Politecnico di Torino

Politecnico di Torino

Department of Computer Engineering

Master's Degree in Data Science and Engineering

Machine learning and deep learning models to discern indoor from outdoor environments based on data recorded by a tri-axial digital magnetic sensor

Supervisors:

Prof. Andrea Cereatti

Dr. Stefano Bertuletti

Candidate:

Vincenzo Marcianò

Matr: 282004

A mia mamma,
alla sua forza,
al suo sorriso,
alla sua genuinità.
Come promesso.

"There's nobody like my mom.
There's no place like my home,
since I was born."

Contents

List of Figures

List of Tables

Chapter 1

Abstract

Wearable devices (e.g., smartphones, smartwatches, etc.) are increasingly used for position estimation of users both in indoor environments to provide personalized contents (e.g., malls, museums, crowded venues, etc.) and outdoor environments to provide additional information about their way of moving. Existing mainly solutions exploit the use of GPS which can provide very accurate location information in outdoor environments, while in indoor environments due to a variety of physical barriers that can attenuate the GPS signal (e.g., walls, floors) the quality of the provided information can be very poor. Other approaches which consist in the use of Wi-Fi, Bluetooth, barometers, and light sensors have been investigated and presented in the literature. Despite these approaches can provide very accurate indoor location information, their use is limited because they require the presence of beacons (i.e., Wi-Fi, Bluetooth) and suitable mapping surveys of the interested areas (i.e., barometer, light sensors) which is not feasible for outdoor environments. A technology that's been around for an extremely long time which however, in this thesis, has an innovative and very interesting application is the magnetometer. Thanks to its intrinsic properties, a magnetometer can measure variation of the Earth's magnetic field strength result of the disturbances due to the presence of ferromagnetic materials which are easier to find in indoor environments (e.g., ferromagnetic mate-

rials, electric power lines, etc.) with respect to the outdoor ones. In addition, the advantages of the use of a magnetometer are that i) it is already integrated in most wearable devices; ii) it is less-power consuming with respect to the abovementioned technologies; and, finally, iii) it does not require the use of additional infrastructure (i.e., beacons) and/or areas mapping. Therefore, the aim of this thesis is to apply machine learning techniques to discern indoor from outdoor environments by looking at the local magnetic field strength variations recorded by a magnetometer during activities of daily living of 20 subjects.

Chapter 2

Introduction

The goal of this thesis is to achieve a simple, yet efficient solution to understand whenever a subject is in an indoor environment or not. In these cases gait analysis is the usual way to explore the mobility behaviours: it is the methodical examination of animal locomotion, in particular human motion, through the use of observers' eyes and minds in conjunction with equipment that records movements, mechanics, and muscle activity. Gait analysis is used to diagnose and treat people whose walking ability is affected by diseases. The study includes both quantification (the introduction and analysis of measurable gait parameters) and interpretation, or deducing various information about the animal (health, age, size, weight, speed, etc.) from its gait pattern. It is also frequently used in sports biomechanics to help athletes run more efficiently and to identify posture-related or movement-related problems in people with injuries.

Several attempt have been made during these years for artificial intelligence (AI) applications in walking context: human activity recognition is one of the main example. These applications moved the attention of the researchers in discovering new advanced tools capable of predicting users' behaviours or routines in their own mobility environment.

Moreover, the advent and subsequent distribution of GPS to a broad public

in recent years has profoundly altered the popular understanding of location and navigation. In practice, no universal system exists that would give accurate and economical positioning solutions in every imaginable type of setting because most contemporary location technologies, such as GPS, rely on radio-frequency transmissions, they are quite vulnerable. GPS, as an example, is insufficiently available, rather inaccurate, and worthless for indoor applications in places such as restaurants, parking spaces, subways, and forests.

Meanwhile, the development of MEMS (micro electro-mechanical systems) has enabled them to reach mass markets. Inertial sensors such as accelerometers and gyroscopes, which were originally devoted to high precision navigation tasks aboard aircrafts or submarines, are now included in many electronic devices such as cellphones, smartphones, digital cameras, or game consoles. Although their cost is quite affordable in terms of manufacturing, their accuracy however is vastly below the standards of conventional inertial navigation techniques. Costly navigation grade sensors can be used to generate position estimations but, in the long run, drifts are unavoidable.

Estimating the location is just not possible with low-cost sensors. In reality, the drifts are too strong. Nonetheless, the attitude may be (at least largely) rebuilt: other sensors must be added to lessen or perhaps eliminate navigation mistakes. The majority of these extra sensors require the installation of infrastructure to function. WiFi base stations, visual signs, and magnetic gateways are a few examples. Deploying with high precision and maintaining this infrastructure may be exceedingly costly, and the resulting set-up time might be prohibitive in many types of applications.

In this study, MIMUs (magneto inertial measurement units) are exploited, thanks to their physical compositions and interesting features. MIMU is a device that can measure and report specific gravity and angular rate of an object to which it is attached. Briefly summarizing, a MIMU typically consists of:

- Gyroscope: it measures the angular velocity in the 3D-space
- Accelerometer: it measures the linear acceleration in the 3D-space
- Magnetometer: it measures the local magnetic field in the 3D-space

MIMUs are available in several performance grades. Chapter 3 will give an overview about these sensors, in particular the INDIP system, displaying the features they can bring to the overall study. Moreover, the presented work relies its focus specifically on the magnetometer physical properties: trivially, outdoor environments are less noisy in terms of ferromagnetic signals. For instance, let's consider a subway environment versus a wide-open park: the former will surely present a more hectic behaviour with respect to the latter. Although this may be seem trivial, the signals must be preprocessed to be further investigated by the models.

In Chapter 4 a brief summary of machine learning techniques is proposed. Moreover, each section describes the background, methods and reason behind the model selection. A comparison between machine learning and deep learning approaches is carried, highlighting pros and cons behind each of them for the application sake.

Chapter 5 deals with the data acquisition protocol. Signals gathering was exploited by means of an automatic tracking mobile application (i.e., Aeqora app), and a manually-driven approach, labelling by hand the timestamps captured by taking a photo of the environment everytime a subject entered or not from different locations.

Chapter 6 will explain the experimental protocol and setup for data acquisition, along with the right way to preprocess the gathered data. For some of the suggested models, a feature extraction method must be followed: time series data can display some interesting information if projected in time-frequency domain (e.g. RMS, dominant frequency), and for this reason can be useful to exploit those properties.

Chapter 7 will provide an overall evaluation about the performance of the aforementioned models. Accuracy, f1-score, and AUROC are computed to measure the correctness of the models in identifying the right rate of indoor (1) or outdoor (0)

environments.

Chapter 8 will conclude the whole study by proposing some suggestions for future directions and an overall summary of the experimental work.

Chapter 3

Digital magnetic sensors

Magnetic sensors are the sensors that convert the change magnitude of a magnetic field into an electrical signal. The Earth's magnetic field or the magnetic field referred to as magnetism, is a well-known but invisible phenomenon. Magnetic sensors convert invisible magnetic fields into electrical signals and help to have visible effects. They were designed decades ago as a sensor using the electromagnetic induction effect and has expanded into applications in the magneto-resistance effect, galvano-magnetic effect, the Josephson effect, and other physical phenomena.

3.1 General properties

Magnetic sensors are the solid state devices used in detecting magnetic signals and converting them into electrical signals. The converted signals are processed by electrical circuits. These sensors became popular and used in several applications because of their easy operation, and tolerance to high vibrations, water, and dust.

Moreover, they are devices in which the output switches toggles between the ON and OFF states as an effect of the presence of an external magnetic field. Devices of this type, based on the physical principle of the Hall effect, are widely used as proximity, positioning, speed, and current detection sensors. Unlike a mechanical switch, they are a long-lasting solution as they are free from mechanical wear and

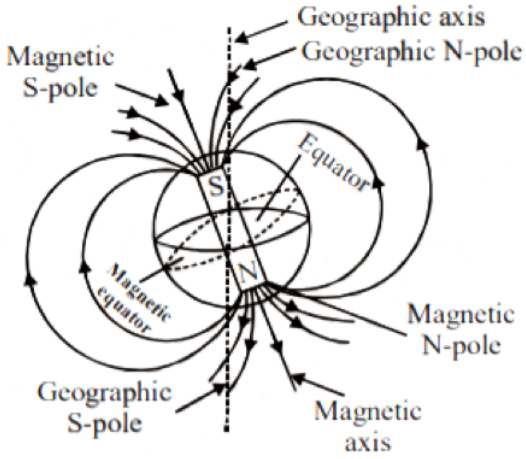


Figure 3.1: Visual representation of the earth magnetic field

can operate even in particularly critical environmental conditions. Digital magnetic sensors are becoming more and more widespread, especially in the automotive and consumer electronics sectors, thanks to features such as contactless operation, lack of maintenance, robustness, and immunity to vibrations, dust, and liquids.

In the automotive sector, for example, these sensors are used to detect position, distance, and speed. Inside the engine they are used to identify the position of the crankshaft, in the passenger compartment they are used to detect the position of the seats and seat belts (basic information for operating the air-bag control system), and on the wheels, they detect the speed of rotation, needed by the ABS.

The heart of each magnetic sensor is represented by the Hall element, whose output voltage (also called Hall voltage) is directly proportional to the intensity of the magnetic field that passes through the semiconductor material. Since this voltage is very low, of the order of a few microvolts, it is necessary to include in the design of other components such as operational amplifiers, voltage comparators, voltage regulators, and output drivers.

3.2 MIMUs for gait assessment

Magnetic field sensors are an integral part of many industrial and biomedical applications, and their utilization continues to grow at a high rate. The development is driven both by new use cases and demand like internet of things as well as by new technologies and capabilities like flexible and stretchable devices. Magnetic field sensors exploit different physical principles for their operation, resulting in different specifications with respect to sensitivity, linearity, field range, power consumption, costs etc.

One of the main scenario in which MIMUs result in being the best option in term of low-cost, low-power and reliability, is the *gait analysis*, i.e. the study relying on the pattern of movement of the limbs of human subjects during locomotion. The importance behind this analysis is determined by the relationship between the walking quality and the individual's health status: an established batch of clinical studies (such as [2, 3]) confirms that the degree of severity of certain disease (such as Parkinson's) will reflect its behaviour on the gait cycle, and vice-versa.

MIMU-based approaches have seen a constant improvement during these years in order to develop and deploy the best optimized system to clinically evaluate temporal features for real-world applications. However one of the main challenges is to collect out-of-laboratory samples to exploit as much as possible the signals' information received by sensors in real-life activities. Moreover, the research of the right MIMU to use may be hard to achieve, and a valuable trade-off between performances and consumption is imperative.

3.2.1 General overview of the INDIP system

In this study, the experiments will rely on a multi-sensor system called INDIP (INertial module with DIstance sensors and Pressure insoles) for real-world walking assessment [4, 5, 6]. The main unit of the INDIP system is a high-performance

MIMU, allowing the user to exploit several sensors in different positions of the human body. The system architecture is depicted as Figure 3.2 suggests.

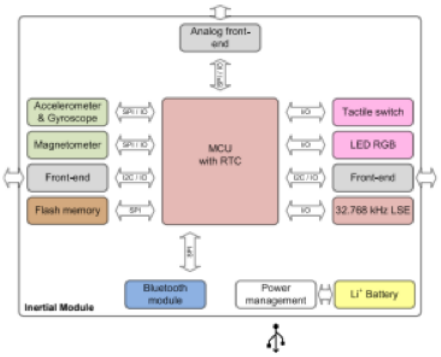


Figure 3.2: Block diagram of the INDIP system

The electronic motherboard was designed to capture both motion sensing and processing thanks to the STM micro controller based on a 32-bit architecture, an up-to-13 hours storage capacity thanks to an Infineon on-board memory, and wire(less) transmission by means of a micro-USB connector plus a Bluetooth low-energy module.

Overall, the system consists of:

- an inertial module, based on a tri-axial gyroscope and accelerometer,
- a magnetic module relying on an ultra-low-power, high performance tri-axial digital magnetic sensor.

Eventually, as intended, the research scope behind this project is solely based on the magnetic data provided by the INDIP. The magnetic signals are recorded at a sampling frequency of 100 Hz.

This system allows the users to work with different sensors to be placed in body areas for their own convenience. The used module is shown in Figure 3.3.

Each subject will indeed be equipped as it follows:

- *Wrist sensor (WR)* linked to the non-dominant wrist

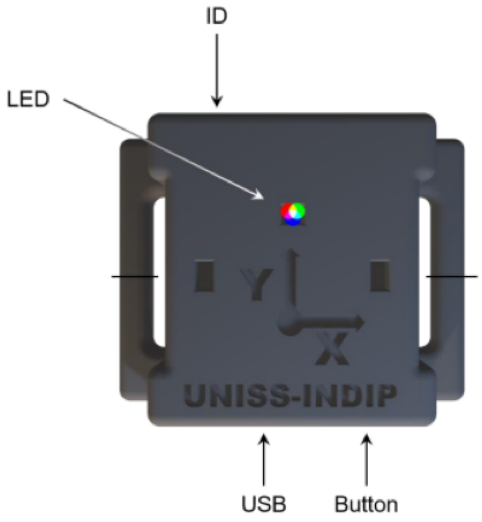


Figure 3.3: One of the provided sensors module of the INDIP system

- *Lumbar sensor (LB)* put on the subject's pelvic area
- *Left foot (LF) Sensor*
- *Right foot (RF) Sensor.*

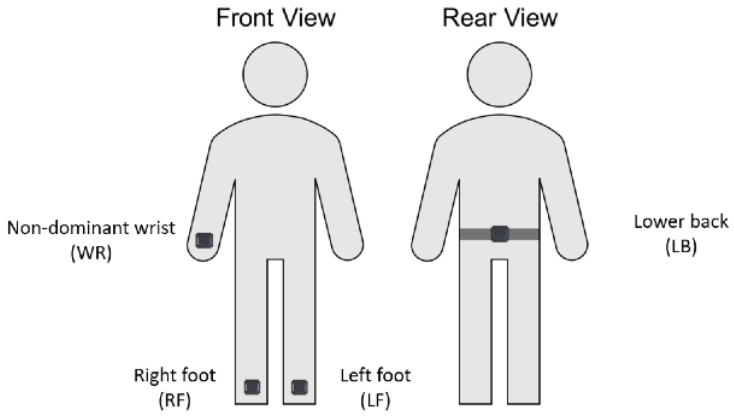


Figure 3.4: INDIP units and relative module positioning for experiments

3.2.2 Technologies for environment detection

A multitude of research works has been done to investigate the best statistic model and the most suitable data to detect whether an individual is inside a specific envi-

ronment or not.

Various suggestions rely on the triangulation of GPS data, WiFi and light sensors: the results achieved can be considered as the state-of-art, however they required a very high energy consumption. Some scientific works indicated that GPS is the most power hungry among the wearable sensors. It is reported to utilize seven-times higher energy as compared to MIMU sensors available. Additionally, a refresh-rate transition time must be taken in consideration when dealing with GPS data.

Magnetometers, on the other side, may be a simple yet efficient solutions to the issues previously pointed out by the other systems. The key idea behind this study is to exploit the magnetic properties of indoor and outdoor environments. For instance, let's investigate two scenarios:

- an underground metropolitan place
- an open-space park in the city centre.

