Universidade do Minho
Escola de Engenharia

Carlos José Gomes Campos

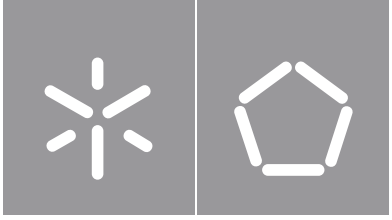
A Machine Learning approach to Boredom Detection in Smartphones

October 2020

A Machine Learning approach to Boredom Detection in Smartphones

Carlos Campos

UMinho | 2020



Universidade do Minho
Escola de Engenharia

Carlos José Gomes Campos

A Machine Learning approach to Boredom Detection in Smartphones

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação do
Professor Doutor Cesar Analide Rodrigues

October 2020

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO POR TERCEIROS

AGRADECIMENTOS

As primeiras pessoas que quero agradecer são os participantes desta investigação. Sem eles, não seria possível construir o dataset utilizado neste estudo científico. Não consigo nomear todos os nomes, mas fica aqui o meu agradecimento colectivo.

Um agradecimento especial aos meus orientadores, Cesar Analide e Bruno Fernandes. Obrigado pela paciência e pela dedicação nesta investigação. Seria muito mais difícil concluir esta etapa da minha vida académica sem vocês.

Uma promessa é uma promessa. Quero agradecer aos meus colegas, José Oliveira e Vitor Castro pelo contributo nesta investigação. Fomos uma bela equipa durante estes dois anos, especialmente no perfil de Sistemas Inteligentes.

A toda a minha família, deixo aqui um agradecimento especial porque sem vocês eu não seria a pessoa que sou hoje. Esta dissertação também deve-se a vocês, porque sem vocês eu não sei se tinha conseguido. Agradeço em especial ao meu primo Luís, à minha prima Inês e ao meu primo Diogo.

Claro que ias ter um parágrafo só para ti Nuno. Sempre foste um exemplo e um mentor para mim. Obrigado por tudo que fizeste por mim este ano. Quero que tudo te corra bem, e espero estar cá para ver os teus sucessos, assim como espero que também vejas os meus.

Santiago não tens idade ainda para perceber isto que estou a escrever, mas espero que um dia te dês ao trabalho de ler a dissertação do teu padrinho. Quero ser um exemplo para ti por isso espero que isto te possa guiar um dia. Quero-te agradecer pelos momentos de alegria que me proporcionaste, especialmente durante os tempos mais negativos que a nossa família passou.

Quero agradecer à minha cúmplice da vida, a minha irmã, que teve um papel fundamental na minha vivência. Tiveste sempre presente nos marcos da minha vida e foste uma peça fundamental para conseguir superar vários obstáculos que encontrei. Também quero-te agradecer pela ajuda que me deste porque sem ela também não tinha concluído a minha dissertação como queria.

Quero agradecer aos meus pais, que ao longo destes anos todos de estudo foram os meus pilares. Sempre fui incentivado por eles a dar o meu melhor e tentar alcançar os meus sonhos. Sem eles esta dissertação não tinha chegado onde chegou. Agradeço-te, pai, que serás sempre o meu herói, e agradeço-te, mãe, que serás sempre o meu primeiro amor.

Por fim, quero agradecer à minha cara metade. Catarina foste e serás sempre o meu porto de abrigo, especialmente durante estes meses mais difíceis que passei. Esta dissertação ganhou muito contigo, assim como a minha vida. Obrigado por fazeres parte do meu mundo e por me tornares uma pessoa melhor. Quero simplesmente dizer que te amo até que o oito se deite (Vais ficar chateada por te mencionar em último, mas a conclusão é a parte mais importante de qualquer documento).

STATEMENT OF INTEGRITY

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

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

RESUMO

Deteção de Aborrecimento durante o uso de um Smartphone através de técnicas de Machine Learning

Desde tempos antigos que o ser humano tenta combater sentimentos negativos, como a tristeza e a solidão. Uma outra emoção que sempre perturbou a humanidade é o aborrecimento. Desta causa nascem vários tipos de arte e também diversos desportos, sendo que estes continuam a ser observados e/ou praticados até hoje. Nos dias de hoje, dado o facto que os smartphones se tornaram dispositivos utilizados a nível global, faz com que as pessoas sejam submetidas a cada vez mais estímulos. Assim sendo, quando as mesmas não estão ocupadas, sentem a necessidade de fazer algo que mantenha o seu cérebro activo. Por esta razão, detectar aborrecimento quando se usa um smartphone, abre caminho para melhorar os índices de sucesso dos sinais de estímulo, com um sistema menos intrusivo e mais inteligente. Numa fase inicial, este trabalho de investigação focou-se na recolha de dados. Desenvolveu-se uma aplicação inicial, para cumprir este objectivo. O principal propósito desta aplicação protótipo é a recolha da gama de valores dos sensores físicos e virtuais de um dispositivo móvel, e dados que possam ilustrar o comportamento digital durante o seu uso. Posteriormente à construção do conjunto de dados, foi realizado uma série de técnicas e processos relacionados com Machine Learning para eleger o melhor modelo possível. Por fim, a última etapa foi a elaboração da aplicação final, já com o modelo ideal incorporado, que é capaz de indicar quando o utilizador de um smartphone está aborrecido, com base nos valores indicados pelos sensores e pelo estado do próprio dispositivo móvel. O modelo incorporado trata-se de uma Rede Neuronal Artificial que tem a capacidade de prever o nível de aborrecimento da pessoa que está a interagir com o telemóvel. Este modelo consegue prever o sentimento em causa com uma precisão de 70%.

Palavras-chave: Aborrecimento, Apredizagem Automática, Computação Afectiva, Inteligência Artificial, Sensores

ABSTRACT

A Machine Learning approach to Boredom Detection in Smartphones

Since ancient times, mankind has been trying to combat negative feelings such as sadness and loneliness. Another emotion that has always disturbed humanity is boredom. This cause is probably the major factor that led to the birth of various types of art and sports, and that has led to the continuity of its practice and observation over the centuries. Today, given the fact that smartphones have become globally used devices, people are increasingly subjected to stimuli. So when they are not busy, they feel the need to do something that keeps their brain active. For this reason, detecting boredom when using a smartphone paves the way for improving the success rates of stimuli signals, with a less intrusive, and more intelligent system. At an early stage, this research work focused on data collection. An initial application was developed, to fulfil this objective. The most relevant purpose of this prototype application is to collect the range of values of the physical and virtual sensors of a mobile device, and data that can illustrate the digital behaviour during its use. Following the construction of the dataset, a series of techniques and processes related to Machine Learning was performed, to choose the best possible model. Finally, the last step was the elaboration of the final application, already with the ideal model incorporated, which can indicate when the smartphone user is bored, based on the values of the sensors and the state of the mobile device itself. The built-in model is an Artificial Neural Network that can predict the level of the boredom of the person interacting with the mobile phone. This model can predict the feeling in question with an accuracy of 70%.

Keywords: Affective Computing, Artificial Intelligence, Boredom, Machine Learning, Smartphone

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

A

AdaBoost Adaptive Boosting

AC Agglomerative Clustering

ANN Artificial Neural Network

B

BIRCH Balanced Iterative Reducing and Clustering using Hierarchies

C

CRISP-DM Cross-industry Standard Process for Data Mining

D

DL Deep Learning

DT Decision Tree

G

GB Gradient Boosting

K

K-Means K-Means Clustering

KNN K-Nearest Neighbors

I

IDE Integrated Development Environment

L

Lin-Reg Linear Regression

Log-reg Logistic Regression

LP Label Propagation

LS Label Spreading

M

ML Machine Learning

MLP Multilayer Perceptron

N

NB Naive Bayes

R

RF Random Forest

RL Reinforcement Learning

RMSE Root-Mean-Square Error

S

SC Spectral Clustering

SL Supervised Learning

SSL Semi-Supervised Learning

SDK Software Development Kit

SVM Support Vector Machine

U

UL Unsupervised Learning

GLOSSARY

Bias is an influence out of proportion for or against an idea or thing. It usually instils a notion of injustice in a specific situation. In the field of Machine Learning, it may be associated with some models aiding the learning process.

Dataset is a set of data records, where each entry represents an example of a situation/entity. Each record has several aspects that characterise it. The target feature is commonly the same variable that is wanted to be identified or quantified.

Entry, register or row, is the assemblage of all features and the target label. In a data collection, it represents an occurrence of a situation or entity.

Feature, or column, is a piece of information about a situation or entity that can be measured, quantified or qualified. In a data collection, each record may have a value that characterizes that information.

Outlier is a statistical term referring to atypical values in a data series. A record is considered an outlier when its value is far above the average, or far below it.

Target is a characteristic of a record in a data collection, such as a feature. However, its role is more important, because this variable is the one to be measured, qualified or quantified. Some learning paradigms need example values for their training process.

Threshold is a value that represents a limit, that when surpassed causes a determined reaction. Commonly it is used in Machine Learning to stop some time consuming computational process.

1. INTRODUCTION

The first chapter of this dissertation portrays the whole introductory notion about this document. Starting by making a contextualization about the theme, the reasons that led to carry out this research, what is expected to be achieved, what kind of problem this investigation is trying to solve, the challenges that may appear, and the methodology used in this work. This chapter ends with a brief presentation of the structure of this dissertation.

1.1 CONTEXTUALIZATION

Since ancient times, mankind always tried to avoid negative feelings, such as sadness and loneliness. Every human with a perfect mental condition, make his lifetime goals in a way that these emotions are avoided. This methodology has accompanied the evolution of our specie throw the ages, and will probably do the same across the next generations. Another identical example is the case of boredom, that always has disturbed people. This human emotion is described by Bergler (1945) as a feeling of displeasure stemming from the fight between the need for intensive psychic action and the lack of interest or the inability to be interested. Nowadays, people are more digitally connected than ever, having access to the Internet on multiple devices, leading to constant exposure to external stimuli. For the same reason, like Pielot et al. (2015) appointed, most people spend a lot of time performing self-stimulation activities, such as web browsing, social networking and playing online games. This can lead to constant exposure to this stimulation, and when this level drops, a feeling of boredom arises, this same thought was also concluded by Eastwood et al. (2012) and Oulasvirta et al. (2012).

There is no doubt that smartphones are indispensable electronic devices for daily life. Studies like Price et al. (2018) claim that by 2020, there will be more than 6 billion smartphones around the globe carried by the public. Another research conducted by Miller (2012) affirms that by 2025 more than 5 billion people on our planet will be using ultra-broadband, sensor-rich smartphones far beyond the abilities of today's iPhones and Androids. Smartphones are usually the devices to look for when this state of boredom has its peak, just like it is indicated by Oulasvirta et al. (2012), and are eventually used to fill the time when this happens, whether on public transportation, work breaks or waiting lines. With that said, a good way to detect the boredom of a person would be through its mobile device. There is a great number of strategies and algorithms that comes to mind to achieve this objective, from stiffer approaches to others that may be more flexible. Traditional software programming has a great difficulty to give a more personalized response without gaining some inconsistency, for example, since boredom is a subjective feeling, it is difficult to indicate that two people are upset when both have a different sense of boredom with the same program. *Machine Learning (ML)* may give us the right answer to this situation, since *ML* models eventually learn over time.

1.2 MOTIVATION AND OBJECTIVES

Imagining a world where our cell phones would have the capability to predict when we are bored and indicate what to do when we search for something to entertain us, opens the path to the existence of a more productive society. An innovation could be the creation of more intelligent and less intrusive recommender systems, for example, with digital entertainment platforms recommending content that could be to the interest of an individual when this same person would feel bored. Numerous areas could use this technology to reach new heights, for example, online stores could send notifications for new promotions when this state of boredom arises, and with that been said, advertising could become more efficient, since the ads themselves would become less annoying. An *ML* approach to boredom detection using smartphones is a more reasonable way to get a global solution for everyone and persevere over time. To make this possible is required to collect a *dataset* and conduct a study over it. This data collection must have various values of sensors that are manufactured within the cell phone, and data corresponding to the state of the device. The ideal population for this test is a group of people of various ages and ethnicities so that it is possible to find a global pattern for boredom. With this research completed, the elected model incorporated in a mobile app would send a signal to its owner when a peak of boredom is detected.

The purpose behind this research is to find a way of predicting when a human being is reaching a state of boredom, without any kind of biometric data from the person itself, and just with the help of our daily companion in these days, our smartphone. To build a generic solution capable of achieving this task, regardless of whether it is a man or a woman, young or old, from the western or eastern hemisphere, numerous milestones must be attained:

- Focus on understanding the purpose of the research;
- Gathering of data and the construction of a *dataset*, with the aid of an embryonic Mobile App;
- Processing the collected data, and its respective cleaning and treatment;
- