimize the algorithm choosing the best para meters for the application. In the case of this regression experiment, the column ”Values” from the table 3.8 shows the hyper-parameters and its values for searching the best configuration for the SVR learning algorithm, this process is achieved using the tool GridsearchCV from Sci-kit Learn library [43].

Hyperparameter	Values	Selected Values
Batch size	[1, 8, 16]	[1]
Number of neurons in hidden layers	[4, 8, 16, 32, 64, 128]	1st [16] 2nd[8]
Activation function	[relu, tanh, sigmoid, linear]	[relu]
Optimizer	[’SGD’, ’RMSprop’, ’Adam’, ’Adamax’ , ’Nadam’]	[’Adam’]
Dropout rate	[0.1, 0.2, 0.3, 0.4, 0.5]	[None]

Table 3.8: Hyper-parameters for MLPR model.

The tuning process executes the algorithm with every variation of the selected Hyper-parameters in order to reduce the error of the model, which in this model is the MAE. In every execution of the code, the tuning tool also uses the K-Fold evaluation process where the NB-IoT training database ($X-train, Y-train$) is divided into test and train data sets and executed five times with different combinations of 80/20 percent split proportion. The figure 3.14 illustrates the K-Fold process of train test split for five different folds [43].

The final selected parameters are showed in the table 3.8, and the final results after select the best parameters for the model are added below.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                80
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 225
Trainable params: 225
Non-trainable params: 0
_________________________________________________________________
None
20/20 [==============================] - 0s 938us/step -
loss: 0.1792 - mae: 0.1792 - mse: 0.0799...
[0.1791994571685791, 0.1791994571685791, 0.07992436736822128]
```

3.4.4.3. Train test and validation

The process of validating the model results in order to avoid overfitting in the ML model is carried out by a K-Fold cross validation process. This process is the same as illustrated in the figure 3.14 with 10 set folds for the validation process. The table 3.9 shows the individual scores for every execution, where the mean score of all the executions is 0.175 accuracy with a standard deviation of 0.018.

K-Fold Cross Validation Algorithm Executions									
1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**
0.168134	0.165864	0.164043	0.164950	0.180914	0.179199	0.224806	0.168806	0.167124	0.167996

Table 3.9: K-Fold Cross validation scores for MLPR in urban and rural scenarios.

3.4.5. Estimation of Node Position

The final process after the algorithm configuration is to estimate the position of the nodes. This process is carried out using the same formulas for calculating the distance between the BS and the nodes. The haversine formulas illustrated in the equations 3.1, 3.2 and 3.3 allows to calculate the latitude and longitude coordinates of the predicted points knowing the distance estimation and the BS latitude, longitude and Azimuth angle. The figure 3.15 illustrated the flow diagram of the point estimation algorithm part. Table 3.10 shows the ten first rows of the location predicted points in the NB-IoT database.

The location estimation error is defined as the distance between the predicted points and the original nodes location. This error is calculated using haversine equations 3.1, 3.2

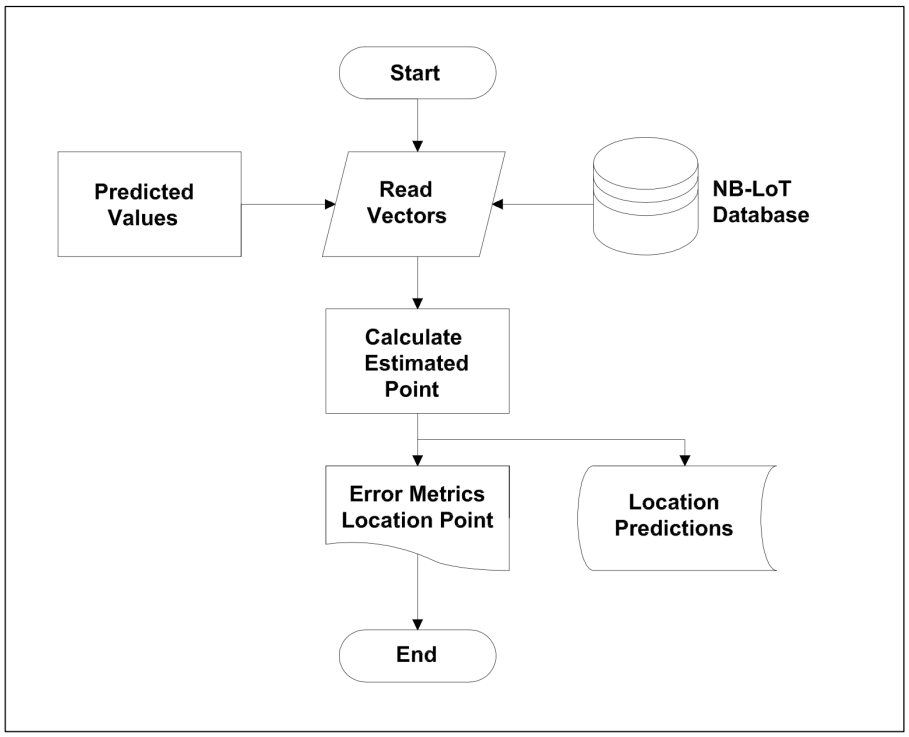


Figure 3.15: Flow diagram of the predicted location algorithm.

Node_Lat	Node_Lon	New_Lat	New_lon	Distance_Pred_Error
51.07876205	4.434211730	51.07941240	4.420119372	0.9871162596
51.22121429	4.402654647	51.22138082	4.404259	0.1132560550
51.24987411	4.468908309	51.25393074	4.452164633	1.249562559
51.25625228	4.505593776	51.2581015	4.503569055	0.2492701227
51.08998870	4.383974552	51.09004548	4.380579038	0.2372316992
51.20126342	4.405999660	51.20014509	4.403441840	0.2173105004
51.23548889	4.429390430	51.23492671	4.429081	0.0661190765
51.21386337	4.440446376	51.21108554	4.435871286	0.4438097975
51.18851089	4.42354679	51.19010374	4.413793657	0.7024071428
51.21251678	4.414055347	51.21747880	4.41601198	0.5683349651

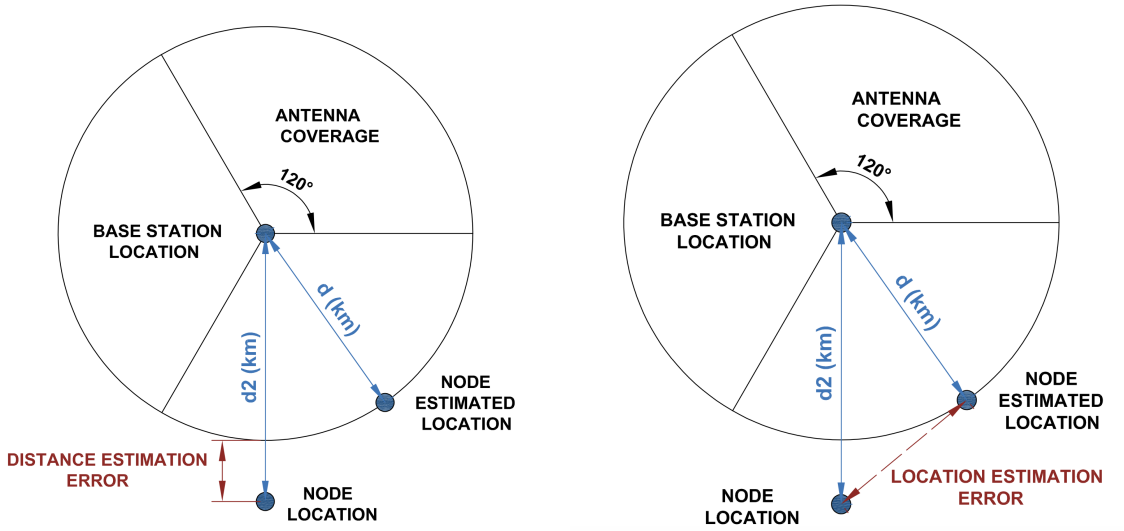
Table 3.10: First 10 Rows of the location predicted points database.

and 3.3. Finally the Mean Absolute Error (MAE) and Median Absolute Error (MED) of this distance are calculated and used to measure the accuracy of the implemented algorithms.

CHAPTER 4. RESULTS AND DISCUSSION

The correct algorithm implementation and validation process allows to obtain reliable results for node locations estimation. In this chapter, the results are divided in two main groups characterized by the environment, namely urban and rural areas. First, the Urban area representing 44% of the samples (1355) in the NB-IoT database and second, the rural area with the remaining 56% (1733) of the database.

The results analysis is based on two performance metrics. The first one is the distance estimation error which is calculated as the difference between the estimated distance between the nodes and the BS, illustrated in the figure 4.1 a). Second, the location estimation error which is calculated as the distance between the nodes original location coordinates (latitude and longitude) and the predicted coordinates, it is illustrated in the figure 4.1 b).



(a) Distance estimation error

(b) Node location error

Figure 4.1: Error measures for distance and location predictions.

The estimation results of the three implemented algorithms are compared to each other in order to discuss their performance in the two different scenarios. The mean and median values of the 2 performance metrics are calculated and summarized on measure tables, and different statistic graphs as error plots, histograms, cumulative functions and GPS plots of the predicted points are presented and discussed in the sections below [45].

4.1. Results for The Urban Area

Table 4.1 shows the results for the urban area, where the distance estimation error is around one hundred meters, with median values of around seventy meters. For the location estimation error the mean value is around two hundred eighty meters with a median value of around two hundred twenty. This estimation errors confirms that the proposed solution outperforms the proximity and ranging methods explained before and are applicable for solutions where there is no need of an accurate position estimation. The three

ML Model	Distance Estimation		Location Estimation	
	Mean Error	Median Error	Mean Error	Median Error
SVR	94.986 m	73.388 m	280.691 m	213.230 m
RFR	95.789 m	62.132 m	280.353 m	224.957 m
MLP R	101.843 m	71.556 m	284.306 m	228.436 m

Table 4.1: Estimation error results for urban area.

implemented algorithms show a uniform behavior which adds reliability to the obtained results.

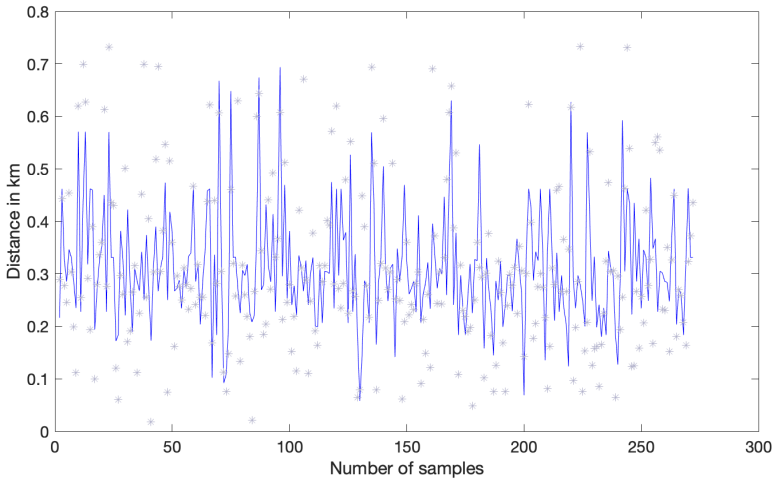


Figure 4.2: SVR distance between predicted and test values.

Figure 4.2 shows the plot of the predicted values and the test values of the distance between the BS and the nodes, using the Support Vector Regression ML model. The analysis of this graph shows a good algorithm performance predicting distances between one hundred 5 hundred meters. This accuracy could be appreciated in the graph where the blue plot represents the behaviour of the predicted values and the gray points represent the shape of the test values. The predicted values achieves most of the test values, but there is important to denote that the regressor can not predict values under the fifty meters boundary. Also, the distance estimation score is the best of the three models with 94.986 meters of mean distance estimation error.

Figure 4.3 shows the plot of the predicted values and the test values of the distance between the BS and the nodes, using the Random Forest Regression ML model. The analysis of this graph shows a good algorithm performance predicting distances between one hundred and six hundred meters. The predicted values line achieves most of the test ones, but there is important to denote that the RFR can not predict values under the one hundred meters boundary.

Figure 4.4 shows the plot of the predicted values and the test values of the distance between the BS and the nodes, using the Multi-layer Perceptron Regression ML model. The analysis of this graph shows a good algorithm performance predicting distances between one hundred and seven hundred meters. This accuracy could be appreciated in the graph where some test values of more than five hundred meters are achieved by the prediction

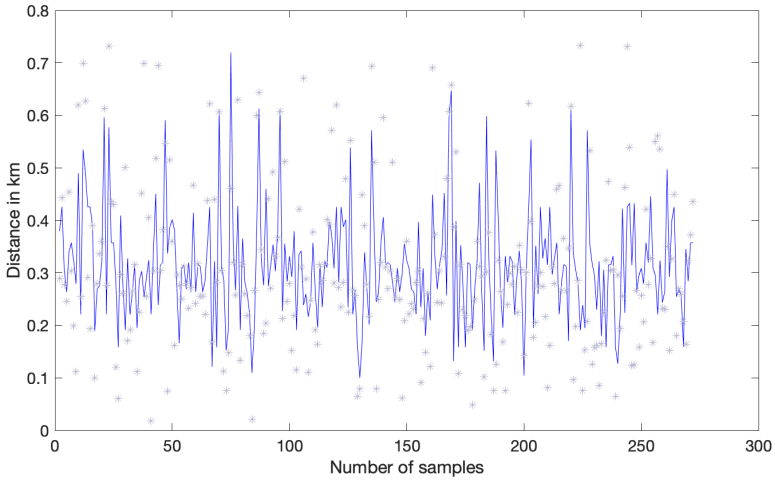


Figure 4.3: RFR distance between predicted and test values.

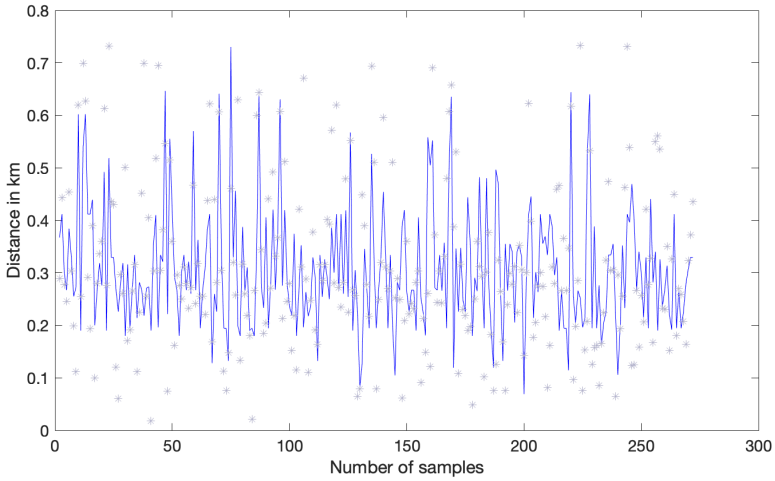


Figure 4.4: MLPR distance between predicted and test values.

values line. A well as the RFR, the regressor can not predict values under the one hundred meters boundary.

The figure 4.14 shows the cumulative distribution of the estimated location errors for the test dataset, where, the values are concentrated between one hundred and three hundred meters for the three ML models. Near of the six hundred meters, the MLPR concentrates more values than the other algorithms, this fact could be explained as the behaviour of the model where the memory of the neurons allows to classify the results before the regression with this groups segmentation. The comparison between the graphs 4.3, 4.4, and 4.4 also shows how the MLPR achieves more values upper the six hundred meters, compared with the two other models.

The Box plot on the figure 4.15 compares the location estimation error between the ML models in the urban area. Where, the highlights of this comparison are first the median value of this error around two hundred meters. this fact adds reliability to the solution because shows that at least the half of the values are estimated with an accuracy of two

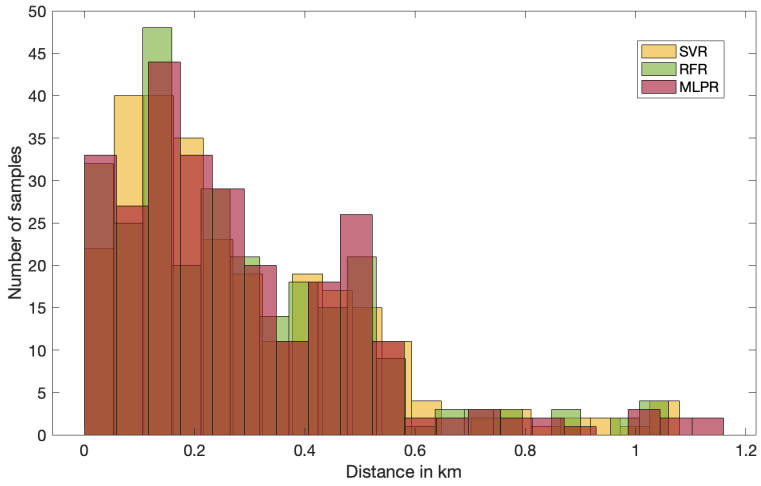


Figure 4.5: Histogram of Error value distribution for the urban area.

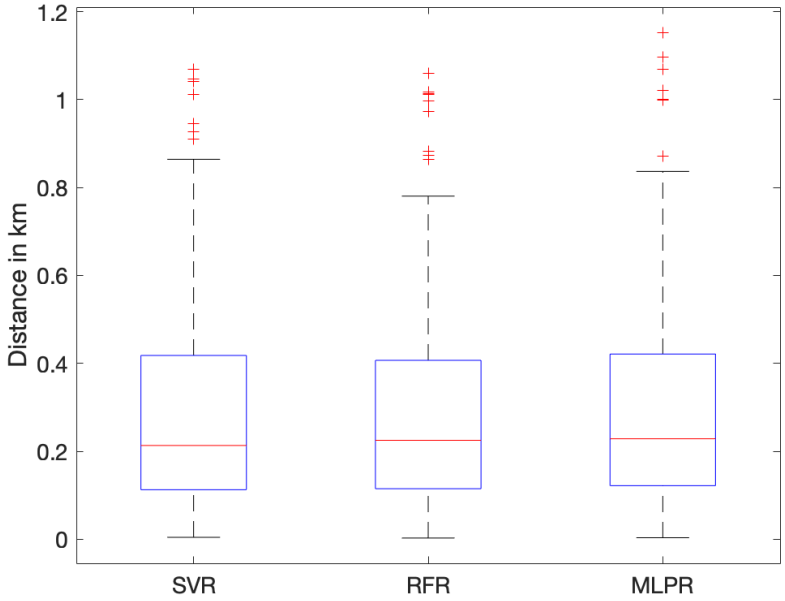


Figure 4.6: Location estimation error for the urban area.

hundred meters or less. Second, most of the measures are less than four hundred meters, with some values up to eight hundred meters and out-layer values from 0.82 to 1.2 km which adds some noise to the whole accuracy of the models. Finally, the algorithm with the best performance in this comparison is the RFR with the lower values in the Q4 group and lower values for the out-layers.

The cumulative distribution function of the location estimated errors, denotes a similar behaviour for the three ML models. Where supporting the box plot analysis the 90% of the values are situated below the six hundred meters. This fact represents a good performance of the model achieving the ninety percent of the values in the half of the 1.2 km that are maximum distance error for this urban analysis.

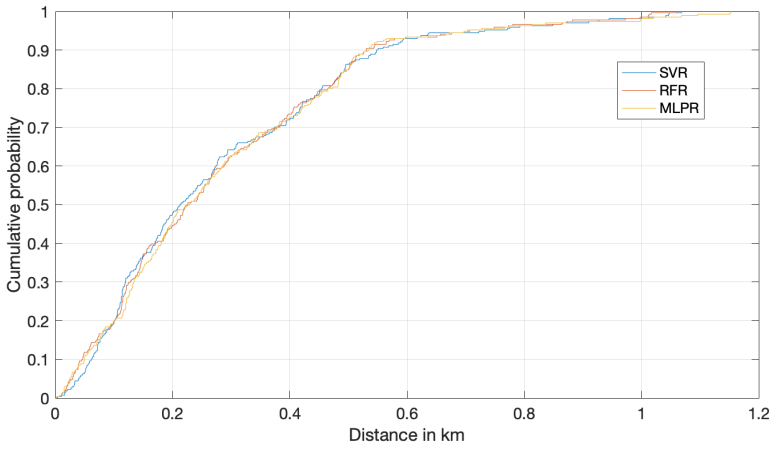


Figure 4.7: Cumulative density function of the location error for the urban area.

Finally the group of maps illustrated in the figures 4.8, 4.9 and 4.10 shows the predicted points and their comparison, first with the original nodes location (literal a) and second with the BS around the predicted nodes location. The three algorithms shows a good performance in terms of the location for the urban area. this performance allows the use of this solution for logistics tracking, where it is not necessary the exact location. The approach of using the azimuth angle could be perceived in the BS and predicted nodes graphs, because the nodes around the BS are distributed in the shape of 120 degrees of the three antennas on each BS.

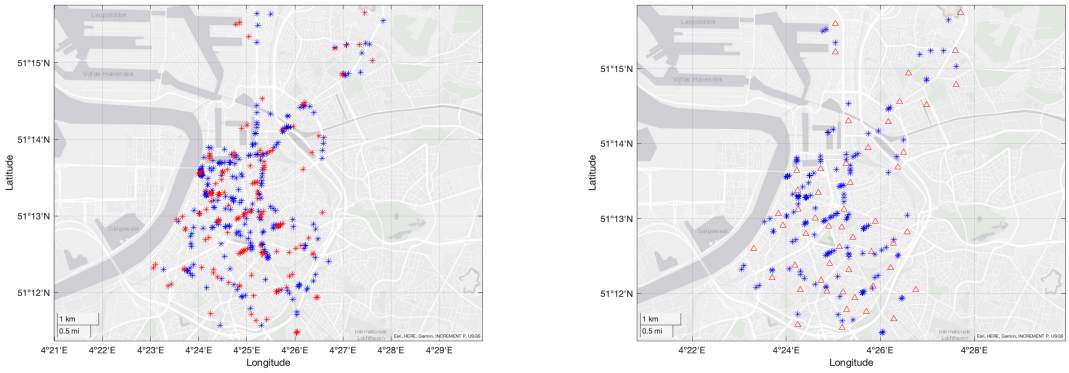
(a) Original and predicted node location points

(b) BS and predicted node location points

Figure 4.8: SVR Prediction Results for the urban area.

4.2. Results for The Rural Area

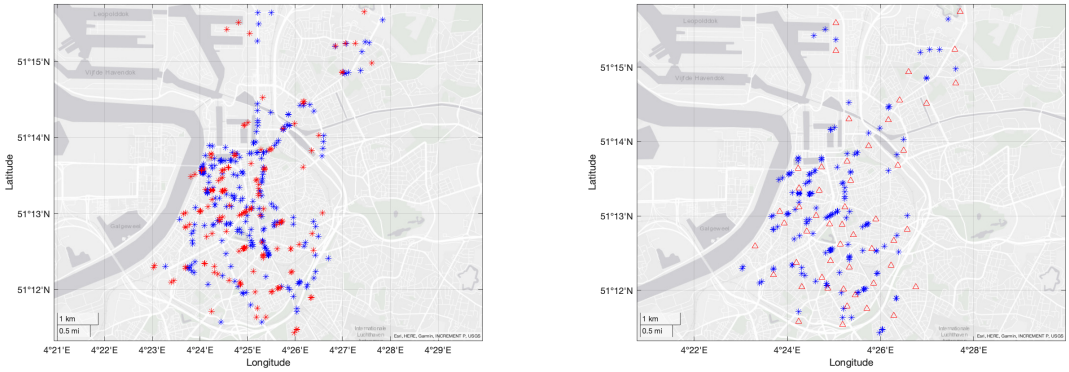
Table 4.1 shows the results for the urban area, where the distance estimation error is around 220 meters, with median values of around 115 meters. For the location estimation error the mean value is around 920 meters with a median value of around 570 meters. This estimation errors confirms that the proposed solution outperforms the proximity and ranging methods explained before and are applicable for solutions where there is no need an



(a) Original and predicted node location points

(b) BS and predicted node location points

Figure 4.9: RFR Prediction Results for the urban area.



(a) Original and predicted node location points

(b) BS and predicted node location points

Figure 4.10: MLPR Prediction Results for the urban area.

ML Model	Distance Estimation		Location Estimation	
	Mean Error	Median Error	Mean Error	Median Error
SVR	223.573 m	114.487 m	921.489 m	556.591 m
RFR	228.875 m	123.723 m	957.744 m	570.258 m
MLP R	235.077 m	117.841 m	947.246 m	577.561 m

Table 4.2: Estimation error results for rural area.

accurate position estimation. The three implemented algorithms show a uniform behavior which adds reliability to the obtained results.

Figure 4.11 shows the plot of the predicted values and the test values of the distance between the BS and the nodes, using the Support Vector Regression ML model. The analysis of this graph shows a good algorithm performance predicting distances between until 3.5 km. This accuracy could be appreciated in the graph where the blue plot represents the behaviour of the predicted values and the gray points represent the shape of the test values. The predicted values achieves most of the test values, and on the contrary to the urban area the predictions can cover with a good percentage the lowest and highest test values. Also, the distance estimation score is the best of the three models with 921.489

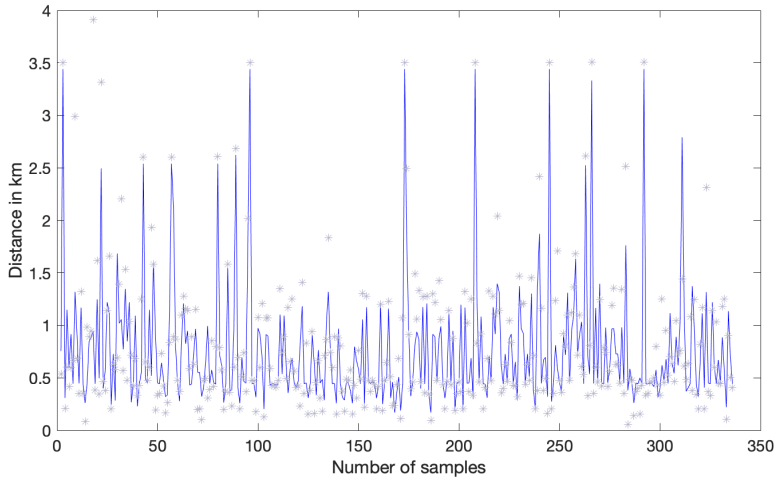


Figure 4.11: SVR distance between predicted and test values.

meters of mean distance estimation error.

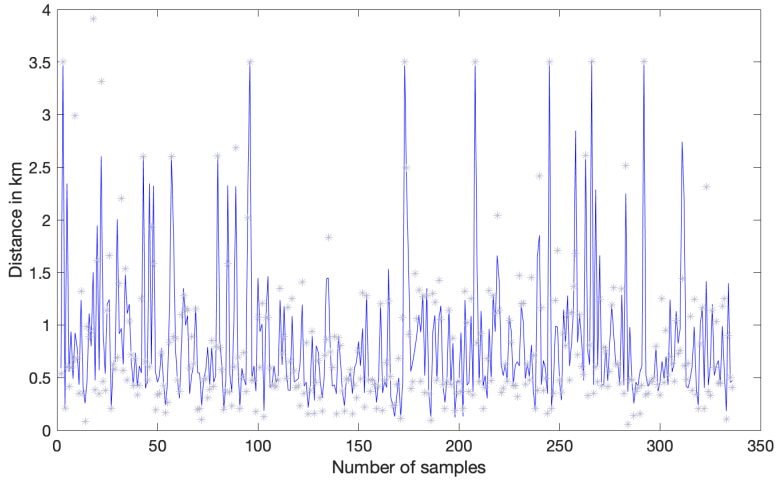


Figure 4.12: RFR distance between predicted and test values.

Figure 4.12 shows the plot of the predicted values and the test values of the distance between the BS and the nodes, using the Random Forest Regression ML model. The analysis of this graph shows a good algorithm performance predicting distances up to 3.5 km. The predicted values achieves most of the test values, covering and concentrating the predictions between two hundred meters until 1.5 km.

Figure 4.13 shows the plot of the predicted values and the test values of the distance between the BS and the nodes, using the Multi-layer Perceptron Regression ML model. The analysis of this graph shows a good algorithm performance predicting distances between 200 meters and 3.5 km. This accuracy could be appreciated in the graph where some test values of more than 3 km are achieved by the prediction values, and also the values under the 200 are well predicted. A well as the RFR, the regressor can not predict values under the one hundred meters boundary.

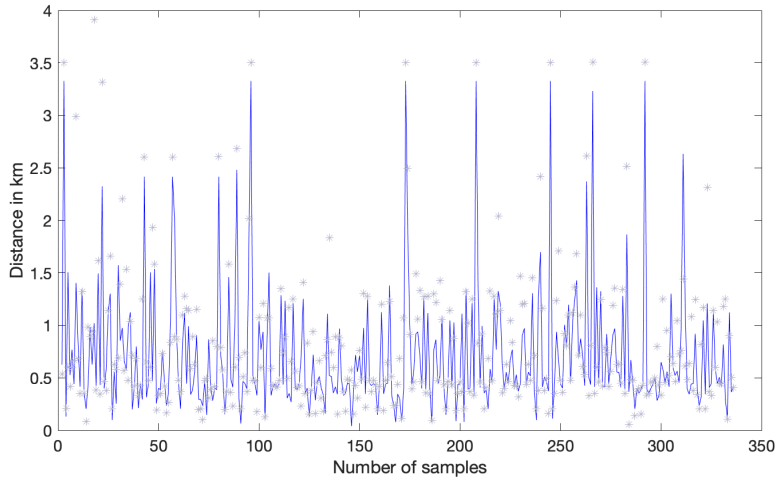


Figure 4.13: MLPR distance between predicted and test values.

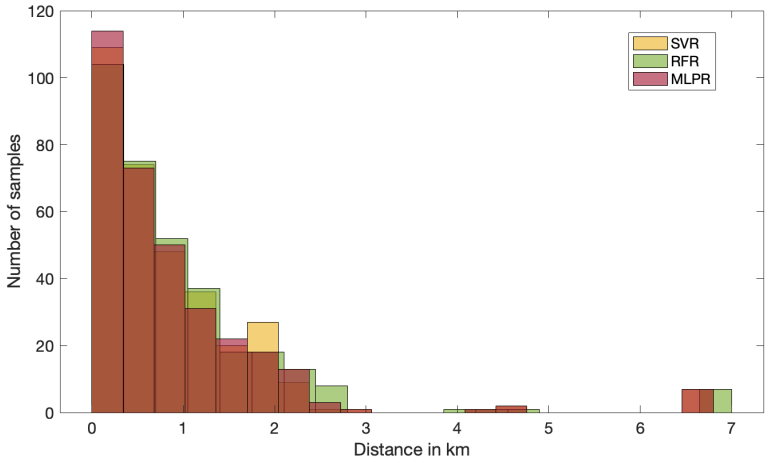


Figure 4.14: Histogram of Error value distribution for the urban area.

Figure 4.14 shows the cumulative distribution of the estimated location errors for the test dataset, where, the values are concentrated below 1 km for the three ML algorithms. This similarity is also supported by the Box plot on the figure 4.15 which compares the location estimation error between the ML models in the rural area. Where, the highlights of this comparison are first the median value of this error around 500 meters. This fact adds reliability to the solution because shows that at least the half of the values are estimated with an accuracy of 500 meters or less. Second, most of the measures are less than 1.2km, with some values up to 2.5 and out-layer values from 2.5km to 7km which adds some noise to the whole accuracy of the models. Finally, the algorithm with the best performance in this comparison is the MLPR with the lower values in the Q3, Q4 and for the out-layers.

The cumulative distribution function of the location estimated errors, denotes a similar behaviour for the three ML models. Where supporting the box plot analysis the 90% of the values are situated below the 2.2 km. This fact represents a good performance of the model achieving the 90 per cent of the values in less than the half of the 7 km that are maximum distance error for this urban analysis.

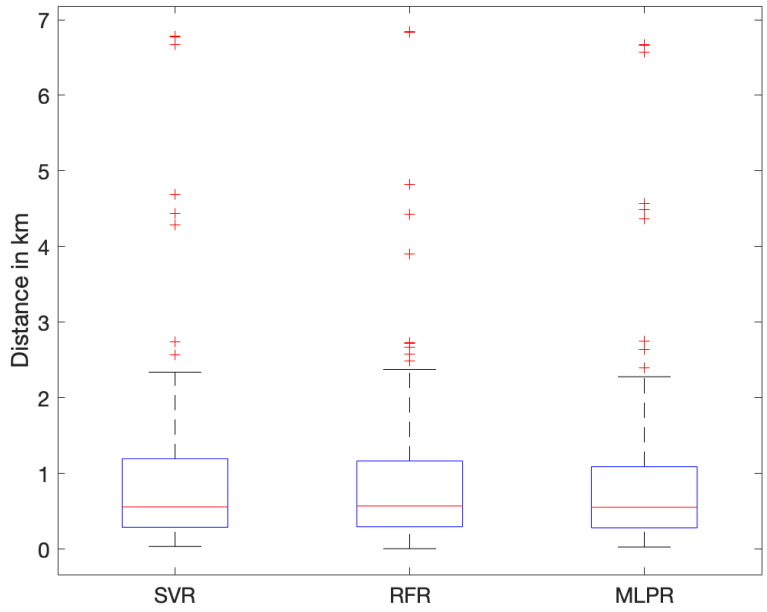


Figure 4.15: Location estimation error for the urban area.

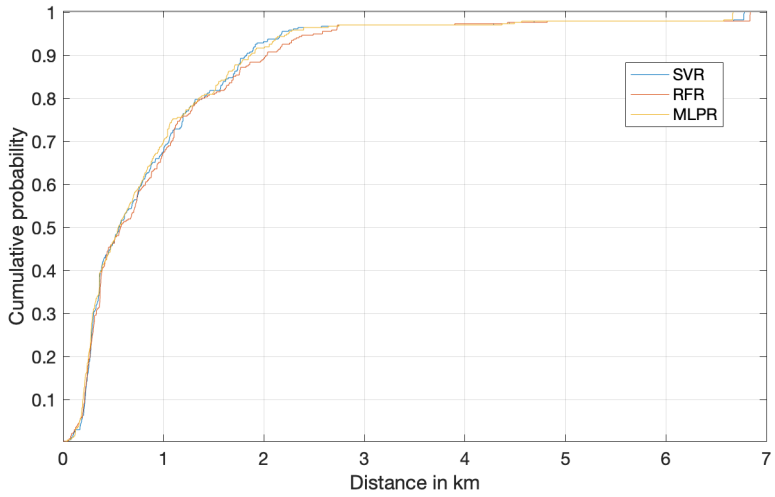
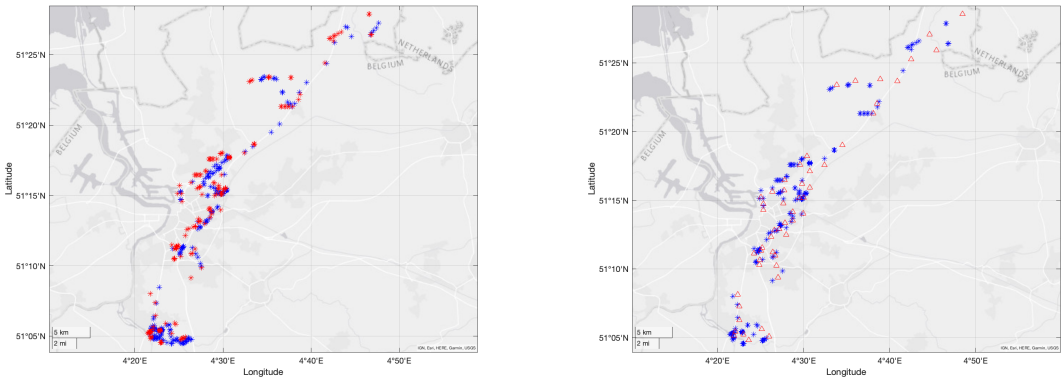


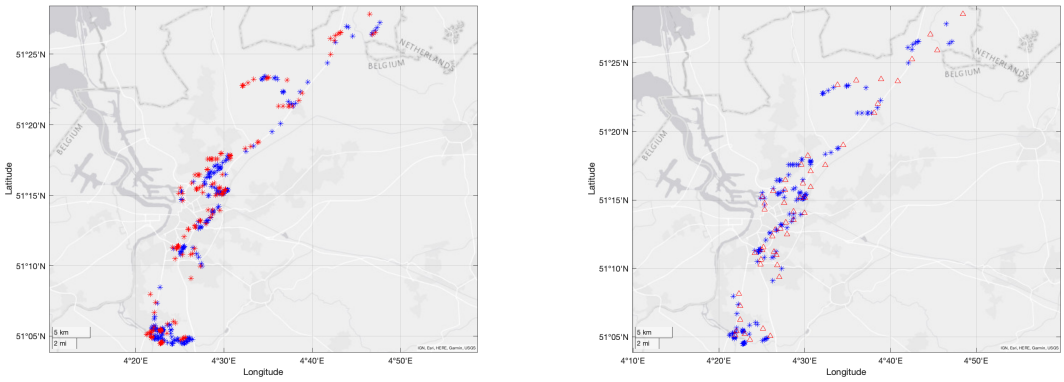
Figure 4.16: Cumulative density function of the location error for the urban area.

Finally the group of maps illustrated in the figures 4.17, 4.18 and 4.19 shows the predicted points and their comparison, first with the original nodes location (literal a) and second with the BS around the predicted nodes location. The three algorithms shows a good performance in terms of the location for the rural area, where the points are covering some highways. This performance allows the use of this solution for logistics tracking, where it is not necessary the exact location. The approach of using the azimuth angle could be perceived in the BS and predicted nodes graphs, because the nodes around the BS are distributed in the shape of 120 degrees of the three antennas on each BS.



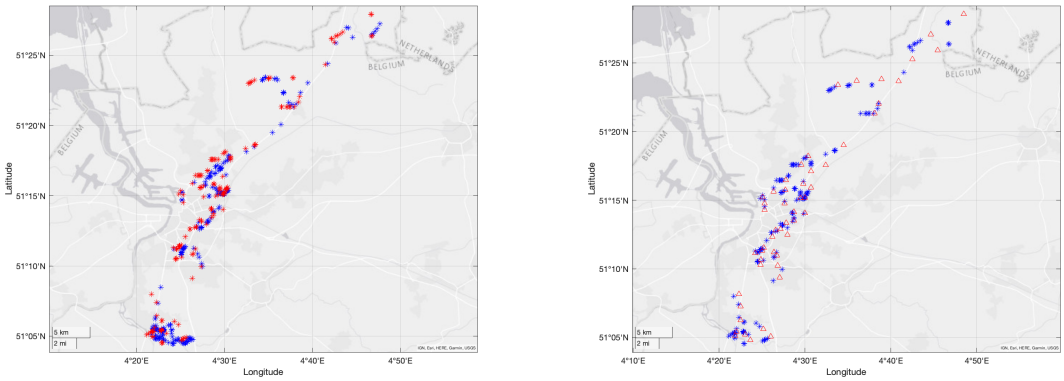
(a) Original and predicted node location points (b) BS and predicted node location points

Figure 4.17: SVR Prediction Results for the rural area.



(a) Original and predicted node location points (b) BS and predicted node location points

Figure 4.18: RFR Prediction Results for the rural area.



(a) Original and predicted node location points (b) BS and predicted node location points

Figure 4.19: MLPR Prediction Results for the rural area.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

In this master thesis project, a real deployed NB-IoT network has been analyzed. This is a crucial point to emphasize because real data was used for all the implemented methodology, and two important conclusions were made according to this fact. First, the deployed NB-IoT network in Antwerp uses the recommendations in release 13 of 3GPP for NB-IoT, which means no use of time difference approaches for localization. Also, the devices could not establish a connection with more than one BS per message, impeding triangulation techniques for localization. Second, the managed data represented a challenge for its interpretation. It was a challenge because the relation between the RSSI values and the distance between the BS and the nodes was not uniformly distributed as in some simulated scenarios; sometimes, a large distance was related to a low RSSI value. After the complete analysis, the reason for these measures was the irregular building distribution in some parts of the city where the line of sight was direct with the bs, and in other urban areas was not.

The solution establishes a new method based on the path loss estimation models but introducing ML to solve the distance estimation. This approach is practical for real scenario applications because it is based on the measures obtained over the urban distribution, collecting all the actual characteristics. This is reflected in outperforming the accuracy compared with the path loss models. The ML algorithms are 40 meters more accurate and 50 meters more accurate for the mean and median error values, respectively [6].

The obtained results for the ML algorithms ratify the methodology, showing an R squared coefficient of more than 0.7 for the three algorithms. This fact means that the chosen variables explain in 70 percent the predicted values. Thus, they are showing congruence in the application of the ML models. The hyper-parameter tuning for the ML models represented a challenge for this project. First, due to the short information about relating work, there were no similar applied models. However, the regression applications were present in other studies where the SVR, RFR, and MLPR parameters were used. Next, the computational capacity for the hyper-parameter tuning needed a GPU, especially for the MLPR. Finally, the google cloud services were enough for the grid-search in the used ML algorithms.

The BS azimuth angle and the angle of the predicted points are directly related to the accuracy of the predictions. The proposed solution sets the angle in the half of the azimuth for each BS. This approach inserts uncertainty into the model because, in most cases, the angle is different or far from the set angle. To illustrate this finding, the figure 4.1 b) shows the error when the angle is set in the middle of the azimuth value. For all the cases, the mean location error is high due to this fact. The distance estimation shows good accuracy, but the accuracy decreases when the final node position is calculated.

After the hyper-parameter tuning and test results, MLPR shows better performance and scalability than the SVR and RFR. The configuration of each algorithm could explain this fact. For example, the MLP could learn differently due to the Backpropagation algorithm, where the independent variable was set before training the algorithm and changing the weights with every epoch indistinct of the size of the database. Also, the possibilities to configure the MLPR are more than with the other algorithms, allowing to manage the

settings of the NN with different layers and different activation functions to improve the performance of the predictions. On the other hand, the RFR and SVR have less computational cost on training the models, and the accuracy levels (Table 4.1 and 4.2) are closer to the MLPR, being an advantage for implement these algorithms rather than the NN.

Finally, the approach of this solution introduces new possibilities to understand the urban and rural environments by collecting measures in place. Comparing the path loss model and the implemented regression model for distance prediction, the models catch exactly the studied environment's characteristics. On the other hand, the path loss models estimate the urban and rural distribution in the city and calculate the distance between the BS and the nodes. This fact allows concluding that the implemented regression models represent exactly the urban and rural characteristics of the city and characterize it for further analysis, opening more related lines for research on this topic.

5.1. Future Work

Based on the analysis and obtained results, three future lines are established for further research. First, the neural network used in this study corresponds to a basic configuration of a multi-layer perceptron. For this reason, a line to continue the research is to improve the characteristics and configuration of the used NN. It could be achieved by changing the number of hidden neurons and layers but principally setting a different type of NN, for example, a non fully connected network, to adapt the model more to the characteristics studied case. This analysis would result in a valuable contribution to improving the predicted location's accuracy by reducing the mean location error. Second, The possibility of improving the accuracy in the estimated location can be achieved by modifying the implemented ML algorithms. This new solution predicts the angle of the node's location and the distance from the BS instead of just the angle. This prediction can be performed by a multi-variable regression, where the new dependent variables were the distance from the BS and the direction angle. This new approach can improve the accuracy of the model meaningfully compared with the current solution, which uses the center of the antenna sector as the direction angle (see figure 1.1). Finally, the methodology process could be performed in a different database, from a different city or a different technology as SigFox or LoRa [46], where there is not the problem of having just one BS. On the contrary, the collected information for LoRa and SigFox shows that the nodes establish a connection with different BS simultaneously, allowing the use of triangulation techniques to predict the estimated location of the nodes.

5.2. Sustainability Considerations

According to the project's sustainability, implementing this solution in a real NB-IoT network will represent a positive contribution to reducing the electronic contamination and the carbon footprint related to the use of this technology. This significant reduction is directly related to saving the energy used for the GPS device and the IoT module to send the message with the current position. The fast growth of IoT and the increasing number of devices turns the energy-saving solutions priority for implementing it in different applications. Farming, Logistics, supply chain, and Industry could reduce the environmental impact by saving energy in the IoT modules. For example, a device can reduce the 25%

of energy consumption. This fact translated to the battery life means that a battery with a five-year duration can be extended until 6.5 years. This improvement represents a direct contribution to the direct environmental impact just in the implementation of the solutions. Indirectly implementing energy-saving solutions like the one proposed in this work can contribute to the best performance of the IoT applications and resource-saving in logistics, supply chain, farming, and home appliances. A final insight from the sustainability point of view is that the energy consumed by this master thesis project by algorithm implementation and the energy spend in this computational cost are easily covered by the future benefits of implementing this and other solutions for localization in IoT networks.

5.3. Ethical Considerations

Since the beginning of the use of IoT, there has existed the ethical dilemma of invading the privacy of the users by covering all the environment with IoT devices for collecting different types of information. According to this point of view, the work realized in this master thesis project allocates the localization solution below the border of invading the users' privacy. It is because many applications are directly related to production and logistics, not daily human life. Furthermore, because the accuracy levels achieved in this work are around 280 meters, which situates the possible area of localization in around 24 squared kilometers in an urban environment, without interfering directly with the users' privacy. Therefore, discarding the indoor localization and other application of IoT devices, for the current applications, the outdoor localization with an allowed accuracy can improve human life without interfering with their privacy and the ethical restrictions for this technology.

ACRONYMS

3GPP	3rd Generation Partnership Project
AI	Artificial Intelligence
ANN	Artificial Neural Network
BP	Backpropagation
BS	Base Station
CART	Classification and Regression Tree
CSI	Channel State Information
dB	Decibels
eDrx	Extended Discontinuous Reception
eNodeB	Evolved Node B
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HSS	Home Subscribed Server
IoT	Internet of Things
KNN	K Nearest Neighbors
LPWAN	Low Power Wide Area Network
LTE	Long Term Evolution
LTE-M	Long Term Evolution Mobile
MAE	Mean Absolute Error
MED	Median Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MLPR	Multi-Layer Perceptron Regressor
MME	Mobility Management Entity
MSE	Mean Squared Error
MTU	Maximum Transmission Unit
NB-IoT	Narrowband Internet of Things
NN	Neural Networks
NPBCH	Narrowband Physical Broadcast Channel
NPDCCH	Narrowband Physical Downlink Control Channel
NPDSCH	Narrowband Physical Downlink Shared Channel
NPRACH	Narrowband Physical Random-Access Channel
NPUSCH	Narrowband Physical Uplink Shared Channel
NRS	Narrowband Reference Signal
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OTDoA	Observed Time Difference of Arrival
PDN	Packet Data Network
PSM	Power Saving Mode
RB	Resource Block
RF	Random Forest
RFR	Random Forest Regressor
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator

SC-FDMA	Single-Carrier Frequency-Division Multiple- Access
S-GW	Service Gateway
SVM	Support Vector Machines
SVR	Support Vector Regressor
TDoA	Time Difference of Arrival
UE	User Equipment

BIBLIOGRAPHY

[1] Oracle Mexico. ¿Qué es Internet of Things (IoT)? *Internet of Things*, page 6, 2021. 1

[2] José Renato de Mello Gonçalves. El crecimiento de IoT en un mundo cada vez más. *Orange Business Services*, page 1, 2021. 1

[3] IOT Factory Group. Overview of IOT Networks. *Wired and Short Range wireless networks*, page 3, 2021. 1

[4] THALES Group. IoT-enabled Asset Tracking. *Digital Identity and Secutirty*, page 3, 2021. 1

[5] Guus Leenders, Gilles Callebaut, Liesbet Van Der Perre, and Lieven De Strycker. An Experimental Evaluation of Energy Trade-Offs in Narrowband IoT. pages 1–5, 2020. 1

[6] Thomas Janssen, Maarten Weyn, and Rafael Berkvens. A Primer on Real-world RSS-based Outdoor NB-IoT Localization. 1, 2, 6, 11, 12, 17, 18, 23, 45

[7] Nb-iot Indoor Localization, Qianwen Song, Songtao Guo, Xing Liu, and Yuanyuan Yang. CSI Amplitude Fingerprinting-Based. 5(3):1494–1504, 2018. 1

[8] Ahmed Abdel Ghany, Bernard Uguen, and Dominique Lemur. A Robustness Comparison of Measured Narrowband CSI vs RSSI for IoT Localization. 2020. 1

[9] Thomas Janssen, Rafael Berkvens, and Maarten Weyn. Comparing Machine Learning Algorithms for RSS-Based Localization in LPWAN. *Lecture Notes in Networks and Systems*, 96(October 2019):726–735, 2020. 1, 2, 14

[10] Hazem Sallouha, Alessandro Chiumento, and Sofie Pollin. Localization in long-range ultra narrow band IoT networks using RSSI. *IEEE International Conference on Communications*, 2017. 1, 2

[11] Kamiar Radnosrati, Gustaf Hendeby, Carsten Fritsche, Fredrik Gunnarsson, and Fredrik Gustafsson. Performance of OTDOA positioning in narrowband IoT systems. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, 2017-Octob:1–7, 2018. 1, 11

[12] Mauro Salomon, Stefan Lippuner, Matthias Korb, and Qiuting Huang. Implementation and performance evaluation of cellular NB-IoT OTDOA positioning. *2020 IEEE/ION Position, Location and Navigation Symposium, PLANS 2020*, pages 1365–1371, 2020. 1

[13] Fei Tong, Yuyi Sun, and Shibo He. On Positioning Performance for the Narrow-Band Internet of Things: How Participating eNBs Impact? *IEEE Transactions on Industrial Informatics*, 15(1):423–433, 2019. 1

[14] Hazem Sallouha, Alessandro Chiumento, Sreeraj Rajendran, and Sofie Pollin. Localization in Ultra Narrow Band IoT Networks: Design Guidelines and Tradeoffs. *IEEE Internet of Things Journal*, 6(6):9375–9385, 2019. 1, 2

[15] Filip Lemic, Vlado Handziski, Michiel Aernouts, Thomas Janssen, Rafael Berkvens, Adam Wolisz, Senior Member, Jeroen Famaey, and Senior Member. Regression-Based Estimation of Individual Errors in Fingerprinting Localization. *IEEE Access*, 7:33652–33664, 2019. 1, 2

[16] Stephen Farrell. Low-Power Wide Area Network (LPWAN) Overview RFC 8376. *Internet Engineering Task Force (IETF)*, RFC 8376:1689–1699, 2018. 5, 6, 7, 8, 9

[17] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT. *2018 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2018*, pages 197–202, 2018. 5, 6

[18] Olof Liberg, Mårten Sundberg, Y.-P. Eric Wang, Johan Bergman, and Joachim Sachs. *NB-IoT*. 2018. 5, 6, 7, 8, 9, 10

[19] Emmanuel M. Migabo, Karim D. Djouani, and Anish M. Kurien. The Narrowband Internet of Things (NB-IoT) Resources Management Performance State of Art, Challenges, and Opportunities. *IEEE Access*, 8:97658–97675, 2020. 6, 7, 8, 9, 10

[20] Orange Belgium. Orange Belgium new IoT Network reaches 100massive Internet of Things solutions, 2017. 6, 17

[21] Mona Bakri Hassan, Elmustafa Sayed Ali, Rania A. Mokhtar, Rashid A. Saeed, and Bharat S. Chaudhari. *NB-IoT: concepts, applications, and deployment challenges*. INC, 2020. 7, 8, 9, 10

[22] Ahmed Elhaddad, Heiko Bruckmeyer, Markus Hertlein, and Georg Fischer. Energy Consumption Evaluation of Cellular Narrowband Internet of Things (NB-IoT) Modules. pages 1–5, 2020. 7, 9, 11

[23] Rapeepat Ratasuk, Nitin Mangalvedhe, Zhilan Xiong, Michel Robert, and David Bhatoolaul. Enhancements of Narrowband IoT in 3GPP Rel-14 and Rel-15. *2017 IEEE Conference on Standards for Communications and Networking (CSCN) Enhancements*, pages 60–65, 2017. 8, 9, 11

[24] Radheshyam Singh and Ying Yan. An Experimental Evaluation of NB-IoT Coverage and Energy Consumption. 12

[25] Jui Chan Huang, Kuo Min Ko, Ming Hung Shu, and Bi Min Hsu. Application and comparison of several machine learning algorithms and their integration models in regression problems. *Neural Computing and Applications*, 32(10):5461–5469, 2020. 12, 15

[26] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Regression Models for Ordinal Data : A Machine Learning Approach. *Technical Report, TU Berlin. TR-99/03*, (February), 1999. 12

[27] Ahmet Selman Bozkir, Hakan Ahmet Nefeslioglu, Ogan Kartal, Ebru Akcapinar Sezer, and Candan Gokceoglu. Geological strength index (GSI) determination by local image descriptors and machine learning methods. *Human-Computer Interaction*, (February):175–210, 2019. 13, 14

[28] S. Kavitha, S. Varuna, and R. Ramya. A comparative analysis on linear regression and support vector regression. *Proceedings of 2016 Online International Conference on Green Engineering and Technologies, IC-GET 2016*, 2017. 13

[29] Freek Stulp and Olivier Sigaud. Many regression algorithms, one unified model: A review. *Neural Networks*, 69:60–79, 2015. 13, 14

[30] Brad Boehmke and Brandon Greenwell. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. 2019. 13, 14, 15, 16

[31] ALEX J. SMOLA and BERNHARD SCHOLKOPF. A tutorial on Support Vector Regression. *Statistics and Computing*, 14:199–222, 2004. 13

[32] S. B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. *Hyperfine Interactions*, 237(1):1–8, 2016. 14

[33] Hoang Nguyen, Thanh Vu, Thuc P. Vo, and Huu Tai Thai. Efficient machine learning models for prediction of concrete strengths. *Construction and Building Materials*, 266:120950, 2021. 14

[34] Eac Each. Chpater 6: feedforw eedforward ard Deep Net Netw w orks. (1):1–66, 2015. 15, 16

[35] Jayashri; Anupama Kumar Madalgi. Congestion Detection in Wireless Sensor Networks Using MLP and cLASSIFICATION by rEGRESSION. 15

[36] Stad Antwerpen. Urban development in Antwerp. *Urban Development in Antwerp*, page 170, 2012. 17, 22

[37] Atlasofurbanexpansion Org. Atlas of Urban Expansion - Antwerp, 2021. 17, 22

[38] Google (n.d). Map of Antwerp City, 2021. 17

[39] Jeff Reback, Wes McKinney, Jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, Gfyoung, Sinhrks, Adam Klein, Matthew Roeschke, Simon Hawkins, Jeff Tratner, Chang She, William Ayd, Terji Petersen, Marc Garcia, Jeremy Schendel, Andy Hayden, MomIsBestFriend, Vytautas Jancauskas, Pietro Battiston, Skipper Seabold, Chris-b1, H-vetinari, Stephan Hoyer, Wouter Overmeire, Alimcmaster1, Kaiqi Dong, Christopher Whelan, and Mortada Mehyar. pandas-dev/pandas: Pandas 1.0.3. *Zenodo*, latest, mar 2020. 18, 21

[40] Thomas Janssen, Rafael Berkvens, and Maarten Weyn. RSS-based localization and mobility evaluation using a single NB-IoT cell. *Sensors (Switzerland)*, 20(21):1–14, 2020. 18

[41] Ruben Oliveira, Lucas Guardalben, and Susana Sargento. Long range communications in urban and rural environments. *Proceedings - IEEE Symposium on Computers and Communications*, pages 810–817, 2017. 22

[42] Thomas Janssen, Rafael Berkvens, and Maarten Weyn. Benchmarking RSS-based localization algorithms with LoRaWAN. *Internet of Things*, 11(1):100235, 2020. 23

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 26, 27, 28, 29, 30, 31

[44] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 31

[45] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 9.10.0 (R2021a)*, 2021. 35

[46] Michiel Aernouts, Rafael Berkvens, Koen Van Vlaenderen, and Maarten Weyn. Sigfox and LoRaWAN datasets for fingerprint localization in large urban and rural areas. *Data*, 3(2):1–15, 2018. 46

APPENDICES

APPENDIX A. IMPLEMENTED ALGORITHMS

A.1. Pre-Procesing Algorithms

A.1.1. Exploring Databases

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar  3 14:08:07 2021
@author: jimmy
"""
print ("Exploratory Analysis from NB-IoT DB")
import pandas as pd
import pandas_profiling

#Reading the two databases

#df = pd.read_csv('2019_dataset_antwerp_nbiot_messages.csv',
header=0)
#df2 = pd.read_csv('2019_dataset_antwerp_base_stations.csv',
header=0)
#, encoding = "ISO-8859-1"

#Printing explorative information
print(df.head())
print(df.tail())
print(df.shape)
print(df.info())
print(df.describe())

#Generating messages database profile
profile = df.profile_report(title='NB-IoT')
profile.to_file("basicsNBIoT.html")

#Generating BS database profile
profile2 = df2.profile_report(title='NB-IoT-BS')
profile2.to_file("basicsNBIoTBaseStations.html")
```

A.1.2. Data Merging and Distance Calculation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 18 20:11:20 2021
```

```
@author: jimmy
"""
#Adding necessary libraries

from math import sqrt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from haversine import haversine, Unit

df_MS = pd.read_excel('MS_BS_dataAZ.xlsx', sheet_name='Sheet1')

#Adding new variables

df_MS["distance_BS_alt"]=""
df_MS["Nlat"]=""
df_MS["Nlong"]=""
df_MS["Urban"]=""
df_MS["Rural"]=""
df_MS["eliminar"]=""

df_MS = df_MS[["timestamp", "id", "rssi","latitude","longitude",
"altitude","Lat_BS","Lon_BS","alt_BS","distance_BS","distance_
BS_alt","Nlat","Nlong","Urban","Rural","Azimuth","eliminar"]]

print(df_MS.head()) #Checking variable

l_MS = len(df_MS)# database length
c = 0 #The counter

print(df_MS.iloc[2,1]) #Checking variable
'''
#Merging information from 2 DB
for i in range(0,l_MS):
    for j in range(0,l_BS):
        if df_MS.iloc[i,1] == df_BS.iloc[j,0]:
            print("BS Encontrada")
            df_MS.iloc[i,6] = df_BS.iloc[j,1]  #Assigning lattitude
            df_MS.iloc[i,7] = df_BS.iloc[j,2]  #Assigning longitude
            df_MS.iloc[i,8] = df_BS.iloc[j,3]  #Assigning altittude

#Debugging zeros
for i in range(1607,1612):
    df_MS.iloc[i,6] = 0
    df_MS.iloc[i,7] = 0
    df_MS.iloc[i,8] = 0

print("Merging ended")
```

```
'''
#Calculating device distance from BS

for i in range(0,l_MS):
    if df_MS.iloc[i,5] < 0:
        altitude = float(df_MS.iloc[i,8])
        distance = sqrt((pow(altitude/1000, 2))+
        (pow(float(df_MS.iloc[i,9]), 2)))
        df_MS.iloc[i,10] = distance
        print(distance*1000,"km")
    if df_MS.iloc[i,5] >= 0:
        altitude = (float(df_MS.iloc[i,5])-float(df_MS.iloc[i,8]))
        dis1=float(df_MS.iloc[i,9])
        distance = sqrt((pow(altitude/1000, 2))+(pow(dis1, 2)))
        df_MS.iloc[i,10] = distance
        print(distance*1000,"km")

print("Distance calculation ended")

for i in range(0,l_MS):
    df_MS.iloc[i,11]=(df_MS.iloc[i,6])-51
    df_MS.iloc[i,12]=(df_MS.iloc[i,7])-4
    df_MS.iloc[i,2]=(df_MS.iloc[i,2])*-1

for i in range(0,l_MS):
    if df_MS.iloc[i,3] > 51.1922 and df_MS.iloc[i,3] < 51.2359
    and df_MS.iloc[i,4] > 4.38455 and df_MS.iloc[i,4] < 4.44806 :

        #if df_MS.iloc[i,9] > 0.4:
        print('es urbano')
        df_MS.iloc[i,13]=1
        df_MS.iloc[i,14]=0
        if df_MS.iloc[i,10]>0.8:
            print('eliminar')
            df_MS.iloc[i,16]=1
    else:
        print('es rural')
        df_MS.iloc[i,14]=1
        df_MS.iloc[i,13]=0

for i in range(0,l_MS):
    if df_MS.iloc[i,3] > 51.2359 and df_MS.iloc[i,3] < 51.2607
    and df_MS.iloc[i,4] > 4.42 and df_MS.iloc[i,4] < 4.46421 :

        #if df_MS.iloc[i,9] > 0.4:
        print('es urbano')
        df_MS.iloc[i,13]=1
        df_MS.iloc[i,14]=0
```

```
        if df_MS.iloc[i,10]>0.8:
            print('eliminar')
            df_MS.iloc[i,16]=1
#Filtering nodes by distance from BS
'''
    #checking distance more than 1km
    if df_MS.iloc[i,9] > 1:
        print('mas de 1.5')
        df_MS.iloc[i,15]=1
    else:
        df_MS.iloc[i,15]=0
    #checking distance more than 1.2km
    if df_MS.iloc[i,9] > 1.2:
        print('mas de 1.5')
        df_MS.iloc[i,16]=1
    else:
        df_MS.iloc[i,16]=0
    #checking distance more than 1.5km
    if df_MS.iloc[i,9] > 1.5:
        print('mas de 1.5')
        df_MS.iloc[i,17]=1
    else:
        df_MS.iloc[i,17]=0
    #checking distance more than 2km
    if df_MS.iloc[i,9] > 2:
        print('mas de 1.5')
        df_MS.iloc[i,18]=1
    else:
        df_MS.iloc[i,18]=0
'''

#Exporting DB to Excel
df_MS.to_excel("MS_BS_UyRAZ.xlsx")
df_MS.to_csv("MS_BS_UyRAZ.csv", index = False)
print("DB Exported")
```

A.2. ML Regression Algorithms

A.2.1. Support Vector Regressor

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug  2 01:51:33 2021
@author: jimmy
"""
```

```
#Importing libraries
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn import metrics
from haversine import haversine, Unit
from sklearn.model_selection import ShuffleSplit


#IMPORT DATABASE
df_MS = pd.read_csv('Rur_SQ.csv')
df_MS = pd.read_csv('Urb_SQ.csv')

#Defining independent values
independent = df_MS.iloc[:, [2,11,12,8,15]].values

#Defining dependent values (distance)
dependent = df_MS.iloc[:, [10,6,7,8,15,3,4]].values #values for
merge gps position

#Scaling data

x_train1,x_test1,y_train,y_test=train_test_split(independent,
dependent, test_size=0.2, random_state=3) #random_state=3

scaler.fit(x_train1)#1
x_train = scaler.transform(x_train1)
x_test = scaler.transform(x_test1)

'''
#GridSearch Hyperparameters
print('Applying GScv')
param_grid = { 'C':[0.1,1,100],'kernel':['rbf','poly','sigmoid',
'linear'],'degree':[1,2,3,4,5], 'epsilon':[1, 0.1, 0.01]}
grid = GridSearchCV(SVR(),param_grid)
grid.fit(x_train,y_train)


#Obtaining Grid Results
print('GRIDSEARCH RESULTS')
print(grid.best_params_)
print(grid.score(x_test,y_test))
'''

#Applying the SVM Algorithm
```

```
regressor = SVR(kernel = 'rbf', degree=1, gamma='scale',C=1.0,
epsilon=0.1)
#regressor = SVR(kernel = 'rbf')
#regressor.fit(Xt, yt)
regressor.fit(x_train, y_train[:, 0])
#Show predictions
y_pred = regressor.predict(x_test)

#y_pred =regressor.predict(sc_X.transform(Xt))
#y_pred = sc_y.inverse_transform(y_pred)
#print(y_pred.head())

#print(df_MS['rssi'].mean())

#CROSS VALIDATION
from sklearn.model_selection import cross_val_score
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
scores_svr = []
scores_svr = cross_val_score(SVR(kernel = 'rbf', degree=1,
gamma='scale',C=1.0, epsilon=0.1), indep, dependent[:, 0],cv=cv)
print("%0.2f accuracy with a standard deviation of %0.2f" %
(scores_svr.mean(), scores_svr.std()))

#SVM metrics
score=regressor.score(x_test, y_test[:, 0])
print(score)
parameters= regressor.get_params()
print(parameters)

#Printing results
dataset = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test[:, 0],
'Lat_BS': y_test[:, 1],'Lon_BS': y_test[:, 2],'Alt_BS': y_test[:, 3]
,'Azimuth': y_test[:, 4],'Node_Lat': y_test[:, 5],'Node_Lon':
y_test[:, 6]}, columns=['y_pred', 'y_test','Lat_BS','Lon_BS',
'Alt_BS','Azimuth','Node_Lat','Node_Lon'])

#Adding new variables for calculus
plt.plot(y_test[:, 0],'o', color = "gray")
dataset["error"]=""
dataset["New_Lat"]=""
dataset["New_lon"]=""
dataset["Distance_Pred_Test"]=""
plt.plot(y_pred)
plt.title("Results")
plt.xlabel("Number of values")
plt.ylabel("Distance Km")
plt.show()
```

```
l_dt = len(dataset)
for i in range(0,l_dt):
    dataset.iloc[i,8]=abs(float(dataset.iloc[i,0])
    -float(dataset.iloc[i,1]))

fig = plt.figure(figsize =(10, 7))
toplot = dataset.iloc[:, 2].values
plt.boxplot(toplot)
plt.show()

# Predicting the location points
import math
R = 6378.1 #Radius of the Earth
for i in range(0,l_dt):
    #NEW POINT CALCULATION
    bear=float(dataset.iloc[i,5])
    bear_rad=math.radians(bear)
    lat1 = math.radians(float(dataset.iloc[i,2]))
    #lat point converted to radians
    lon1 = math.radians(float(dataset.iloc[i,3]))
    #long point converted to radians
    lat2 = math.asin(math.sin(lat1)*math.cos(float(
    dataset.iloc[i,0])/R) + math.cos(lat1)
    *math.sin(float(dataset.iloc[i,0])/R)*math.cos(bear_rad))
    dataset.iloc[i,9]=math.degrees(lat2)
    lon2 = lon1 + math.atan2(math.sin(bear_rad)*math.sin(float
    (dataset.iloc[i,0])/R)*math.cos(lat1),
           math.cos(float(dataset.iloc[i,0])/R)-math.sin(lat1)
           *math.sin(lat2))
    dataset.iloc[i,10]=math.degrees(lon2)
    #DISTANCE BETWEEN PREDICTED AND TEST POINTS
    distance = haversine((dataset.iloc[i,6], dataset.iloc[i,7]),
    (dataset.iloc[i,9], dataset.iloc[i,10]))
    dataset.iloc[i,11]=distance

print('PREDICTION ANALYSYS')
print(dataset['Distance_Pred_Test'].mean())
print(dataset['Distance_Pred_Test'].median())
print(dataset['Distance_Pred_Test'].std())
print(dataset['Distance_Pred_Test'].max())
print(dataset['Distance_Pred_Test'].min())

##ADITTIONAL INFORMATION
print('Mean Absolute Error (MAE):', metrics.mean_absolute_error
(y_test[:, 0], y_pred))
print('Mean Squared Error (MSE):', metrics.mean_squared_error
(y_test[:, 0], y_pred))
print('Root Mean Squared Error (RMSE):', metrics.mean_squared_error
```

```
(y_test[:, 0], y_pred, squared=False))
#print('Mean Absolute Percentage Error (MAPE):', metrics.mean_
absolute_percentage_error(y_test, y_pred))
print('Explained Variance Score:', metrics.explained_variance_score
(y_test[:, 0], y_pred))
print('Max Error:', metrics.max_error(y_test[:, 0], y_pred))
#print('Mean Squared Log Error:', metrics.mean_squared_log_error
(y_test, y_pred))
print('Median Absolute Error:', metrics.median_absolute_error
(y_test[:, 0], y_pred))
print('R^2:', metrics.r2_score(y_test[:, 0], y_pred))
#print('Mean Poisson Deviance:', metrics.mean_poisson_deviance
(y_test[:, 0], y_pred))
#print('Mean Gamma Deviance:', metrics.mean_gamma_deviance
(y_test[:, 0], y_pred))

print('RSSI ANALYSYS')
print("Mean",df_MS['rssi'].mean())
print("Standart_DEV",df_MS['rssi'].std())
print("Max_Value",df_MS['rssi'].max())
print('Min_value',df_MS['rssi'].min())

print('DISTANCE ANALYSYS')
print(df_MS['distance_BS'].mean())
print(df_MS['distance_BS'].std())
print(df_MS['distance_BS'].max())
print(df_MS['distance_BS'].min())

print('EXPORTING DATABASE')
dataset.to_excel("datasetSVR.xlsx")
print('DB Exported')
```

A.2.2. Random Forest Regressor

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 23 17:26:53 2021

@author: jimmy
"""

import numpy as np
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from haversine import haversine, Unit

#IMPORT DATABASE

#IMPORT DATABASE
df_MS = pd.read_csv('Rur_SQ.csv')
df_MS = pd.read_csv('Urb_SQ.csv')

#Defining independent values
independent = df_MS.iloc[:, [2,11,12,8,15]].values

#Defining dependent values (distance)
dependent = df_MS.iloc[:, [10,6,7,8,15,3,4]].values #values
for merge gps position

#Scaling data

x_train1,x_test1,y_train,y_test=train_test_split(independent,
dependent, test_size=0.2, random_state=3) #random_state=3

scaler.fit(x_train1)#1
x_train = scaler.transform(x_train1)
x_test = scaler.transform(x_test1)

'''
#GRIDSEARCH CV
# Create the random grid
random_grid = {'n_estimators': [10, 50, 80, 100],
               'max_features': ['auto','sqrt'],
               'max_depth': [2, 4, 8, 10, None],
               'min_samples_split': [2, 8],
               'min_samples_leaf': [1, 2],
               'bootstrap': [True, False]}
print(random_grid)


# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all
```

```
available cores
print('Applying GScv')
rf_random = GridSearchCV(rf, random_grid)
# Fit the random search model
rf_random.fit(x_train, y_train)

print('GRIDSEARCH RESULTS')
print(rf_random.best_params_)
print(rf_random.score(x_test,y_test))
'''

#Applying the Random Forest Algorithm
regressor = RandomForestRegressor(max_features=None, min_samples
_leaf=1,min_samples_split=10 ,n_estimators = 100, random_state=3)
#,random_state = 0
#regressor = RandomForestRegressor(max_features=None, min_samples
_leaf=1,min_samples_split=10 ,n_estimators = 50, random_state=3)
#,random_state = 0
#regressor.fit(Xt, yt)
regressor.fit(x_train, y_train[:, 0])

#Show predictions
y_pred = regressor.predict(x_test)

score=regressor.score(x_train, y_train[:, 0])
print(score)

parameters= regressor.get_params()
print(parameters)

#CROSS VALIDATION

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
scores_rfr = []
scores_rfr = cross_val_score(RandomForestRegressor(max_features=
None, min_samples_leaf=1,min_samples_split=10 ,n_estimators = 100)
, independent, dependent[:, 0],cv=cv)
print("%0.2f accuracy with a standard deviation of %0.2f" %
(scores_rfr.mean(), scores_rfr.std()))

#Printing Results

dataset = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test[:, 0]
,'Lat_BS': y_test[:, 1],'Lon_BS': y_test[:, 2],'Alt_BS': y_test[:, 3]
,'Azimuth': y_test[:, 4],'Node_Lat': y_test[:, 5],'Node_Lon':
```

```
y_test[:, 6]}, columns=['y_pred', 'y_test','Lat_BS','Lon_BS',
'Alt_BS','Azimuth','Node_Lat','Node_Lon'])

plt.plot(y_test[:, 0],'o', color = "gray")
dataset["error"]=""
dataset["New_Lat"]=""
dataset["New_lon"]=""
dataset["Distance_Pred_Test"]=""
plt.plot(y_pred)
plt.title("Results")
plt.xlabel("Number of values")
plt.ylabel("Distance Km")
plt.show()

l_dt = len(dataset)

for i in range(0,l_dt):
    dataset.iloc[i,8]=abs(float(dataset.iloc[i,0])-float
    (dataset.iloc[i,1]))

fig = plt.figure(figsize =(10, 7))
toplot = dataset.iloc[:, 2].values
plt.boxplot(toplot)
plt.show()

####### PREDICTION POINT PART

import math

R = 6378.1 #Radius of the Earth

for i in range(0,l_dt):
    #NEW POINT CALCULATION
    bear=float(dataset.iloc[i,5])
    bear_rad=math.radians(bear)
    lat1 = math.radians(float(dataset.iloc[i,2]))
    #lat point converted to radians
    lon1 = math.radians(float(dataset.iloc[i,3]))
    #long point converted to radians
    lat2 = math.asin(math.sin(lat1)*math.cos(float
    (dataset.iloc[i,0])/R) +
           math.cos(lat1)*math.sin(float(dataset.iloc
           [i,0])/R)*math.cos(bear_rad))
    dataset.iloc[i,9]=math.degrees(lat2)
    lon2 = lon1 + math.atan2(math.sin(bear_rad)*math.sin
    (float(dataset.iloc[i,0])/R)*math.cos(lat1),
           math.cos(float(dataset.iloc[i,0])/R)-math.sin
           (lat1)*math.sin(lat2))
```

```
    dataset.iloc[i,10]=math.degrees(lon2)
    #DISTANCE BETWEEN PREDICTED AND TEST POINTS
    distance = haversine((dataset.iloc[i,6], dataset.iloc[i,7]),
    (dataset.iloc[i,9], dataset.iloc[i,10]))
    dataset.iloc[i,11]=distance

print('EXPORTING DATABASE')
dataset.to_excel("datasetRFR.xlsx")
print('DB Exported')


print('PREDICTION ANALYSYS')

print(dataset['Distance_Pred_Test'].mean())
print(dataset['Distance_Pred_Test'].median())
print(dataset['Distance_Pred_Test'].std())
print(dataset['Distance_Pred_Test'].max())
print(dataset['Distance_Pred_Test'].min())

##ADITTIONAL INFORMATION
print('Mean Absolute Error (MAE):', metrics.mean_absolute_error
(y_test[:, 0], y_pred))
print('Mean Squared Error (MSE):', metrics.mean_squared_error
(y_test[:, 0], y_pred))
print('Root Mean Squared Error (RMSE):', metrics.mean_squared_error
(y_test[:, 0], y_pred, squared=False))
#print('Mean Absolute Percentage Error (MAPE):', metrics.
mean_absolute_percentage_error(y_test, y_pred))
print('Explained Variance Score:', metrics.explained_variance_score
(y_test[:, 0], y_pred))
print('Max Error:', metrics.max_error(y_test[:, 0], y_pred))
#print('Mean Squared Log Error:', metrics.mean_squared_log_error
(y_test, y_pred))
print('Median Absolute Error:', metrics.median_absolute_error
(y_test[:, 0], y_pred))
print('R^2:', metrics.r2_score(y_test[:, 0], y_pred))
#print('Mean Poisson Deviance:', metrics.mean_poisson_deviance
(y_test[:, 0], y_pred))
#print('Mean Gamma Deviance:', metrics.mean_gamma_deviance
(y_test[:, 0], y_pred))

print('RSSI ANALYSYS')
print("Mean",df_MS['rssi'].mean())
print("Standart_DEV",df_MS['rssi'].std())
print("Max_Value",df_MS['rssi'].max())
print('Min_value',df_MS['rssi'].min())

print('DISTANCE ANALYSYS')
```

```
print(df_MS['distance_BS'].mean())
print(df_MS['distance_BS'].std())
print(df_MS['distance_BS'].max())
print(df_MS['distance_BS'].min())
```

A.2.3. Multi-layer Perceptron Regressor

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 23 18:08:10 2021
@author: jimmy
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn import metrics
from sklearn import preprocessing
from haversine import haversine, Unit

import pathlib
from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)

#IMPORT DATABASE
df_MS = pd.read_csv('Rur_SQ.csv')
df_MS = pd.read_csv('Urb_SQ.csv')

#Defining independent values
independent = df_MS.iloc[:, [2,11,12,8,15]].values

#Defining dependent values (distance)
dependent = df_MS.iloc[:, [10,6,7,8,15,3,4]].values #values
for merge gps position

#Scaling data
x_train1,x_test1,y_train,y_test=train_test_split(independent,
dependent, test_size=0.2, random_state=3)
```

```
scaler.fit(x_train1)#1
x_train = scaler.transform(x_train1)
x_test = scaler.transform(x_test1)

# Set the input shape
#input_shape = (4,)

#Defining model for gridsearch
#REGRESSION WITH NN
# Keras NN Regression

def build_model():
  model = Sequential([
    layers.Dense(16, activation='relu', input_shape=(4,)),
    layers.Dense(8, activation='relu'),
    layers.Dense(1,  activation='linear')
  ])
  optimizer = tf.keras.optimizers.Adam(0.01)

  model.compile(loss='mae',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
  return model
model = build_model()
print(model.summary())

#Training the model

# Display training progress by printing a single dot for each
completed epoch
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

EPOCHS = 100

history = model.fit(
  x_train, y_train[:, 0],
  epochs=EPOCHS, validation_split = 0.2, verbose=0,
  callbacks=[PrintDot()])

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
#model.fit(x_train, y_train, epochs=100,verbose=1, batch_size=1,
validation_split=0.2)
```

```
#Printing the epoch training and predicting results

def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [KM]')
  plt.plot(hist['epoch'], hist['mae'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mae'],
           label = 'Val Error')
  plt.ylim([0,0.3])
  plt.legend()

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Square Error [$KM^2$]')
  plt.plot(hist['epoch'], hist['mse'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mse'],
           label = 'Val Error')
  plt.ylim([0,0.3])
  plt.legend()
  plt.show()

plot_history(history)

#Evaluating the model
print(model.evaluate(x_test, y_test[:, 0]))

y_pred1 = model.predict(x_test)
y_pred = y_pred1[:, 0]

dataset = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test[:, 0],
'Lat_BS': y_test[:, 1],'Lon_BS': y_test[:, 2],'Alt_BS':
y_test[:, 3],'Azimuth': y_test[:, 4],'Node_Lat': y_test[:, 5],
'Node_Lon': y_test[:, 6]}, columns=['y_pred',
'y_test','Lat_BS','Lon_BS','Alt_BS','Azimuth','Node_Lat','Node_Lon'])

plt.plot(y_test[:, 0],'o', color = "gray")
dataset["error"]=""
dataset["New_Lat"]=""
dataset["New_lon"]=""
dataset["Distance_Pred_Test"]=""
plt.plot(y_pred)
```

```
plt.title("Results")
plt.xlabel("Number of values")
plt.ylabel("Distance Km")
plt.show()

l_dt = len(dataset)
for i in range(0,l_dt):
    dataset.iloc[i,8]=abs(float(dataset.iloc[i,0])-float
    (dataset.iloc[i,1]))

fig = plt.figure(figsize =(10, 7))
toplot = dataset.iloc[:, 2].values
plt.boxplot(toplot)
plt.show()

# LOCATION POINTS ESTIMATION
import math
R = 6378.1 #Radius of the Earth

for i in range(0,l_dt):
    #NEW POINT CALCULATION
    bear=float(dataset.iloc[i,5])
    bear_rad=math.radians(bear)
    lat1 = math.radians(float(dataset.iloc[i,2]))
    #lat point converted to radians
    lon1 = math.radians(float(dataset.iloc[i,3]))
    #long point converted to radians
    lat2 = math.asin(math.sin(lat1)*math.cos(float
    (dataset.iloc[i,0])/R) +
           math.cos(lat1)*math.sin(float(dataset.iloc[i,0])/R)
           *math.cos(bear_rad))
    dataset.iloc[i,9]=math.degrees(lat2)
    lon2 = lon1 + math.atan2(math.sin(bear_rad)*math.sin
    (float(dataset.iloc[i,0])/R)*math.cos(lat1),
           math.cos(float(dataset.iloc[i,0])/R)-math.sin
           (lat1)*math.sin(lat2))
    dataset.iloc[i,10]=math.degrees(lon2)
    #DISTANCE BETWEEN PREDICTED AND TEST POINTS
    distance = haversine((dataset.iloc[i,6], dataset.iloc[i,7]),
    (dataset.iloc[i,9], dataset.iloc[i,10]))
    dataset.iloc[i,11]=distance

print('EXPORTING DATABASE')
dataset.to_excel("datasetMLP.xlsx")
print('DB Exported')

print('PREDICTION ANALYSYS')
```

```
print(dataset['Distance_Pred_Test'].mean())
print(dataset['Distance_Pred_Test'].median())
print(dataset['Distance_Pred_Test'].std())
print(dataset['Distance_Pred_Test'].max())
print(dataset['Distance_Pred_Test'].min())

print('RSSI ANALYSYS')

print("Mean",df_MS['rssi'].mean())
print("Standart_DEV",df_MS['rssi'].std())
print("Max_Value",df_MS['rssi'].max())
print('Min_value',df_MS['rssi'].min())

print('DISTANCE ANALYSYS')

print(df_MS['distance_BS'].mean())
print(df_MS['distance_BS'].std())
print(df_MS['distance_BS'].max())
print(df_MS['distance_BS'].min())
```