In the first case, the ferro-magnetic signals capture by the magnetometer will be surrounded by a high amount of noise, due to the volatility of the magnetic spectrum of rails, lights, cabins, etc. On the other hand, the green areas don't show a large amount of magnetic signals and for this reason they must be labelled as outdoor environments. At a first glance, this simple yet effective distinction may provide the right approach to come with for the classification, in order to proceed with further methodical and statistical investigations.

Chapter 4

Machine learning and deep learning

In the last years, with the subsequent improvement of computational performances and statistical tools, machine learning (also known as statistical learning) became one of the main approach to solve engineering and mathematical problems that are more or less complex to be solved by hand. Its popularity derives from the beginning of the "big data era", along with data science: billions of data are processed every day, and a wide range of models may be used to mine them, to extract features and to infer predictions at its own convenience.

The biomedical field is one of the science branches that most benefits from the advent of machine learning: tasks such as detection of anomalies in the ECG signal and recognition of human activities have been studied through the use of statistical models to classify whether or not a subject may suffer by a stroke or may stay sit for a long period. These discoveries have rapidly changed the healthcare background, giving to medicine new tools to achieve fast-growing performance for wellness endurance.

Concerning the models used nowadays, machine learning tasks are different and they depend on the results you want to achieve. Note that, on a philosophical perspective, machine learning can also be a mean of understanding the knowledge of the principles that human intelligence is based on to infer decisional choices.

At this purpose, "learning" can be seen as the ability to achieve the task itself: for instance, if we want a robot to jump, then jumping is the task, and one can program it to learn to do it. In this setup, machine learning tasks are usually implemented to work with *datasets*, in which each instance is called *example*. These examples are a collection of *features*, i.e. the quantitative and qualitative descriptors of each sample. These feature are more or less explicit characteristic of each measured object or event that may be processed in the machine learning pipeline. Formally:

$$n = \text{number of samples}$$

$$k = \text{number of features}$$

$$\mathbf{R}^n = \text{samples' space}$$

$$x \in \mathbf{R}^n = \text{single example, vector of length } k$$

$$x_i, \quad \forall i \in [0, k-1] = i - th \text{ feature in the example space}$$

The goal is to find a function f able to map each *training* input and formalize the task on a target y. In general, the aim is to use this fuction f to model unseen samples (*test set*) and obtaining the predicted $\hat{y}$ for real-world applications. Formally, given a target feature y and a statistical model M:

- fit the model on input x and target y: $M \rightarrow y' = f(x)$
- test the fitted model on unseen data $\rightarrow \hat{y} = f(x_{test})$
- evaluate the model with some pre-defined metrics

This idea is simple, yet efficient. However, one its main big limitations is the very difficult and complex way of working with time series. In addition, in traditional machine learning, the learning process is supervised, and the programmer has to be extremely specific when telling the machine what types of features it should be take into account to assess a certain task, for instance to decide if an image is classified as a dog or not. This is a laborious process called feature extraction, and the computer's

success rate depends entirely upon the programmer's ability to accurately define a feature set for dog. The advantage of deep learning is the program builds the feature set by itself without supervision. Unsupervised learning is not only faster, but it is usually more accurate.

Frank Rosenblatt introduced the artificial perceptron idea in 1957 starting from the original MacCulloch-Pitts (MCP) neuron [7]: a perceptron is an algorithm for supervised learning of binary classifiers that allows neurons to learn and process elements from the training set one at a time (Figure 4.1). The perceptron algorithms learn the weights for the input signals in order to draw a linear decision boundary.

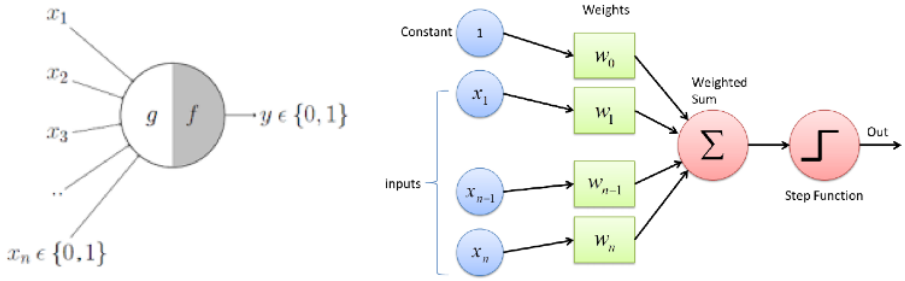


Figure 4.1: Left: First mathematical model of a human neuron by MacCulloch and Pitts. Right: structure of a perceptron

While single layer perceptrons can learn only to distinguish linearly separable patterns, multilayer perceptrons or feedforward neural networks with two or more layers have a greater processing power. From here, let's introduce the idea of deep learning: it is a subset of a larger class of machine learning methods based on artificial neural networks and representation learning that can be categorized as supervised, semi-supervised, or unsupervised learning. It belongs to a sub-family of machine learning techniques because of "deep" network structure to build the model weights.

In order to evaluate how good a deep learning model is, the goal becomes a minimization problem. A loss function (also known as the error/cost/objective function) is the mathematical approach for determining how well the model is learning the samples: trivially, the lower the function value, the lower the error imputed by the

model. For this reason the final task of a neural network is to search for the global minimum in this loss function by updating its gradient. Thus, a *backpropagation* ("backward propagation of errors"), or reverse-mode automatic differentiation, was developed, i.e. a way of gradient computation for gradient descent, and it is a widely used machine learning algorithm to train feed-forward neural networks despite it has many generalizations for other artificial neural networks. It relies on a fine-tuning approach for the weights of a neural network, based on the error rate obtained in the previous iteration, named *epoch*. By fine-tuning the weights, error rates can be reduced and the model's reliability increased by increasing its generalization. As changes are made to the model, the loss function is the best indicator of whether the algorithm is heading in the right direction.

However, the multiple global minima search of the neural models let the network dealing with the exploding gradient phenomena (or the opposite, the vanishing gradient), due to the diverse representation space of the imputed samples. The activation function in a deep neural network specifies how the weighted sum of the input is transformed into an output from a node to another. Since many activation functions are nonlinear, its choice has a large impact on the neural network's capability and performance, and different activation functions may be used in different parts of the model and tailored to the scientist task. Although networks are designed to use the same activation function for all nodes in a layer, it is used within or after the internal processing of each node in the network.

4.1 Related works

4.1.1 Classification

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data. From a modeling perspective, classification requires a training dataset with many examples of inputs

and outputs from which to learn. As mentioned at the beginning of this chapter, a model will use the training dataset and will calculate how to best map examples of input data to specific class labels. As such, the training dataset must be sufficiently representative of the problem and have many examples of each class label.

Class labels are often string values, e.g. “indoor,” “outdoor,” and must be mapped to numeric values before being provided to an algorithm for modeling. This is often referred to as label encoding, where a unique integer is assigned to each class label: in this research study, "Indoor" will be equal to 1 and "Outdoor" will be equal to 0. There are many types of classification algorithms for modeling classification predictive modeling problems. However, there is no good theory on how to map algorithms onto problem types; instead, it is generally recommended that a practitioner use controlled experiments and discover which algorithm and algorithm configuration results in the best performance for a given classification task.

Instead of class labels, some tasks may require the prediction of a probability of class membership for each example. This provides additional uncertainty in the prediction that an application or user can then interpret. A popular diagnostic for evaluating predicted probabilities is the ROC curve (Section 6.5 will discuss the general setup for the chosen metrics). There are four main types of classification tasks that may be encountered. They are:

- Binary classification
- Multi-class classification
- Multi-label classification
- Imbalanced classification

The classification task is then widely used to detect the nature of specific objects, and several studies were conducted to assess the type of environments. Wu et al. [8] propose to use channel state information (CSI) to assess the class of a specific

home place, by designing a CSI-based passive indoor localization system through the development of a naive bayes classifier enhanced with confidence level information. In the field of computer vision, a subset of deep learning based on image/video processing, Morar et al. [9] provided a survey of indoor localization research solutions, proposing a new classification based on the configuration stage (use of known environment data), sensing devices, type of detected elements, and localization methods. Another crucial point in indoor-based samples' classification regards the exploitation of WiFi signals: Abbas et al. [8] presented WiDeep, a deep learning-based indoor localization system that achieves a fine-grained and robust accuracy in the presence of noise, combining a stacked denoising autoencoders deep learning model and a probabilistic framework to handle the noise in the received WiFi signal and capture the complex relationship between the WiFi signals heard by the mobile phone and its location.

Furthermore, some of these related studies (such as [10] and [11]) focused only in a binary classification task for indoor-based prediction. Binary classification relies on the classification between tasks that have only two class labels. Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state. For example "not spam" is the normal state and "spam" is the abnormal state. Another example is "cancer not detected" is the normal state of a task that involves a medical test and "cancer detected" is the abnormal state.

It is common to model a binary classification task with a model that predicts a Bernoulli probability distribution for each example. The Bernoulli distribution is a discrete probability distribution that covers a case where an event will have a binary outcome as either a 0 or 1. For classification, this means that the model predicts a probability of an example belonging to the abnormal state.

Some algorithms are specifically designed for binary classification and do not natively support more than two classes; examples include logistic regression, decision trees and boosting machines, as discussed in Section 4.2.

4.1.2 Time-series

Dealing with signals means dealing with time series data. Effective studies were conducted during the last years to assess how good a machine (deep) learning model can be while handling time series for certain predictive tasks, such as classification or forecasting.

A first strong example comes from the work by Bui et al. work [12]: the proposed scheme combined information from a certain number of GPS satellites, using the GPS sensor interested in a mobile device in which time series data are collected, preprocessed, and classified as indoor or outdoor environment using a machine learning model that is optimized for the best performance, obtaining between 96% and 98% of accuracy. GPS signals are also widely used from NASA laboratories to classify terrain moisture: Grant and his colleagues [13] exploit GPS-derived classification features to identify visible terrain or landcover classes containing a surface/soil moisture component, and use these signals to provide information about the surface that is not obtainable using visible wavelengths alone.

Regarding the biomedical field, and more in particular the gait analysis, Mannini et al. [14] provide an exhaustive approach in using machine learning to run a classification experiment by extracting features from group-specific hidden Markov models (HMMs) and signal information in time and frequency domain, to then validate the model using a support vector machines classifier (SVM). Other works in gait time series data are presented by [15] and [16], in which a comprehensive review of the most up-to-date machine learning techniques are reported.

However, magnetic-field-based detection of indoor and outdoor environments is poorly documented, thus a few of research papers are available in the literature. The current state-of-the-art is represented by [1]: in this work, the authors develop a model, named "MagIO", able to discern indoor from outdoor environments by extracting statistically relevant features in the time domain (e.g. root mean square)

from magnetic field signals. Nonetheless, the experimental trials and the recording methods are different from the ones investigated in this thesis. The experiments last 2 seconds each, sampled at 10 Hz, and they are recorded directly with a smartphone. Moreover, there isn't high diversity in the environments explored: data were collected at Yeungnam University, Gyeongsan campus, a shopping mall and an underground subway station in the Republic of Korea. For this reason, a first naive counter proposal made by this thesis project is to evaluate magnetic data from different nations, and different free-living activities by the subjects that are not based only into specific places.

4.2 Machine learning models

A machine learning model is a program that can find patterns or make decisions from a previously unseen dataset. For example, in natural language processing, machine learning models can parse and correctly recognize the intent behind previously unheard sentences or combinations of words. In image recognition, a machine learning model can be taught to recognize objects - such as cars or dogs. A machine learning model can perform such tasks by having it 'trained' with a large dataset. During training, the machine learning algorithm is optimized to find certain patterns or outputs from the dataset, depending on the task. The output of this process - often a computer program with specific rules and data structures - is called a machine learning model.

A machine learning algorithm is a mathematical method to find patterns in a set of data. Machine learning algorithms are often drawn from statistics, calculus, and linear algebra. Some popular examples of machine learning algorithms include linear regression, decision trees, random forest, and (extreme) gradient boosting machine (GBM).

The process of running a machine learning algorithm on a dataset (called training

data) and optimizing the algorithm to find certain patterns or outputs is called model training. The resulting function with rules and data structures is called the trained machine learning model.

In general, most machine learning techniques can be classified into supervised learning, unsupervised learning, and reinforcement learning. In supervised machine learning, the algorithm is provided an input dataset, and is rewarded or optimized to meet a set of specific outputs. For example, supervised machine learning is widely deployed in image recognition. Supervised machine learning is also used in predicting demographics such as population growth or health metrics, utilizing a technique called regression. In unsupervised machine learning, the algorithm is provided an input dataset, but not rewarded or optimized to specific outputs, and instead trained to group objects by common characteristics. For example, recommendation engines on online stores rely on unsupervised machine learning, specifically a technique called clustering. In reinforcement learning, the algorithm train itself using many trial and error experiments. Reinforcement learning is the algorithm interacting continually with the environment, rather than relying on training data. One of the most popular examples of reinforcement learning is autonomous driving.

There are many machine learning models, and almost all of them are based on certain machine learning algorithms. Popular classification and regression algorithms fall under supervised machine learning, and clustering algorithms are generally deployed in unsupervised machine learning scenarios.

The proposed research study will evaluate three supervised machine learning models (logistic regression, random forest, GBM), as well as a post-processing technique called principal component analysis discussed in Section 8.2.2, to compare the various results.

4.2.1 Logistic regression

Linear regression models are used to identify the relationship between a continuous dependent variable and one or more independent variables. When there is only one independent variable and one dependent variable, it is known as simple linear regression, but as the number of independent variables increases, it is referred to as multiple linear regression. For each type of linear regression, it seeks to plot a line of best fit through a set of data points, which is typically calculated using the least squares method.

These predictions are not sensible, since of course the true target probability must fall between 0 and 1. Any time a straight line is fit to a binary response that is coded as 0 or 1, in principle one can always predict $p(X) < 0$ for some values of X and $p(X) > 1$ for others (unless the range of X is limited). To avoid this problem, $p(X)$ must be modelled using a function that gives outputs between 0 and 1 for all values of X. Given the linear regression formula:

$$p(X) = \beta_0 + \beta_1 X$$

In logistic regression, the logistic function is the following:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Figure 4.2 gives a mathematical view to the expressed function: While both models are used in regression analysis to make predictions about future outcomes, linear regression is typically easier to understand. Linear regression also does not require large sample size as logistic regression needs (it requires an adequate sample to represent values across all the response categories). Without a larger and representative sample, the model may not have sufficient statistical power to detect a significant effect.

Unlike a generative algorithm, such as naïve bayes, it cannot, as the name im-

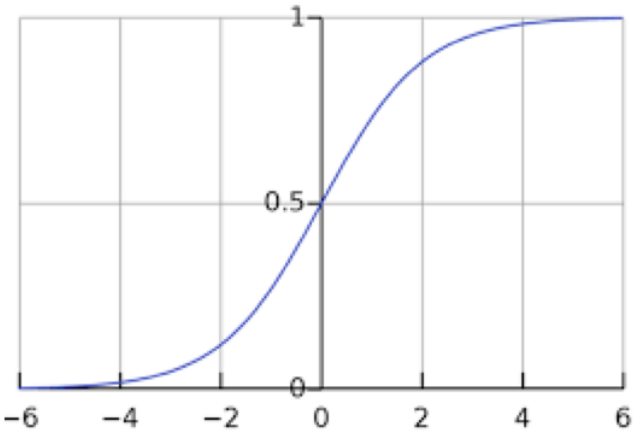


Figure 4.2: Logistic regression domain

plies, generate information. Logistic regression can also be prone to overfitting, particularly when there is a high number of predictor variables within the model. Regularization is typically used to penalize parameters large coefficients when the model suffers from high dimensionality. That's why this approach could benefit from using a dimensionality reduction technique such as PCA.

Given its simple structure and easy-to-use implementation, logistic regression will be used as a first baseline model.

4.2.2 Random forest

The second proposed model is the random forest classifier. A random forest is a statistical model consisting of an ensemble of trees (a collection of decision trees) where each tree is constructed by applying an algorithm A on the training set X and an additional random vector γ sampled *iid* from some distribution. This is followed by the **majority vote** over the predictions of the group of trees: it simply creates a *Dividi et Impera* approach that brings fairly good results (Figure 4.3). Moreover, this method relies on the bootstrap technique: bootstrap is a widely applicable and extremely powerful statistical tool that can be used to quantify the uncertainty associated to a given estimator or statistical learning method [17]. The training algorithm of random forest classifier applies the bootstrap aggregating procedure,

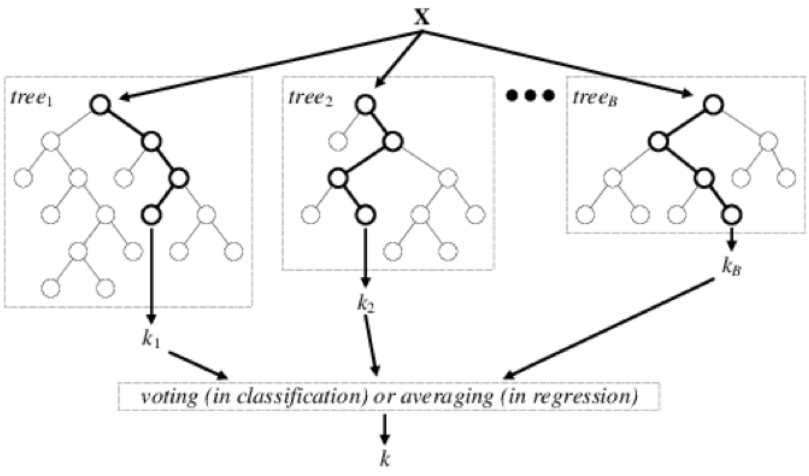


Figure 4.3: Toy illustration of the Random Forest working approach

also called *bagging*, on its trees. Let be:

$$X = x_1, ..., x_n \rightarrow \text{training set}$$

$$Y = y_1, ..., y_n \rightarrow \text{target set}$$

Also, X can be re-written as a function f_θ where θ is the parameter to estimate:

$$X \sim f_\theta$$

Now, bootstrap goal is to repeatedly select (N times) a *random sample with replacement* of the training set X and fit the trees to this updated data. Roughly speaking, the process is the following: So, in the first step the sample X is given and the tree model needs to find $\hat{\theta_1}$; then, the algorithm is fed with a brand-new sample $X^{(1)}$, generated by the previously found parameter. This step is repeated until the $\hat{\theta_N}$ estimate is reached.

Eventually, the unseen samples (like the *test dataset*) can be predicted by two different approaches:

- *majority vote* - classification task (like this research focus);

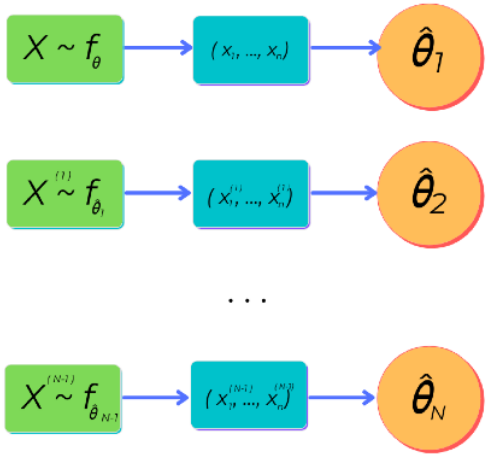


Figure 4.4: Bootstrap mechanism

- *averaging predictions* among the test set - regression task. In this case:

$$\hat{\theta}^* = \frac{\sum^N \hat{\theta}_i}{N}$$

That's where the power of bootstrap comes from: it creates several "tiny" predictors (trees) that *individually* lead to an high noise sensitivity; however, if they are combined together and work, cooperatively, the average of many trees is no more noise-friendly, as long as the trees are not correlated. That's why bootstrap sampling is a de-correlated way to approach each tree family: by showing them different training sets everytime, it decreases the overall variance, still maintaining the same bias.

Also, let's suppose that some features are presented as the *strongest correlated* candidates for the target predictions: this is quite dangerous, because it may lead to a correlated behaviour for each predictor. That's why random forest classifier uses another useful tool, called the *random split*, in generating each sub-tree: this process is crucial to exploit the algorithm because it leads to a gained surplus in accuracy [18].

4.2.3 Extreme gradient boost machine

Unlike many other algorithms, extreme gradient boosting machine (XGB) is an ensemble learning algorithm meaning that it combines the results of many models, called base learners to make a prediction.

Individual decision trees are low-bias high-variance models. They are incredibly good at finding the relationships in any type of training data but struggle to generalize well on unseen data.

The idea of boosting came out of the idea of whether a weak learning model can be modified to become better. A *weak hypothesis* or *weak learner* is defined as one whose performance is at least slightly better than random chance. These ideas built upon Leslie Valiant's work on distribution free or probably approximately correct (PAC) learning which is a framework for investigating the complexity of machine learning problems. Hypothesis boosting was the idea of filtering observations, leaving those observations that the weak learner can handle and focusing on developing new weak learns to handle the remaining difficult observations.

The first realization of boosting that shown great success was the adaptive boosting or AdaBoost for short. The weak learners in AdaBoost are decision trees with a single split, called decision stumps for their shortness.

Extreme gradient boosting, on the other hand, is an efficient open-source implementation of the gradient boosting algorithm: as such, XGB is an algorithm, an open-source project, and a Python library.

It is designed to be both computationally efficient (e.g. fast to execute) and highly effective, perhaps more effective than other open-source implementations. The name XGB refers to the engineering goal to push the limit of computations resources for boosted tree algorithms which is the reason why many machine learning scientists use this model.

In general, gradient boosting involves three elements:

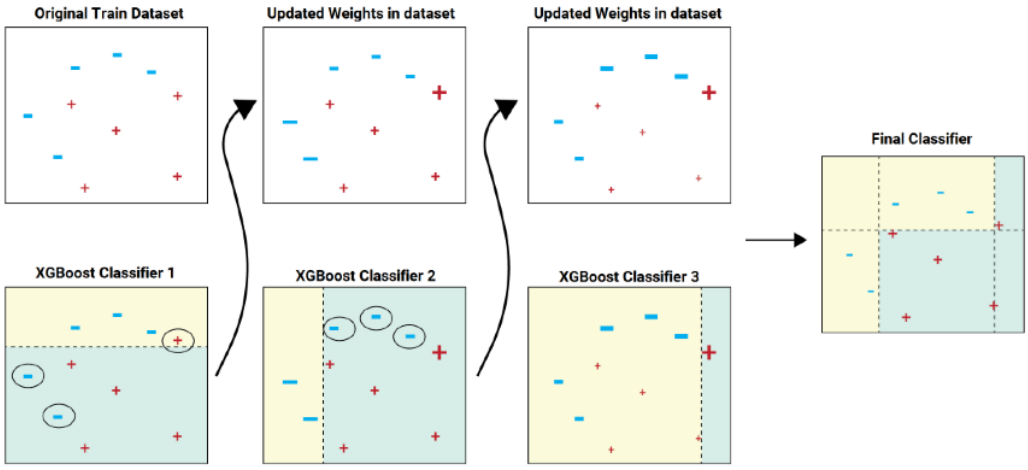


Figure 4.5: Visual representation of how XGB works.

1) A loss function to be optimized. The loss function depends on the type of problem being solved. It must be differentiable but many standard loss functions are supported and anyone can define their own. For example, regression tasks can use a squared error while classification tasks can use logarithmic loss.

2) A weak learner to make predictions. Just like in random forest, XGB uses decision trees as base learners. However, the trees used by XGB are a bit different than the traditional. They are called CART trees (classification and regression trees) and instead of containing a single decision in each "leaf" node, they contain real-value scores of whether an instance belongs to a group. After the tree reaches max depth, the decision can be made by converting the scores into categories using a certain threshold. Trees are constructed in a greedy manner, choosing the best split points based on purity scores like GINI (Section 8.2) or to minimize the loss. It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes. This to keep the learners weak, but still constructed in a greedy manner.

3) An additive model to add weak learners to minimize the loss function. Trees are added one at a time, and existing trees in the model are not changed. A

gradient descent procedure is used to minimize the loss when adding trees. Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error. Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, a tree must be added to the model to reduce the loss (i.e. follow the gradient). Generally this approach is called functional gradient descent or gradient descent with functions.

4.3 Deep learning models

Another valuable choice to achieve the purpose of indoor-outdoor classification is using deep learning:

- It requires large amounts of labeled data
- It requires significant computational power (high performing GPUs)

Machine learning is useful when the dataset is small and well-curated, which means that the data is carefully preprocessed. Data preprocessing (as explained in Section 4.2) requires human intervention. It also means that when the dataset is large and complex, machine learning algorithms will fail to extract information, and it will underfit. Generally, machine learning is alternatively called shallow learning because it is very effective for smaller datasets. Deep learning, on the other hand, is extremely powerful when the dataset is large: it can learn any complex patterns from the data and can draw accurate conclusions on its own. In fact, deep learning is so powerful that it can even process unstructured not adequately arranged data. Furthermore, it can also generate new data samples and find anomalies that machine learning algorithms and human eyes can miss.

Deep neural networks have multiple layers of interconnected artificial neurons or nodes that are stacked together. Each of these nodes has a simple mathematical function, usually a linear function that performs extraction and mapping of information. There are three layer types to define a deep neural network: the input layer, hidden layers, and the output layer (as Figure 4.6). Nowadays, research behind deep

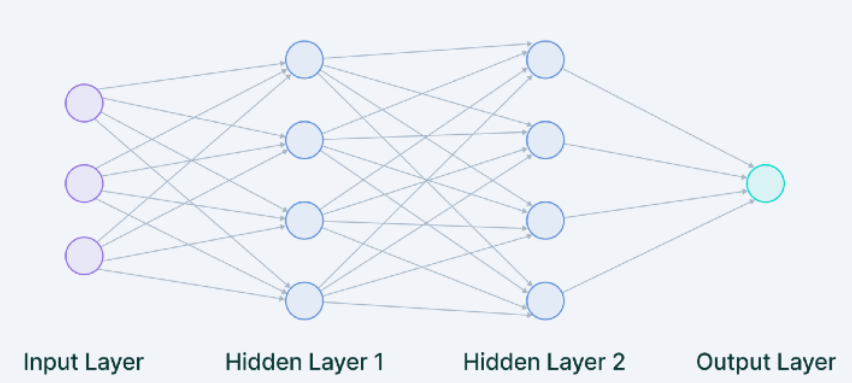


Figure 4.6: Basic neural network design.

learning let several deep learning models available. The main types are:

- Convolutional neural networks (CNNs). CNNs are primarily used for tasks related to computer vision or image processing. CNNs are extremely good in modeling spatial data such as 2D or 3D images and videos. They can extract features and patterns within an image, enabling tasks such as image classification or object detection.

- Recurrent neural networks (RNNs). Section 4.3.1 will focus on these models: RNNs are primarily used to model sequential data, such as text, audio, or any type of data that represents sequence or time. They are often used in tasks related to natural language processing (NLP).

- Generative adversarial networks (GANs). GANs are frameworks that are used for tasks related to unsupervised learning. This type of network essentially learns the structure of the data, and patterns in a way that it can be used to generate new examples similar to those of the original dataset.

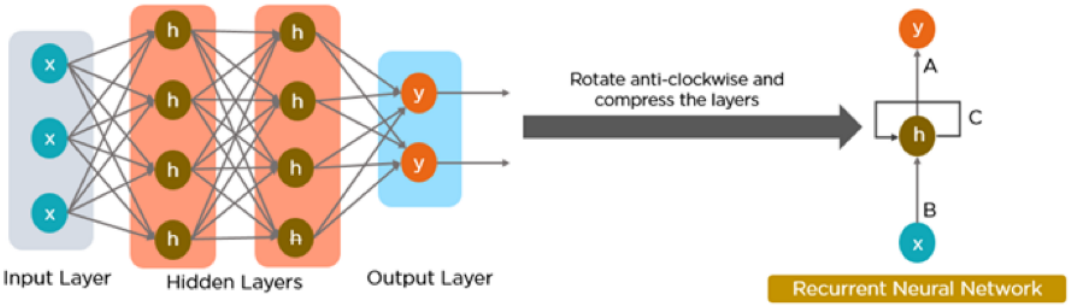


Figure 4.7: Basic RNN structure. The attention is put to the "feedback" loop, in which the direction of the propagation happens anti-clockwise from the hidden state output to its input.

- Transformers. Transformers are the new class of deep learning models that are mostly used for the tasks related to modeling sequential data, like that in NLP. Recently, transformers are also being applied in computer vision tasks.

4.3.1 Recurrent neural networks (RNNs)

A deep learning approach for modelling sequential data (e.g. time series) is recurrent neural networks (RNN). RNNs were the standard suggestion for working with sequential data before the advent of attention models. Specific parameters for each element of the sequence may be required by a deep feed-forward model. It may also be unable to generalize to variable-length sequences.

Recurrent neural networks use the same weights for each element of the sequence, decreasing the number of parameters and allowing the model to generalize to sequences of varying lengths. RNNs generalize to structured data other than sequential data, such as geographical or graphical data, because of its design.

Recurrent neural networks, like many other deep learning techniques, are relatively old. They were first developed in the 1980s, but we didn't appreciate their full potential until lately. The advent of long short-term memory (LSTM) in the 1990s, combined with an increase in computational power and the vast amounts of data that we now have to deal with, has really pushed RNNs to the forefront.

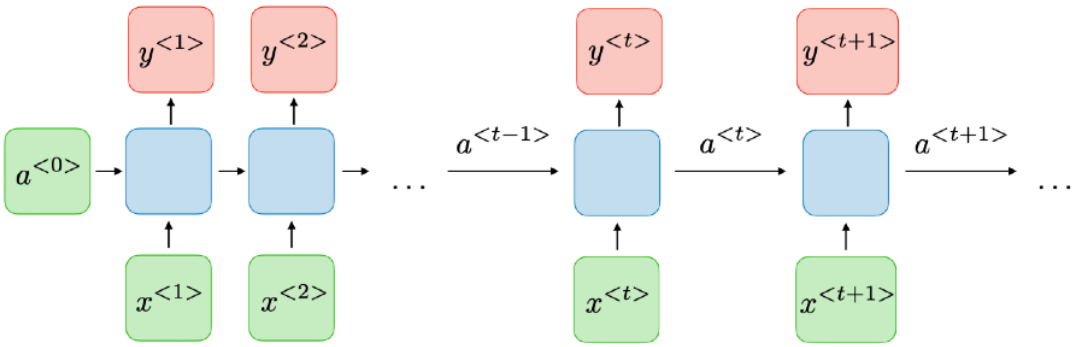


Figure 4.8: A deep dive into RNN internal weight propagation. The input $x^{<i>}$ for each i, processed by the internal weights' matrix a^{i-1} to give the output $y^{<i>}$ and the hidden states $a^{<i>}$ to be further used for the next cell operation.

RNNs, which are derived from feed-forward networks, are similar to human brains in their behaviour. Simply said, recurrent neural networks can anticipate sequential data in a way that other algorithms can't. All inputs and outputs in a standard neural networks are independent from one another, however in some circumstances, such as when predicting the next word of a phrase, the prior words are necessary, and so the previous words must be remembered. As a result, RNN was created, which used a hidden layer to overcome the problem. The most important component of RNN is the hidden state, which remembers specific information about a sequence. RNNs have a memory that stores all information about the calculations. It employs the same settings for each input since it produces the same outcome by performing the same task on all inputs or hidden layers. The following are some examples of RNN architectures used in the research field:

- One to one
- One to many
- Many to one
- Many to many

To train networks in RNNs, backpropagation must be don through time and at each time step (or loop operation) the gradient is calculated and used to update the

weights in the network. Now, if the effect of the previous sequence on the layer is insignificant, then the relative calculated gradient is small. Then if the gradient of the previous layer is smaller then this makes weights to be assigned to the context smaller and this effect is observed when we deal with longer sequences. Due to this network does not learn the effect of earlier inputs and thus causing the short term memory problem. To overcome this problem specialized versions of RNN are created, for instance long short memory unit (LSTM) and gated recurrent unit (GRU).

- The popularity of LSTM is due to the getting mechanism involved with each LSTM cell. In a normal RNN cell, the input at the timestamp and hidden state from the previous time step is passed through the activation layer to obtain a new state. Whereas in LSTM the process is slightly complex, as can be seen in the above architecture at each time it takes input from three different states like the current input state, the short term memory from the previous cell and lastly the long term memory. These cells use the gates to regulate the information to be kept or discarded at loop operation before passing on the long term and short term information to the next cell. We can imagine these gates as filters that remove unwanted selected and irrelevant information. There are a total of three gates that LSTM uses as input gate, forget gate, and output gate.

- The workflow of the gated recurrent unit, in short GRU, is the same as the RNN but the difference is in the operation and gates associated with each GRU unit. To solve the problem faced by standard RNN, GRU incorporates the two gate operating mechanisms called update gate and reset gate.

A comparison between these two models can be found in Table 4.1. Moreover, Figure 4.9 gives more visual insights about LSTM and GRU architectural differences. Eventually only the LSTM cell is used in the proposed final architecture as shown in Section 4.3.2.

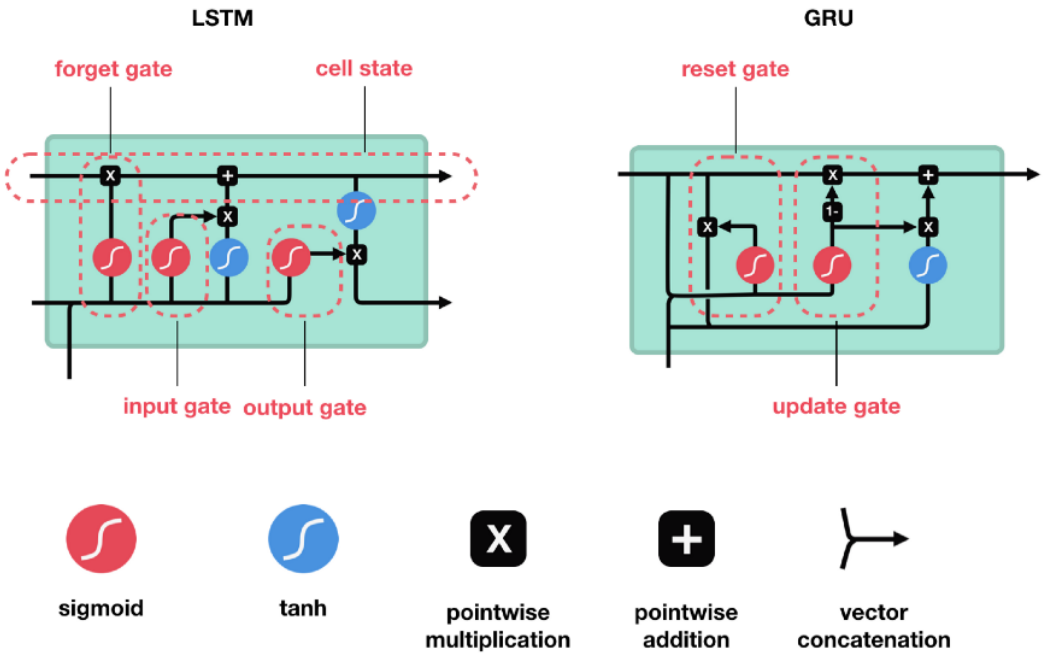


Figure 4.9: Internal comparison of the LSTM vs GRU cell. Note: *tanh* is the hyperbolic tangent operations, while *sigmoid* is another term denoting the Logistic Function

Cell features	LSTM	GRU
# gates	3	2
Internal memory	Yes (output gate)	No
Previous state connection	Coupled by input and target gate	Direct connection with reset gate

Table 4.1: Comparison between LSTM and GRU cell

4.3.2 Proposed architecture

The proposed architecture relies on the use of LSTM units in series, namely *stacked LSTM*. This choice is faster to train and easier to build, yet the concatenation between RNN units may lead to find specific hidden temporal pattern/features.

In between there is a so-called dropout layer: it had been optimised to switch on/off the internal hidden states' neurons according to a properly tuned probability value. This regularizes the network and gives the model a more robust way to train itself, and avoid overfitting (not properly learning the unseen testing samples).

The dense layer is structured as following:

- a linear layer, working as a linear regression, applies a linear transformation to the incoming data: $y = xA^T + b$
- an output layer (a sigmoid, because the task is a binary classification) that squeezes the prediction made by the previous predictor in probabilities, i.e. between 0 and 1

As described in this Section 4.3.1, activation function is needed to deal with gradient-flow related issues. To overcome this, the most trivial solution is using a rectified linear unit (ReLU): it applies a 1:1 identity transformation for positive intermediate outputs, and let the negative values being 0. However, one may find that for very-deep neural networks, a non-negligeable amount of neurons can bring no useful information if set to 0 by ReLU: this well-known problem is called "dying ReLU". For this reason, a gaussian error linear unit (GELU) [19] is used instead of the basic one. In deep learning literature its popularity came after ReLU due to its characteristics that compensate for its drawbacks. Like ReLU, GELU as no upper bound and bounded below. While ReLU is suddenly zero in negative input ranges, GELU is much smoother in this region. It is differentiable in all ranges, and allows to have gradients (although small) in negative range. A visual comparison is given by Figure 4.10. Moreover, the whole architecture may benefit from the usage of GELU aas it brings fairly good results in time-series based experiments and research [20], [21].

Section 6.4 will complete the overview for the proposed models, by adding parametric information both on the stacked RNN architecture and machine learning models. Figure 4.11 will give a visual perspective of the overall model. Section 6.5 will also show all the hidden paramters structure.

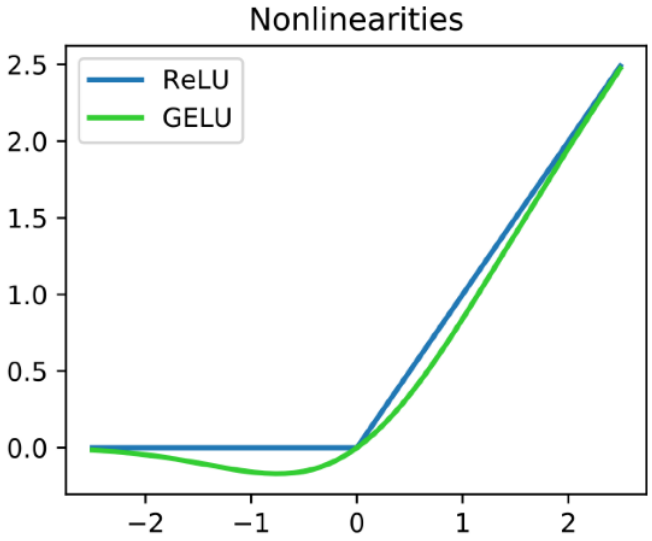


Figure 4.10: Comparison between ReLU and GELU mathematical function. For highly negative and positive values, the two models behave almost the same, while for smaller values GELU apply a gaussian tranformation.

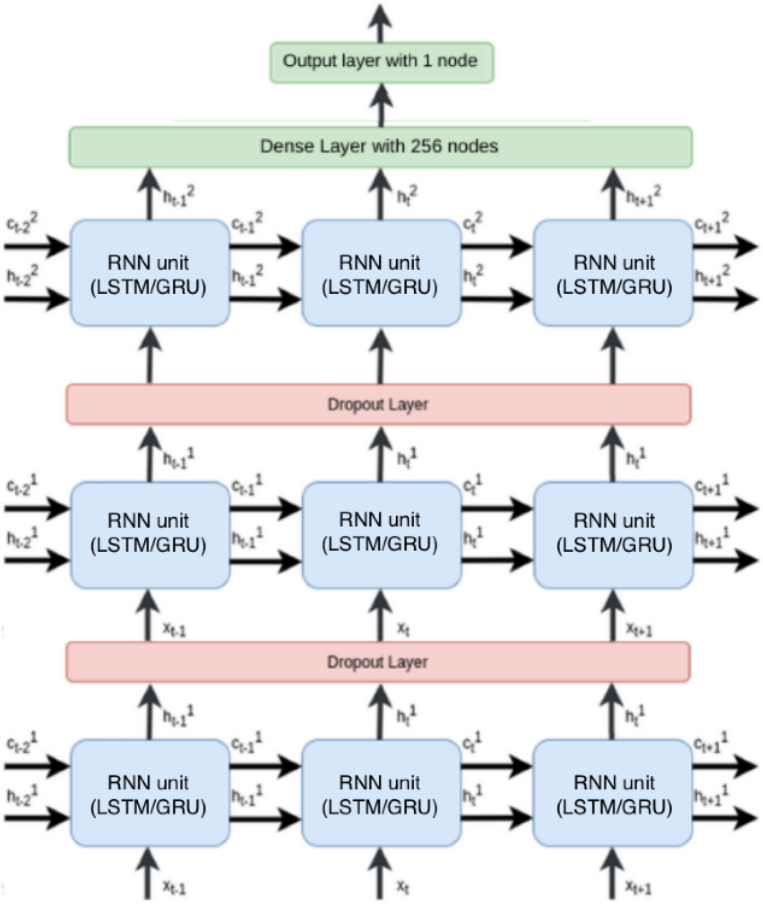


Figure 4.11: Model architecture

Chapter 5

Data acquisition

In this research the study aims to focus in free-living experiments. This mechanism allows more real-world scenario instead of focusing on the idealistic, more artificial way to represent experiments as the in-lab protocol suggests. The beneficial aspect behind this idea is also to gather different and various environments to feed the pipeline with: trivially, more valuable data means more valuable reasoning from the models perspective.

5.1 Experimental setup

Each individual was requested to assess some simple free-living actions. These actions are monitored by equipping the subject with the right positioning of the sensors, and activating them in order to start the recordings.

5.2 Experimental protocol

Each participant will perform free-living and unsupervised activities. However, one of the main constraints is to collect at least 2.5 hours to be consider valid, but there are no requirements with respect to the preference of fully indoor or outdoor samples: balanceness of the final dataset is taken in consideration as a preprocessing

step, in order not to influence the experimental trials.

General tasks that should be completed during the 2.5-hour monitoring (if suitable for the participant) are the following:

- rise from a char and walk to another room
- walk to the kitchen and make a drink
- walk up and down a set of stairs (if possible)
- walk outside (if possibile, for a minimum of 2 minutes)
- if walking outside, walk up and down an inclined path

When the session has been completed, the activities performed during this session are recorded by the operator in the out of lab checklist. Data was recorded in four different countries and different cities as well:

- United Kingdom (Sheffield, Newcastle upon Tyne),
- Italy (Turin),
- Israel (Tel-Aviv),
- Germany (Stuttgart, Kiel)

For the sake of research clarity, the focus must be brought to the different environment(s) visited by each subject, instead of considering the 1:1 relationship "one subject, one environment". However, even if several individuals had been pursuit the activities in different environments during the experiments (been monitored by Aeqora App, more details in Section 5.3), the data protection protocol does not allow the user to know exactly the precise location of every path the participants underwent through. For instance, it can happen that during some active walking path the subject either might stand still around a certain area or going recursively in the same point during time. The app calls these locations as *stay points*. They are of different types, the most frequent are the following:

- Home-And-Work,
- Building,
- Residential

Even though this can give some information about the walking iteration of a certain subject, they are not relevant in terms of added value for the models.

5.3 Data storage

The final step is to gather all the experimental data from the sensors and surrounding environment in one dataframe. Figure 5.1 will display the overall folder architecture to store all the collected data.

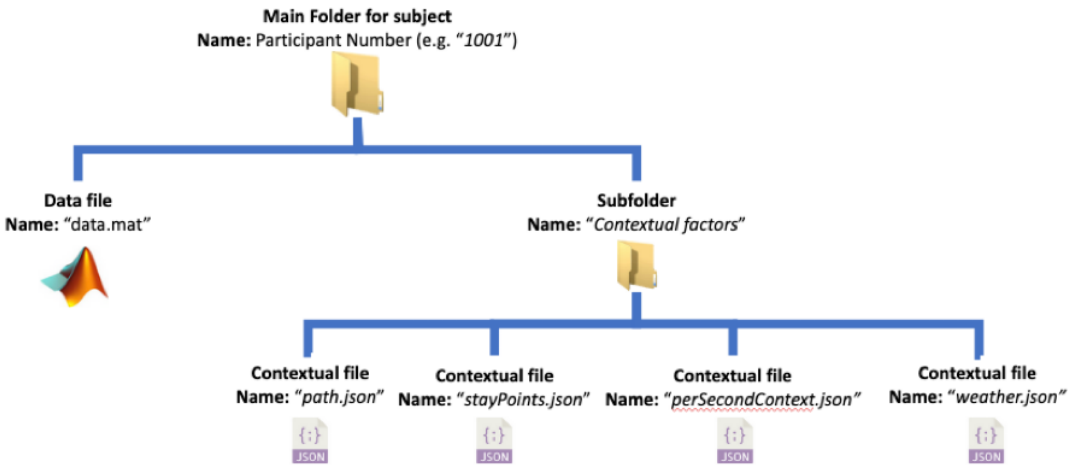


Figure 5.1: Acquisitions' folder skeleton

Data is loaded into the *data.mat* (Matlab) file: its skeleton is a structured schema in which raw triaxial temporal-magnetic data from each sensor is stored, as well as the timestamps and the signals coming from the other embedded modules (unused for our purpose).

On the other hand, the free-living environments are monitored by the usage of a mobile app and a smartphone. The imputed mobile application is called **Aeqora**, developed by the Department of Computer Science, The University of Sheffield, and available both on Play Store and App Store. This tool is extremely useful for automatic labelling by looking at GPS data and activities conducted by the user. In particular, it stores the *indoor probability* of being in an indoor environment (*100*) or not (*0*). These probability values are integers. The smartphone used for this system is the **Samsung s9**, provided with the suggested app pre-installed. The app tracks the smartphone GPS activity, then it iterates on it to retrieve the most valuable points of interest starting from the path of the user. Figure 5.2 shows an internal functioning example.

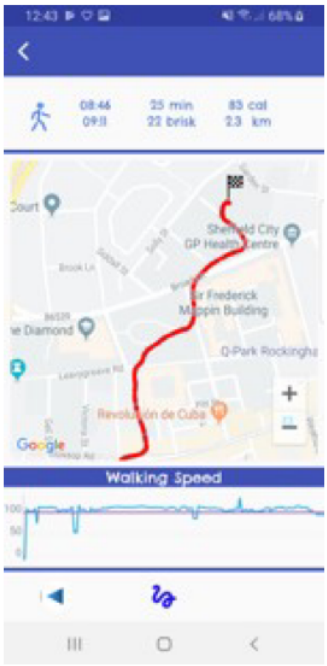


Figure 5.2: Example of Aeqora path by GPS monitoring

Both the path and the stay-points are labelled with an unique ID. The app also stores the weather attributes (wind direction, wind speed, temperature, and precipitation) as well as the elevation percentage. The whole set of data is then collected in a *.json* (JSON) file. The stored informations are indexed with the

corresponding timestamps, sampled at a frequency of 1 Hz. Each instance is then inferred in an array fashion, as the following structure displays:

timestamp : [Indoor probability, staypoint ID, path ID, elevation percentage]

where:

- indoor probability is an integer between 0 and 100
- staypoint ID, an integer positive number identifying the point of interest
- path ID, an integer positive number identifying the path the subject run across
- elevation percentage, an integer value, either positive or negative, indicating the slope of the ground on which the subject is walking

Note that in this thesis the indoor probability and the corresponding timestamp are the only useful retrieved information for the analysis. All the contextual arrays are saved in a file named *per_second_context.json* (5.1)

However, the main drawback behind this system is the cellphone battery and the GPS tracking system that can drop or, worse, let the phone switch off for the low power. In this case, the system automatically infers an indoor probability equal to 50%. These kind of data would be eventually removed because of the binary classification task between indoor and outdoor, and the threshold to round up to 100 or down to 0 is set as 50.

To overcome this issues, another collecting method had been followed for some experiments: after equipping the sensors, the subject is asked to take some pictures through the mobile app **TimestampCamera**, available on Play Store and App-Store: the pictures are taken in order to evaluate if the participant is entering or exiting a specific environment, as well as collect the timestamp saved in the photo itself. After the experimental trial is finished, the operator writes a *.txt file* having the following structure:

1 Start: indoor/outdoor

2 Timestamp: indoor/outdoor (for every taken photo)

3 End: indoor/outdoor

Start and end timestamps are retrieved automatically by the raw signals.

Chapter 6

Data preprocessing

As mentioned in the previous Chapter, each subject is experimentally observed for about 2.5 hours, i.e., $\simeq 1 \cdot 10^4$ seconds of acquisition. However, due to the absence of a stable GPS signal from the raw data, a subset of them experiences a reduced number of available samples. The raw files are then processed iteratively by selecting the INDIP data along with the timestamps.

However, the contextual files and the raw signals are expressed in a different frequency level: the former is sampled with a frequency of 1 Hz, whilst the latter is based on 100 Hz samples, as Table 6.1 shows. The solution to this discrepancy is to inner-joining the per-second contextual data with the INDIP timestamps on the *timestamp* level (Table 6.2). As a final step, a labelling process with 0 as outdoor and 1 as indoor is done, based on the contextual factors file.

timestamps	**feature space**
ts_ 0.00	float value(s)
ts_ 0.99	float value(s)
ts_ 1.00	float value(s)
...	...
ts_ N.xx	float value(s)

timestamps	**indoor probability**
ts_ 0	binary_value_0
ts_ 1	binary_value_1
...	...
ts_ N	binary_value_N

Table 6.1: Tabular version of raw feature data (left) and contextual/indoor-outdoor data (right).

timestamps	**feature space**	indoor probability
ts_ 0.00	float value(s)	binary value (ts_0)
ts_ 0.99	float value(s)	binary value (ts_0)
ts_ 1.00	float value(s)	binary value (ts_1)
...	...	...
ts_ N.xx	float value(s)	binary value (ts_N)

Table 6.2: Inner joined dataframe. Each indoor probability value assigned to each 100 Hz timestamp is the repeated 1 Hz corresponding value.

6.1 Signal handling

The magnetometer was previously introduced as a system able to capture the local magnetic field. For the sake of simplicity this field can be noted as B.

To easily spot every sensor name, the following naming system is adopted:

- Wrist unit $\rightarrow$ *WR*
- Lumbar unit $\rightarrow$ *LB*
- Left foot unit $\rightarrow$ *LF*
- Right foot unit $\rightarrow$ *RF*

Then, every 100 Hz sample S_i is composed by:

$$S_i = B_{LB_x}, B_{LB_y}, B_{LB_z}, B_{LF_x}, B_{LF_y}, B_{LF_z}, B_{RF_x}, B_{RF_y}, B_{RF_z}, B_{WR_x}, B_{WR_y}, B_{WR_z}$$

where x, y, z are the sensor triaxial directions of the magnetic field. Eventually, a total of 12 magnetic signal features for each sample is then extracted.

6.2 Feature engineering

The feature engineering step can be seen as a data manipulation to catch the most valuable and informative features from the feature space. Before any further exploration, must be

6.2.1 Norm computation

A first step of feature analysis and engineering was dedicated to compute the norm of every sensor signal along its axes. Naming k the k-th sensor, then:

$$\hat{B}_i = \sqrt{B_{i_x}^2 + B_{i_y}^2 + B_{i_z}^2}$$

The norm is then added to the previous feature set, with a total of 12 features per sample:

$$S_i = [B_{LB_x}, B_{LB_y}, B_{LB_z}, \hat{B}_{LB},$$

$$B_{LF_x}, B_{LF_y}, B_{LF_z}, \hat{B}_{LF},$$

$$B_{RF_x}, B_{RF_y}, B_{RF_z}, \hat{B}_{RF},$$

$$B_{WR_x}, B_{WR_y}, B_{WR_z}, \hat{B}_{WR}]$$

6.2.2 Window and feature selection

A dedicated window is passed through the entire dataset: this adds to each 1s timestamp the context about its previous window of T samples' features behaviour. This new data can be intended as a **time series** with dimensionality **[N, F, T]**, where N = the total number of datapoints. For the F and T dimensions, different trials had been experimented. In particular, for the signals dimension F:

- F = 16 is used when the *all-sensors* experiment is run, i.e. involving all the sensors
- F = 4 is applied when the *single-sensor* configuration is used. Hence, running the models for WR, LB, LF, RF separately.

Instead for the temporal sequence T three setup had been exploited, to evaluate the feature complexity and signal quality:

- T = 50 (half second),
- T = 100 (one entire second),
- T = 200 (two seconds),

6.2.3 Time-Frequency features

Despite the usage of deep learning models that can find automatically a hidden pattern inside the feature space, for simpler machine learning models this can be less trivial to do. A key idea is to manually extract some set of relevant features, such as temporal-frequency information from the time series input.

The current state-of-art [1] suggested a specific set of features, mostly based on the temporal domain. However, due to the noisy nature of the input signals, an enhanced and more accurate exploration must be done. For this reason, a frequency dedicated feature set is proposed, as Table 6.3 suggests.

Feature domain	**Feature name**
Temporal domain	Root mean square
	Mean
	Median
	Variance
	Mean Absolute Deviation
	Kurtosis
	Skewness
	Percentiles (0.01, 0.10, 0.25, 0.50, 0.75, 0.99)
	Interquartile Range
	Trimmed Mean (10%)
	Sensor correlation
Frequency domain	First Dominant frequency
	Power at the first dominant frequency
	Second Dominant frequency
	Power at the second dominant frequency

Table 6.3: Temporal and Frequency domain features used for machine learning models

6.3 Selection of the participants

Data recordings can be affected by unbalanceness due to participants' activities selection. Since a critical unbalanceness may lead the classifier to a poor knowledge during the training procedure, an empirical yet efficient filtering among all subjects is mandatory.

Firstly, a section of reasonable ratio for the research purpose must be made. Several studies may agree that a maximum threshold of 60/40 for a binary classification task seems adequate for the models.

At this point, a methodical selection had been carried to select the valid subjects to be used as input:

- select only the participants with at least $4.5{\cdot}10^5$ samples, i.e. $\sim$ 75 min of collected data;

- select those subjects that do not have gaps in the GPS recordings, i.e. null data in the contextual array;

- discard for each subject those timestamps that have a different length from the imputed T(= 25,50,100 or 200)

- select only the seconds with indoor probability different from 50%.

Finally, **25 participants** had been collected, for a total of $\sim$ **1.42** $\cdot 10^5$ seconds (or $\sim$ **2360 minutes**) of valid experiments. The participants are divided in the following way: 20 for the training set and 5 for the testing set.

6.4 Model parameters and functions

6.4.1 Hyperparameters tuning (machine learning)

All proposed models have been tuned according to the leave-one-subject-out validation: at each training step, the model is tested on the n-1 subjects and their corresponding environments, while using the one left out to validate the process. Validation results are repeated in Chapter 7. The models need to be validated to gain some insights on their overall behaviour, and to understand whether or not they are going in the right direction of predicting samples.

Another crucial point is the hyperparameters tuning process. Hyperparameters control the over-fitting and under-fitting of the model. Optimal hyperparameters often differ for different datasets, and to get the best one(s) the following steps should be followed:

- The model is evaluated (validated) for each proposed hyperparameter setting
- The hyperparameters that give the best model are selected.

Several approaches are available nowadays for this optimization technique.

- *Grid search* picks out a grid of hyperparameter values and evaluates all of them. Guesswork is necessary to specify the min and max values for each hyperparameter.
- *Random search* randomly chooses a sample of points on the hyperparameter grid. It is more efficient than grid search.
- *Smart hyperparameter tuning* picks a few hyperparameter settings, evaluates the validation matrices, adjusts the hyperparameters, and re-evaluates the validation matrices.

It's worth mentioning that grid and random search are completely uninformed by past evaluations, and as a result, often spend a significant amount of time evaluating "bad" hyperparameters. They do not pay attention to past results, keeping searching across the entire range of the number of estimators even though it's clear the optimal answer (probably) lies in a small region of the hyperparameters' space. Hence, another technique is adopted: the Bayesan optimization.

Bayesian approaches, contrary to random or grid search, keep track of past evaluation results used to infer a probabilistic model, by mapping hyperparameters into a probability value of the metric score, based on the following objective function:

$$\mathbf{P}(score|hyperparameter)$$

In the literature, this model is called a *surrogate* for the objective function and is represented as $p(y|x)$. The surrogate is much easier to optimize than the objective function. The steps are:

1 Build a surrogate probability model of the objective function

2 Find the hyperparameters that perform best on the surrogate

3 Apply these hyperparameters to the true objective function

4 Update the surrogate model incorporating the new results

5 Repeat steps 2–4 until max iterations or time is reached

The aim of Bayesian reasoning is to become "less wrong" by continually updating the surrogate probability model after each evaluation of the objective function. At a high-level, Bayesian optimization methods are efficient because they choose the next hyperparameters in an informed manner. The basic idea is: spend a little more time selecting the next hyperparameters in order to make fewer calls to the objective function. In practice, the time spent selecting the next hyperparameters is inconsequential compared to the time spent in the objective function. By evaluating

hyperparameters that appear more promising from past results, Bayesian methods can find better model settings than random search in fewer iterations.

After evaluating this method on every model, the resulting hyperparameters for random forest and XGB are collected in Table 6.4:

Model	Hyperparameters	Results
Random forest	n_estimators = [100,200,500] max_features = ['auto', 0.5, 0.8] max_depth = [2,3,4,5,9,10]	n_estimators = 200, max_features = 0.5, max_depth = 4
XGB	subsample = [0.3, 0.5, 0.8] max_depth = [2,3,4,5, 9,10] eta = [0.1, 0.3, 0.5] min_child_weight = [1, 5, 10]	subsample = 0.5 max_depth = 2 eta = 0.1 min_child_weight = 1

Table 6.4: Hyperparameters tuning for machine learning models

6.4.2 Optimization and loss functions (deep learning)

A different approach is undertaken in deep learning models tuning. Due to machine computing limitations (CPU and GPU) the selection of the optimal parameters has to be done manually, by monitoring step by step the updating training loss and validation loss: generally, a deep learning model has a good generalization in predicting unseen samples if the validation loss doesn't increase during the training, otherwise over-fitting occurs, letting the model learning only samples similar to the ones provided by the training dataset.

Firstly, the binary cross-entropy (BCE) loss function was chosen for the study purpose, since it relies on a binary classification task. Once the network computes the predicted probability for a certain sample, then it has to include the corrected probability: this is the probability of having a particular observation belonging to its original class. For instance, let's consider the toy example in Table 6.5:

The observation ID8 is from class 0, and its predicted probability (i.e, the chances that ID8 belongs to class 1) is 0.56 whereas, the corrected probability (i.e., the chances that ID8 belongs to class 0) is 0.44 (1-predicted probability). In the same

ID	Class	Predicted prob	Actual prob	Log value
ID8	0	0.56	0.44	-0.3565473235

Table 6.5: Simple working process of the Binary Cross Entropy

way, corrected probabilities for all the instances will be calculated. Then the log value for each of the corrected probabilities is computed. The reason behind the use of the log value is that it offers less penalty for small differences between predicted and corrected probability. When that difference is large enough the penalty will be higher. To compensate for the negative value of the log function (the results are in the range of 0-1 so the log of this quantity will be negative) the formula simply let the outcome of the function being negative. Further, instead of calculating corrected probabilities, one can calculate the log loss can be computed using the formula given below. With some algebraic manipulations it becomes:

$$log\ loss = \frac{1}{N}\sum^{N} -(y_i \log p_i + 1 - y_i \log (1 - p_i)$$

The optimizer used is called adaptive moment estimation (Adam) [22]. The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

- adaptive gradient algorithm (AdaGrad) maintains a learning rate passed as parameter that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

- root mean square propagation (RMSProp) maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm performs well on online and non-stationary problems (e.g. noisy).

Adam gets the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

The final deep learning parameters, both for training and model architecture, are displayed in Table 6.6:

Section	**Parameter/Function**	**Value**
Training	Loss function	BCE
	Optimizer	Adam
	Batch size	128
	Epochs	100
	learning rate	0.001
Model	Dropout	0.75
	RNN hidden nodes	128
	Number of stacked layers	3
	Linear nodes	256

Table 6.6: Deep learning model's parameters, split in training and model.

It's worth noticing that the choice behind the number of epochs and learning rate was selected heuristically. In fact, in the conducted experiments, the model tended to reach the loss minimal plateau at around 60 epochs, but the duration was extended to explore more local minima if needed. Moreover, in order to have a good balance between GPU usage and loss research, early stopping was implemented. The early stopping is a technique that helps the model to not overfit, thus "learning more than needed".It monitors the difference between training loss and validation loss (called *delta*), and start a count whenever this delta is greater than the provided value. Eventually, if this counter reaches a critical value, namely *patience*, given by the user, the early stopping module raises a flag that let the model exiting from the training process. For this setup, the patience was set to 5 and the delta set to 0.3.

6.5 Metrics

Metrics are used both for validation and testing data. They are also shared among all the proposed models, since their usage is the same for every instance.

At the basis of the metrics concept there is the *confusion matrix*. In machine learning, the *confusion matrix* (Figure 6.1) consists of a table that describes the performances of the used algorithms. This is extremely useful in order to evaluate how "good" the model is behaving by looking at the final outcomes. In this study, the matrix rows represent the instances in an actual class while each column represents the instances in a predicted class.

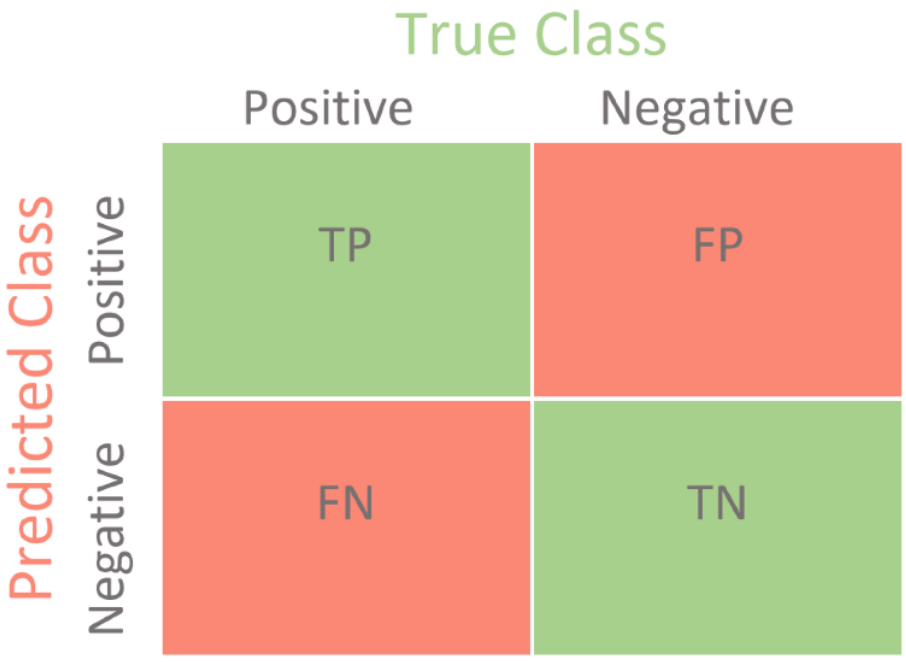


Figure 6.1: General overview of the confusion matrix to assess model quality.

Where:

- **TP** are the true positives
- **TN** are the true negatives

- **FP** are the false positives
- **FN** are the false negatives

From here, several metrics have been deducted to understand whether or not the model is going in the right direction in terms of generalization. In particular:

- Accuracy is the proportion of the total number of predictions that were correct.
- Positive predictive value or Precision is the proportion of positive cases that were correctly identified.
- Negative predictive value is the proportion of negative cases that were correctly identified.
- Sensitivity or recall is the proportion of actual positive cases which are correctly identified.
- Specificity is the proportion of actual negative cases which are correctly identified.

From this set, accuracy will be explicitly used to evaluate the results. The given formula is the following:

$$\text{Acc} = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

Furthermore, F1-score can be used as a middle point between precision and recall. F1-Score is the harmonic mean of precision and recall values for a classification problem. The formula for F1-Score is as follows:

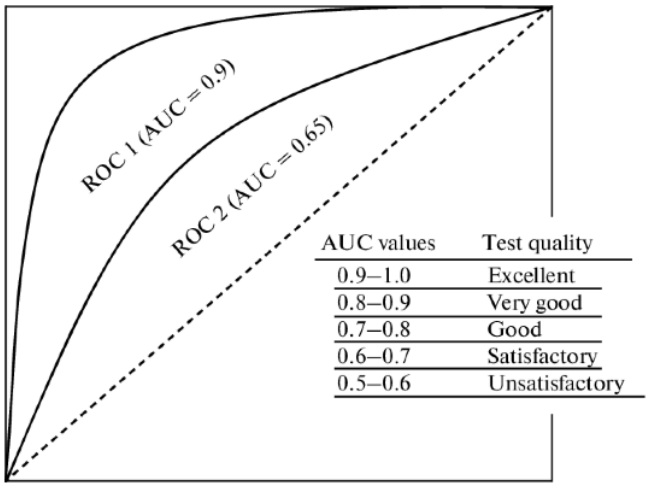
$$\text{F1-score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Let's assume having a binary classification model with the following results:

$$\text{Precision} = 0, \text{Recall} = 1$$

Here the arithmetic mean is equal 0.5. It is clear that the above result comes from a wrong classifier which just ignores the input and just predicts one of the classes as output.

Another metric that is worth considering is the area under the ROC curve (AUROC). The biggest advantage of using ROC curve is that it is independent of the change in proportion of the outcomes. The ROC curve is the plot between sensitivity and (1- specificity). (1- specificity) is also known as false positive rate and sensitivity is also known as true positive rate.



AUC values	Test quality
0.9–1.0	Excellent
0.8–0.9	Very good
0.7–0.8	Good
0.6–0.7	Satisfactory
0.5–0.6	Unsatisfactory

Figure 6.2: Graphic explanation of the AUROC performance evaluation. A classifier achieving around .5 of AUROC is considered as a random classifier

Note that the area of entire square is $1 \cdot 1 = 1$. Hence AUROC itself is the ratio under the curve and the total area. Following are a few thumb rules showed in Figure 6.2:

- .90-1 = excellent (A)
- .80-.90 = very good (B)
- .70-.80 = good (C)
- .60-.70 = satisfactory (D)
- .50-.60 = unsatisfactory/fail (F)

Chapter 7

Results

In general, all the models obtained comparable results to the ones obtained by [1]. Moreover, the state-of-the-art is achieved for some specific configurations. The study demonstrates how this discerning between indoor and outdoor environments can be successfully conducted by the only usage of both machine learning tools and magnetic signals.

For every model and sensor position, three different windows (0.5s, 1s, 2s) are computed and then evaluated. The metric used for the LOSO validation is the accuracy. Following the idea behind the work proposed by [1], where no overlapping window is used. A comparison among the whole set of models is then computed, to assess which model performs the best. Validation results are displayed in Table 7.1.

The testing subjects are collected in different cities, belonging to different states, and conducted on a wide range of free-living activities, as well as exploring various environments (e.g. parks, hospitals, universities, shops). A total number of 5 participants are used as independent test setif , each of them with a different ratio of indoor/outdoor samples (Table 7.2).

As mentioned in Section 6.5, accuracy, f1-score, and AUROC score are exploited. Note that the goal of this research is not only to focus in achieving quite high performances, but also to understand which of the sensor position can bring the most

	All-sensors Accuracy	Wrist Accuracy	Lumbar Accuracy	Left foot Accuracy	Right foot Accuracy
Logistic Regression	0.66 (0.12)	0.67 (0.11)	0.62 (0.19)	0.73 (0.10)	0.71 (0.12)
Random Forest	0.80 (0.10)	0.73 (0.11)	0.66 (0.14)	0.80 (0.10)	0.79 (0.09)
XGB	0.80 (0.09)	0.74 (0.10)	0.67 (0.10)	0.82 (0.08)	0.81 (0.12)
	All-sensors Accuracy	Wrist Accuracy	Lumbar Accuracy	Left foot Accuracy	Right foot Accuracy
Logistic Regression	0.68 (0.13)	0.71 (0.12)	0.66 (0.19)	0.70 (0.13)	0.71 (0.15)
Random Forest	0.78 (0.10)	0.73 (0.13)	0.68 (0.15)	0.82 (0.10)	0.78 (0.11)
XGB	0.82 (0.10)	0.73 (0.10)	0.67 (0.10)	0.84 (0.10)	0.81 (0.09)
	All-sensors Accuracy	Wrist Accuracy	Lumbar Accuracy	Left foot Accuracy	Right foot Accuracy
Logistic Regression	0.72 (0.09)	0.60 (0.05)	0.61 (0.06)	0.75 (0.12)	0.70 (0.09)
Random Forest	0.76 (0.10)	0.65 (0.08)	0.70 (0.11)	0.80 (0.08)	0.80 (0.10)
XGB	0.81 (0.10)	0.65 (0.08)	0.71 (0.11)	0.82 (0.09)	0.80 (0.10)

Table 7.1: Validation results for 0.5s, 1s, 2s

ID	**Indoor samples**	**Total experiment duration (min)**
1001	86%	∼164
2003	79%	∼194
3005	100%	∼75
6002	79%	∼504
6003	52%	∼167

Table 7.2: Indoor ratio with respect to the overall samples, for each subject

useful information for detecting whether or not a subject is in an indoor environment. If successful, these results can be either used for successive data science studies or exploited in the biomedical scientific community to assess walk conditions in outdoor or indoor environments. For this reason, an exhaustive ensemble of metrics must be used: for each of them the *average value* as well as the *standard deviation (std)* is presented as (*average, std*), to assess not only the mere final result, but how the model behaves in different scenarios and human environments. For instance, let's consider two model performances:

1 Accuracy = 0.92, Standard Deviation = 0.20,

2 Accuracy = 0.88, Standard Deviation = 0.04

Considering this research study, in this toy example the second model can be the

best choice. Even though the average accuracy is 4% less, the standard deviation explains how the second model is more stable across the testing subjects. This is both crucial and beneficial for a future tool deployment.

Eventually, this section will provide a wide model evaluation according to the configuration granularity, based on the testing results. Firstly, all the models are compared inside each sensor configuration. From this pool, the best sensor configuration is consider for the final comparison among the best aggregating temporal window. As a final discussion, a direct comparison with the corresponding best model against the baseline inducted by MagIO [1] is followed.

The next tables are divided as it follows:

- Table 7.3 will show the performances of every model and sensor for the data aggregated on 0.5s windows.
- Table 7.4 refers to model performances achieved by using the 1s window for data aggregation
- Table 7.5 will finally display the results achieved with 2s windows timeframes.

All-sensors

	Accuracy	F1-score	AUROC
Logistic Regression	0.87 (0.06)	0.87 (0.07)	0.74 (0.32)
Random Forest	0.83 (0.18)	0.83 (0.18)	0.93 (0.04)
XGB	0.87 (0.05)	0.87 (0.06)	0.81 (0.24)
Stacked-LSTM	0.71 (0.12)	0.79 (0.13)	0.65 (0.25)

Wrist

	Accuracy	F1-score	AUROC
Logistic Regression	0.77 (0.08)	0.76 (0.10)	0.65 (0.17)
Random Forest	0.78 (0.13)	0.76 (0.15)	0.76 (0.07)
XGB	0.80 (0.08)	0.81 (0.10)	0.76 (0.14)
Stacked-LSTM	0.74 (0.09)	0.74 (0.10)	0.70 (0.08)

Lumbar

	Accuracy	F1-score	AUROC
Logistic Regression	0.84 (0.07)	0.83 (0.10)	0.56 (0.39)
Random Forest	0.75 (0.06)	0.76 (0.05)	0.77 (0.16)
XGB	0.67 (0.28)	0.68 (0.22)	0.65 (0.34)
Stacked-LSTM	0.68 (0.20)	0.67 (0.18)	0.54 (0.23)

Left foot

	Accuracy	F1-score	AUROC
Logistic Regression	0.86 (0.09)	0.86 (0.10)	0.83 (0.10)
Random Forest	0.87 (0.08)	0.88 (0.08)	**0.92 (0.03)**
XGB	**0.89 (0.07)**	**0.90 (0.07)**	0.91 (0.05)
Stacked-LSTM	0.84 (0.12)	0.82(0.11)	0.82 (0.09)

Right foot

	Accuracy	F1-score	AUROC
Logistic Regression	0.86 (0.10)	0.82 (0.12)	0.81 (0.12)
Random Forest	0.85 (0.08)	0.87 (0.09)	0.90 (0.05)
XGB	0.82 (0.16)	0.81 (0.19)	0.86 (0.12)
Stacked-LSTM	0.82 (0.10)	0.80 (0.08)	0.83 (0.11)

Table 7.3: Performances for the 0.5s time window. Every sensor configuration is considered.

All-sensors

	Accuracy	F1-score	AUROC
Logistic Regression	0.84 (0.12)	0.83 (0.13)	0.69 (0.34)
Random Forest	0.86 (0.11)	0.87 (0.11)	0.93 (0.06)
XGB	0.83 (0.11)	0.84 (0.06)	0.87 (0.13)
Stacked-LSTM	0.66 (0.20)	0.69 (0.13)	0.63 (0.25)

Wrist

	Accuracy	F1-score	AUROC
Logistic Regression	0.71 (0.06)	0.79 (0.08)	0.64 (0.20)
Random Forest	0.78 (0.10)	0.76 (0.13)	0.78 (0.02)
XGB	0.79 (0.06)	0.79 (0.08)	0.74 (0.07)
Stacked-LSTM	0.74 (0.10)	0.74 (0.15)	0.67 (0.06)

Lumbar

	Accuracy	F1-score	AUROC
Logistic Regression	0.85 (0.07)	0.83 (0.10)	0.55 (0.41)
Random Forest	0.75 (0.11)	0.79 (0.07)	0.76 (0.18)
XGB	0.69 (0.29)	0.70 (0.23)	0.71 (0.30)
Stacked-LSTM	0.68 (0.11)	0.74 (0.12)	0.69 (0.06)

Left foot

	Accuracy	F1-score	AUROC
Logistic Regression	0.87 (0.08)	0.88 (0.09)	0.83 (0.13)
Random Forest	0.80 (0.22)	0.81 (0.22)	**0.94 (0.03)**
XGB	**0.90 (0.07)**	**0.90 (0.07)**	0.92 (0.05)
Stacked-LSTM	0.80 (0.16)	0.80 (0.16)	0.85 (0.10)

Right foot

	Accuracy	F1-score	AUROC
Logistic Regression	0.88 (0.07)	0.86 (0.09)	0.86 (0.09)
Random Forest	0.88 (0.07)	0.88 (0.08)	0.91 (0.04)
XGB	0.85 (0.14)	0.85 (0.14)	0.90 (0.08)
Stacked-LSTM	0.78 (0.14)	0.79 (0.12)	0.81 (0.10)

Table 7.4: Performances for the 1s time window. Every sensor configuration is considered.

All-sensors

	Accuracy	F1-score	AUROC
Logistic Regression	0.84 (0.13)	0.84 (0.13)	0.68 (0.38)
Random Forest	0.76 (0.18)	0.79 (0.13)	0.94 (0.05)
XGB	0.83 (0.10)	0.84 (0.06)	0.87 (0.14)
Stacked-LSTM	0.78 (0.11)	0.80 (0.09)	0.77 (0.11)

Wrist

	Accuracy	F1-score	AUROC
Logistic Regression	0.73 (0.09)	0.73 (0.12)	0.58 (0.32)
Random Forest	0.76 (0.13)	0.74 (0.18)	0.78 (0.01)
XGB	0.82 (0.06)	0.82 (0.08)	0.79 (0.12)
Stacked-LSTM	0.69 (0.18)	0.72 (0.12)	0.70 (0.08)

Lumbar

	Accuracy	F1-score	AUROC
Logistic Regression	0.84 (0.09)	0.82 (0.12)	0.49 (0.42)
Random Forest	0.78 (0.08)	0.81 (0.06)	0.80 (0.12)
XGB	0.68 (0.28)	0.70 (0.23)	0.75 (0.24)
Stacked-LSTM	0.67 (0.08)	0.66 (0.11)	0.61 (0.22)

Left foot

	Accuracy	F1-score	AUROC
Logistic Regression	0.87 (0.05)	0.88 (0.06)	0.85 (0.10)
Random Forest	0.80 (0.21)	0.83 (0.22)	0.94 (0.04)
XGB	**0.89 (0.05)**	**0.90 (0.05)**	**0.94 (0.03)**
Stacked-LSTM	0.84 (0.07)	0.86 (0.06)	0.83 (0.10)

Right foot

	Accuracy	F1-score	AUROC
Logistic Regression	0.85 (0.12)	0.83 (0.14)	0.90 (0.09)
Random Forest	0.86 (0.09)	0.86 (0.10)	0.91 (0.04)
XGB	0.84 (0.13)	0.83 (0.16)	0.87 (0.10)
Stacked-LSTM	0.81 (0.08)	0.83 (0.10)	0.82 (0.08)

Table 7.5: Performances for the 2s time window. Every sensor configuration is considered.

7.1 Dimensionality reduction

Taking a first look at the feature space used for the machine learning models, it's also worth considering if it might be necessary to reduce the dimensionality of the dataset. The mathematical function that better satisfies our need is *principal component analysis (PCA)*. This is needed for exploring different projection spaces starting from the given temporal-frequency features, and understand how many of them give useful information, i.e. the highest variation in data. More details can be found in on Appendix 8.2.2.

The reason why this is taken into account is because several statistical algorithms, such as XGB and random forest, may benefit from a dimensionality reduction [17, 23], escaping from the curse of dimensionality (Appendix 8.2.1), and trying to create better decision trees with less redundant information. The intuition in using this mathematical trick derives from the high number of temporal-frequency features extracted for some models (83 for single-sensor configuration, 332 if all sensors are involved). Moreover, even though applying PCA directly on non-stationary time series is feasible, yet not so common, this method is not used to process the raw magnetic signals in this research, not involving the deep learning networks in its analysis. The key value to select the right number of embedded PCA dimensions is the *explained variance ratio*, a value that goes from 0 to 1, correlated to the number of principal components used for the new projection. For this experiments, it was set heuristically to 0.85, obtaining a total number of new features equal to 12, as shown in Figure 7.1. Since this approach may be beneficial for the aforementioned machine learning models, then the process is the same as discussed in Chapter 7, segmenting each evaluation through models, sensors and window timeframes. Table 7.6, 7.7, 7.8 will show how models behave with PCA. In general, an increment in the performance is expected.

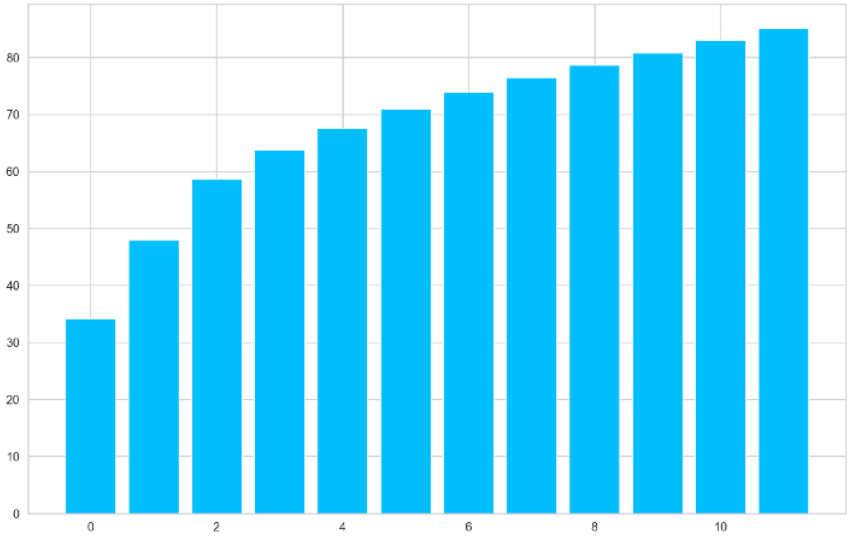


Figure 7.1: Explained variance ration and relative components. for each component the cumulative sum of the variace ratio gained until that iteration is done.

	All-sensors (PCA)		
	Accuracy	F1-score	AUROC
Logistic Regression	0.79 (0.17)	0.71 (0.23)	0.56 (0.01)
Random Forest	0.79 (0.17)	0.71 (0.23)	0.57 (0.01)
XGB	0.79 (0.17)	0.70 (0.18)	0.60 (0.01)
	Wrist (PCA)		
	Accuracy	F1-score	AUROC
Logistic Regression	0.79 (0.17)	0.71 (0.23)	0.39 (0.14)
Random Forest	0.78 (0.15)	0.74 (0.18)	0.48 (0.05)
XGB	0.70 (0.15)	0.70 (0.18)	0.51 (0.09)
	Lumbar (PCA)		
	Accuracy	F1-score	AUROC
Logistic Regression	0.79 (0.17)	0.71 (0.23)	0.39 (0.37)
Random Forest	0.64 (0.28)	0.67 (0.23)	0.59 (0.33)
XGB	0.62 (0.28)	0.65 (0.23)	0.61 (0.27)
	Left foot (PCA)		
	Accuracy	F1-score	AUROC
Logistic Regression	0.84 (0.09)	0.83 (0.10)	0.68 (0.18)
Random Forest	0.89 (0.07)	0.90 (0.07)	0.94 (0.04)
XGB	**0.89 (0.05)**	**0.90 (0.05)**	**0.94 (0.03)**
	Right foot (PCA)		
	Accuracy	F1-score	AUROC
Logistic Regression	0.81 (0.16)	0.78 (0.19)	0.76 (0.12)
Random Forest	0.84 (0.09)	0.84 (0.10)	0.82 (0.11)
XGB	0.82 (0.13)	0.82 (0.16)	0.81 (0.13)

Table 7.6: PCA-based performances for 0.5s window timeframe.

All-sensors (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.88 (0.08)	0.85 (0.11)	0.79 (0.10)
Random Forest	0.88 (0.05)	0.88 (0.07)	0.88 (0.05)
XGB	0.87 (0.02)	0.88 (0.04)	0.89 (0.03)
Wrist (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.79 (0.17)	0.71 (0.23)	0.36 (0.13)
Random Forest	0.74 (0.13)	0.70 (0.21)	0.361 (0.14)
XGB	0.76 (0.08)	0.75 (0.12)	0.62 (0.07)
Lumbar (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.79 (0.17)	0.71 (0.23)	0.28 (0.26)
Random Forest	0.65 (0.21)	0.67 (0.14)	0.62 (0.27)
XGB	0.66 (0.22)	0.68 (0.15)	0.66 (0.24)
Left foot (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.86 (0.07)	0.86 (0.09)	0.72 (0.18)
Random Forest	**0.92 (0.04)**	**0.92 (0.05)**	**0.95 (0.03)**
XGB	0.88 (0.08)	0.89 (0.09)	0.94 (0.04)
Left foot (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.84 (0.11)	0.83 (0.11)	0.83 (0.04)
Random Forest	0.89 (0.06)	0.88 (0.07)	0.91 (0.03)
XGB	0.83 (0.12)	0.82 (0.15)	0.71 (0.32)

Table 7.7: PCA-based performances for 1s window timeframe.

All-sensors (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.88 (0.07)	0.86 (0.09)	0.66 (0.37)
Random Forest	0.84 (0.05)	0.84 (0.07)	0.88 (0.08)
XGB	0.84 (0.07)	0.86 (0.05)	0.85 (0.19)
Wrist (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.79 (0.17)	0.71 (0.23)	0.44 (0.25)
Random Forest	0.76 (0.17)	0.70 (0.22)	0.47 (0.18)
XGB	0.73 (0.12)	0.73 (0.14)	0.59 (0.12)
Lumbar (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.78 (0.14)	0.76 (0.16)	0.51 (0.28)
Random Forest	0.79 (0.08)	0.78 (0.11)	0.69 (0.17)
XGB	0.77 (0.04)	0.79 (0.05)	0.62 (0.31)
Left foot (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.86 (0.08)	0.85 (0.09)	0.72 (0.20)
Random Forest	**0.90 (0.04)**	**0.91 (0.05)**	**0.94 (0.03)**
XGB	0.90 (0.06)	0.91 (0.06)	0.93 (0.05)
Right foot (PCA)			
	Accuracy	F1-score	AUROC
Logistic Regression	0.82 (0.12)	0.82 (0.13)	0.82 (0.06)
Random Forest	0.89 (0.07)	0.89 (0.08)	0.90 (0.04)
XGB	0.83 (0.12)	0.83 (0.14)	0.86 (0.07)

Table 7.8: PCA-based performances for 2s window timeframe.

7.2 Discussion

Despite of the well-established usage of deep learning in time series and feature retrieval, the stacked-LSTM model didn't perform well as intended, being outperformed by XGB and random forest. However, this can be expected while considering noisy data as the one used for this research, for which a deep learning approach couldn't be enough to achieve state-of-the-art, and manual intervention in extracting feature is needed.

Among all the configurations, foot sensors result to be the most stable in term of signals, letting the classifiers follow a clear pattern to retrieve the best information. Figure 7.2 provides a visual representation of this intuition.

Furthermore, the intuition behind PCA confirmed to be the key idea for the models in achieving outstanding results in the foot positioning. Even though the slightly difference in results between LF and RF, it can be ensured that for following studies a single foot/feet sensor system may be recommended instead of the whole set. This can be beneficial for two main reasons: firstly, the one-sensor configuration is easier to deal with for experimental trials and setup; secondly, the model can infer faster outcomes if working with less features.

On the other hand, the worst configuration in every experimental window seems to be the pelvic sensor, LB: this can be caused by a more "broad" and noise-friendly signal spectrum for which the models don't may be able to discern between the two typologies of environments. Wrist gave a slightly better outcomes, although it's 10% lower than the optimal configuration: this isn't a surprise, considering the high usage of electronic devices during the day used by a subject, for which the hand is mandatory (e.g. smartphone, laptop, tablet).

This thesis also demonstrated how simpler models, if tuned accordingly, can lead to state-of-the-art performances, and outperform deeper architectures in quite complex tasks.

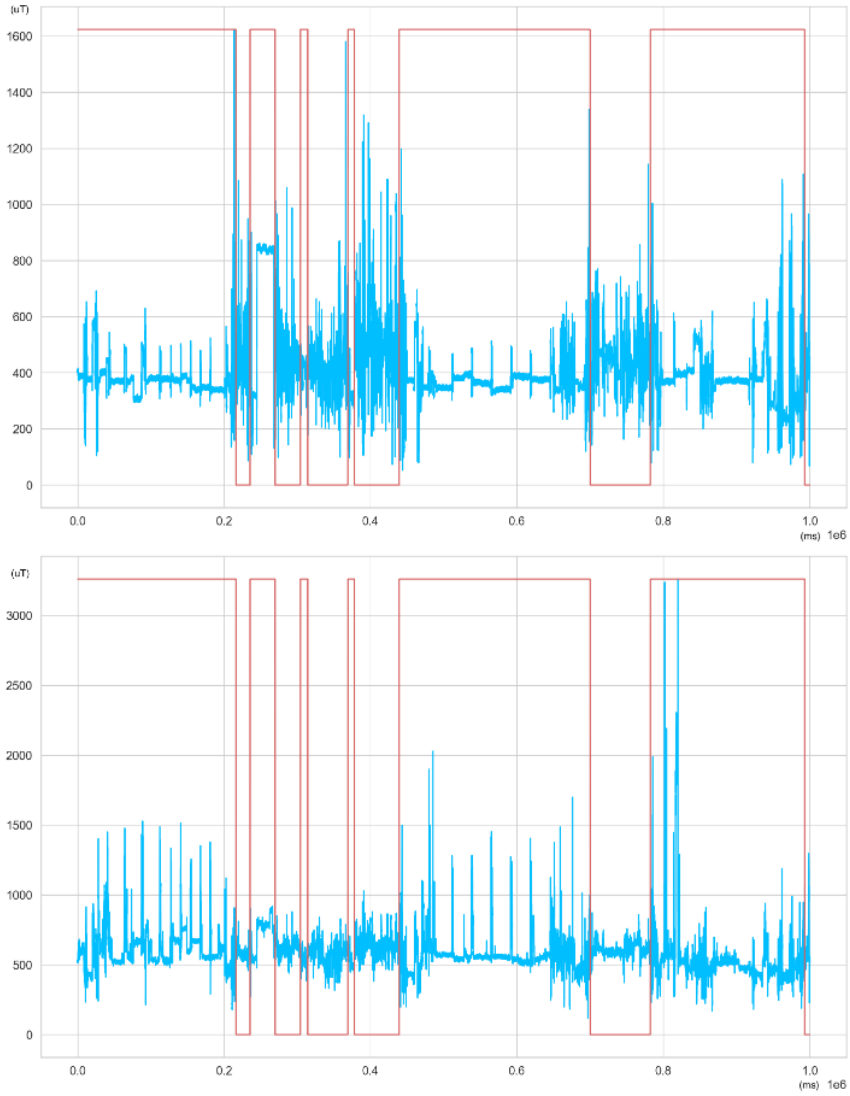


Figure 7.2: Comparison with LF norm (top), and WR norm (bottom) for a random sampled subject. The blue lines displays the raw signal expressed in μT, while the red ones are the indoor labels, scaled to the maximum of the signals for a better visualization. Wrist signals are more hectic in term of stability between indoor and outdoor samples with respect to the left foot signals. This can be seen especially with indoor timestamps.

To have a synthetic view of the achieved results, Table 7.9 includes both a recap of the overall best configuration, and a 2s comparison with the current baseline showed by MagIO [1]. The random forest re-affirms to be the best choice in several machine learning purposes, along with a right approach in dimensionality reduction, i.e. PCA.

Model	Accuracy	Window length	Sensor used
MagIO (baseline)	86.3%	2 s	Smartphone
Proposed feature set + PCA + RF	90%	2 s	Left Foot
Proposed feature set + PCA + RF	**92%**	1 s	Left Foot

Table 7.9: Best model and best setup overall. PCA and random forest may lead to astonishing performances if tuned correctly, leading in low inference time, low memory consuption, and excellent results. Even though the best window is set to 1s, it's worth mentioning how the proposed model beats the straight 1:1 comparison with the baseline, using the 2s window as mentioned in [1]

Although the authors of [1] used a smartphone, and consequently a different magnetometer from the one exploited for this thesis, the big problem behind their work was to investigate if that position was good enough to not explore other solutions. For instance, it's interesting how even the WR sensor performs worse than the LF/RF configurations: while it was expected for the all-sensors setup to have comparable results with MagIO [1], this didn't happen for the wrist, i.e. the most similar positioning to the one exploited in the baseline. Nonetheless, this thesis was conducted for a clinical purpose, that often doesn't involve a direct usage of a smartphone, due to its battery consumption and signal fuzz.

Chapter 8

Conclusions

The aims of this thesis were to develop a simple, yet reliable statistical model to discern indoor from outdoor environments based on data recorded by a tri-axial digital magnetic sensor, and to compare the resulting outcomes with the actual baseline. As a result of this research, a model able to detect these classes of environments is developed, reaching the **92% average accuracy (0.04)**, **0.92 F1-score (0.05)**, and **0.95 AUROC score (0.03)**, outperforming the current state-of-the-art. For every metric, PCA was useful to improve the model capabilities in generalization, as well as the random forest. The best sensor configuration turns to be the foot one.

Good results are also achieved by using all sensors. Intuitively, this setup is not the best because it lets the models training slowly and not learning enough good samples from all the signals exploited. This can be expected: during the learning phase, both the "good" signals (the ones coming from feet sensors) and "bad" signals (wrist and lumbar) are explored, indicating that the proposed models receive a mix of useful and noisy signals to elaborate. Poor performances are then observed using the wrist and lumbar sensors alone: the metrics standard deviation among all the subject isn't stable, thus indicating that some subjects were badly affected in terms of results. Among all the models, random forest and XGB are the best ones, followed by stacked-LSTM and logistic regression. Hidden states' feature manipulation by

deep learning architectures is not found to have a significant effect for this study.

Results of indoor classification experiments also evidenced the enhancement of performances if frequency-based features are involved in the pre-processing. Fast Fourier transform (FFT) alone can efficiently identify different patterns between indoor and outdoor samples, due to its projection in a new feature space. PCA on the other hand emphasises this temporal-frequency variation, compressing their information in a small batch of embedded dimensions. However, PCA was tuned accordingly to an heuristic threshold of 85% of *explained variance ratio* to be retained by the new features: if oversampled, the dimensionality reduction given by PCA increases the complexity of the dataset, hence a tradeoff between amount of informations stored and low complexity must be taken into account.

It would appear that the effects of a different window length, model selection, and feature extraction approach are enough to lead to better performance in detecting indoor and outdoor environments. Nonetheless, it may necessary to investigate different approaches to increment the new performance baseline. For instance, current machine learning research is exploring Transformers for time series forecasting or classification. In particular, Google research proposed a new method called FNet [24] that uses a FFT-based *attention mechanism* instead of the standard one: since FFT seems to be the right way to investigate more the temporal features of magnetic signals, it may be worth to try this new implementation. Also, the proposed stacked-LSTM can be further tuned exploiting more powerful CPU and GPU machines, increasing the number of hidden layers and/or the number of available RNN cells.

Appendix

8.1 Nomenclatures

- WR: non dominant wrist
- LB: lumbar
- LF: left foot
- RF: right foot (this notation isn't used for Random Forest)
- ML: machine learning
- DL: deep learning
- AI: artificial intelligence
- MCP: McCulloch-Pitts
- XGB: eXtreme gradient boost
- SGD: stochastic gradient descent
- FFT: fast-Fourier transform
- PCA: principal component analysis
- ADAM: adaptive moment estimation
- STD: standard deviation

- ROC: receiving operator characteristic
- AUROC: area under the receiving operator characteristic
- MEMS: micro electro-mechanical systems
- MIMUs: magneto inertial measurement units
- INDIP: inertial module with distance sensors and pressure insoles

8.2 Mathematical notes

8.2.1 Curse of dimensionality

The curse of dimensionality refers to the phenomena that occur when classifying, organizing, and analyzing high dimensional data that does not occur in low dimensional spaces, specifically the issue of data sparsity and "closeness" of data. [23]

Sparsity of data occurs when moving to higher dimensions. the volume of the space represented grows so quickly that the data cannot keep up and thus becomes sparse. The sparsity issue is a major one for anyone whose goal has some statistical significance. As the data space seen above moves from one dimension to two dimensions and finally to three dimensions, the given data fills less and less of the data space. In order to maintain an accurate representation of the space, the data for analysis grows exponentially

The second issue that arises is related to sorting or classifying the data. In low dimensional spaces, data may seem very similar but the higher the dimension the further these data points may seem to be.

A careful choice of the number of dimensions (features) to be used is the prerogative of the data scientist training the network. In general the smaller the size of the training set, the fewer features it should use.

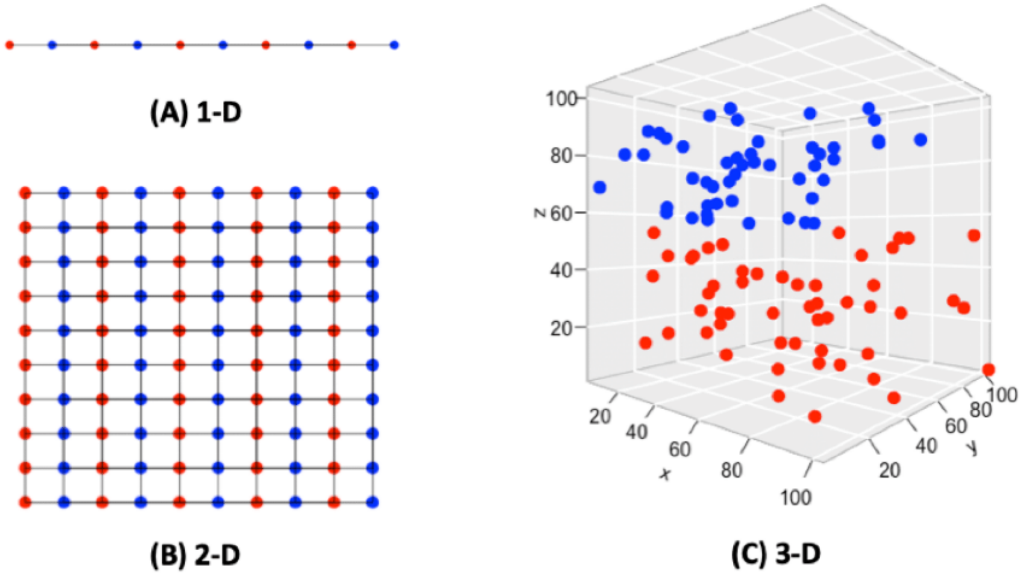


Figure 8.1: Visual explaination of increasing dimensionality of feature in a sparse matrix. The more dimensions are added, the more "distant" is a sample from another.

8.2.2 Principal component analysis

PCA builds its foundations on *compressed sensing*, in which the prior assumption is the original operating vector being *sparse* in some basis, i.e:

$$||x||_0 := |\{i \ : \ x_i \neq 0\}| \leq s$$

$$s = \text{ the number of nonzero elements}$$

Surely x can be compressed by representing it using s pairs, however this is not the case: the provided dataset is clearly stated as a non-sparse features matrix. What's different from PCA is the otpimization problem behind the latter technique:

$$\arg\min \sum_{i=1}^{m} ||x_i - UWx_i||_2^2$$

$$\text{s.t. } W \in \mathbb{R}^{n,d},$$

$$U \in \mathbb{R}^{d,n}$$

Roughly, PCA uses this problem to behave as Figure 8.2 shows. According to the

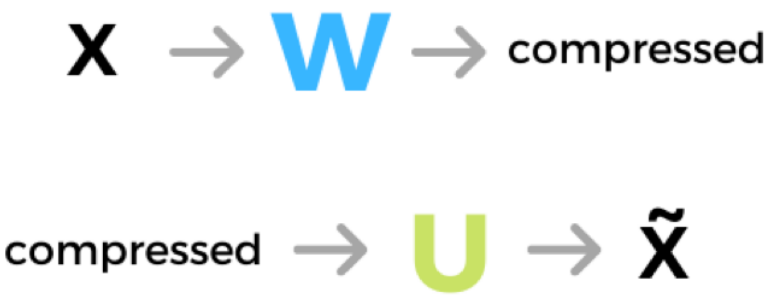


Figure 8.2: Simple and visual representation of the compression process done by PCA. The $\tilde{x}$ remarks that after reverse the operation, only an approximation of the input vector is achieved (while this does not happen in *Compressed Sensing*).

[23] and doing some algebraic calculations, the previous problem becomes:

$$\arg\max \ trace\Big(U^T \sum_{i=1}^{m} x_i x_i^T U\Big)$$

s.t.

$$U \in \mathbb{R}^{d,n} : U^T U = I$$

Mathematically speaking, what the equation is saying is to set the matrix U whose columns are the n eigenvectors of the matrix, let's say, $A = \sum_{i=1}^{m} x_i x_i^T$, corresponding to the largest n eigenvectors of the matrix A itself, then to set $W = U^T$[23].

Moreover, it's common practice to align the sample data before applying PCA, i.e., working with scaled data:

$$(x_1 - \mu), ..., (x_m - \mu)$$

$$\text{where: } \mu = \frac{1}{m} \sum_{i=1}^{m} x_i$$

Now, with this assumption, an important interpretation of PCA could be **variance maximation** [23]. Let's say we want this optimization problem:

$$\arg\max \text{Var}[\langle w, x \rangle] = \arg\max \frac{1}{m} \sum_{i=1}^{m} (\langle w, x \rangle)^2$$

$$s.t. w : ||w|| = 1$$

Note that for every unit vector $w \in \mathbb{R}^d$ and for every $i \in m$:

$$(\langle w, x \rangle)^2 = trace(w^T x_i x_i^T w)$$

Hence, the optimization problem here coincides with the previous one, allowing us to work with the variance maximation due to the *eigendecomposition* of the *covariance matrix*.

8.3 Coding packages

- **MatLab**: "programming and numeric computing platform used by millions of engineers and scientists to analyze data, develop algorithms, and create models". It's used to store the raw INDIP signals.
- **Python**: "programming language that lets the programmers work quickly and integrate systems more effectively". It's the main framework used to develop the system for this research project. It's the base infrastructure for other machine learning libraries.
- **NumPy**: "the fundamental package for scientific computing with Python". Used for data handling, matrix manipulation, feature extraction.
- **Pandas**: "pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language". It's used to load, store and manipulate data directly from matlab files.
- **scikit-learn**: simple and efficient tools for predictive data analysis; accessible to everybody, and reusable in various contexts; built on NumPy, SciPy, and matplotlib. Open source, commercially usable - BSD license. Used to generate the models and dimensionality reduction.

- **matplotlib**: "Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python". Used for plot evaluation, and data visualization

- **PyTorch**: "An open source machine learning framework that accelerates the path from research prototyping to production deployment". Main framework to build the deep learning architecture, data handling and GPU parallelization.

- **PyTorch Lightning**: "The ultimate PyTorch research framework. Scale your models, without the boilerplate". It allows a fast and efficient deployment of deep learning models, along with faster training and evaluation.

Bibliography

[1] I. Ashraf, S. Hur, and Y. Park, "Magio: Magnetic field strength based indoor-outdoor detection with a commercial smartphone," *Micromachines*, vol. 9, no. 10, p. 534, 2018.

[2] M. E. Morris, F. Huxham, J. McGinley, K. Dodd, and R. Iansek, "The biomechanics and motor control of gait in parkinson disease," *Clinical biomechanics*, vol. 16, no. 6, pp. 459–470, 2001.

[3] A. Mirelman, P. Bonato, R. Camicioli, T. D. Ellis, N. Giladi, J. L. Hamilton, C. J. Hass, J. M. Hausdorff, E. Pelosin, and Q. J. Almeida, "Gait impairments in parkinson's disease," *The Lancet Neurology*, vol. 18, no. 7, pp. 697–708, 2019.

[4] S. Bertuletti, A. Cereatti, D. Comotti, M. Caldara, and U. Della Croce, "Static and dynamic accuracy of an innovative miniaturized wearable platform for short range distance measurements for human movement applications.," *Sensors*, 2017.

[5] S. Bertuletti, , U. Della Croce, and A. Cereatti, "A wearable solution for accurate step detection based on the direct measurement of the inter-foot distance," *Biomechanics*, 2019.

[6] S. F., B. S., C. M., D. C. U., M. C., and C. A., "Multi-sensor integration and data fusion for enhancing gait assessment in and out of the laboratory," *Gait and Posture*, 2019.

[7] R. F., "The perceptron: A probabilistic model for information storage and organization in the brain," 1958.

[8] Z. Wu, Q. Xu, J. Li, C. Fu, Q. Xuan, and Y. Xiang, "Passive indoor localization based on csi and naive bayes classification," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 9, pp. 1566–1577, 2018.

[9] A. Morar, A. Moldoveanu, I. Mocanu, F. Moldoveanu, I. E. Radoi, V. Asavei, A. Gradinaru, and A. Butean, "A comprehensive survey of indoor localization methods based on computer vision," *Sensors*, vol. 20, no. 9, p. 2641, 2020.

[10] D. P. Sultana A., Deb K., "Classification of indoor human fall events using deep learning.," 2021.

[11] O. Canovas, P. E. Lopez-de Teruel, and A. Ruiz, "Detecting indoor/outdoor places using wifi signals and adaboost," *IEEE Sensors Journal*, vol. 17, no. 5, pp. 1443–1453, 2017.

[12] B. V., "Gps-based indoor/outdoor detection scheme using machine learning techniques," 2020.

[13] M. S. Grant, S. T. Acton, and S. J. Katzberg, "Terrain moisture classification using gps surface-reflected signals," *IEEE Geoscience and Remote Sensing Letters*, vol. 4, no. 1, pp. 41–45, 2007.

[14] A. Mannini, D. Trojaniello, A. Cereatti, and A. M. Sabatini, "A machine learning framework for gait classification using inertial sensors: Application to elderly, post-stroke and huntington's disease patients," *Sensors*, vol. 16, no. 1, p. 134, 2016.

[15] C. Prakash, R. Kumar, and N. Mittal, "Recent developments in human gait research: parameters, approaches, applications, machine learning techniques,

datasets and challenges," *Artificial Intelligence Review*, vol. 49, no. 1, pp. 1–40, 2018.

[16] P. Khera and N. Kumar, "Role of machine learning in gait analysis: a review," *Journal of Medical Engineering & Technology*, vol. 44, no. 8, pp. 441–467, 2020.

[17] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*, vol. 112. Springer, 2013.

[18] T. K. Ho, "A data complexity analysis of comparative advantages of decision forest constructors," *Pattern Analysis & Applications*, vol. 5, no. 2, pp. 102–112, 2002.

[19] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[20] Z. Zhang, B. He, and Z. Zhang, "Transmask: A compact and fast speech separation model based on transformer," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5764–5768, IEEE, 2021.

[21] S. Shi, Y. Wang, H. Dong, G. Gui, and T. Ohtsuki, "Smartphone-aided human activity recognition method using residual multi-layer perceptron," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, IEEE, 2022.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

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

[24] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontañón, "Fnet: Mixing tokens with fourier transforms," *CoRR*, vol. abs/2105.03824, 2021.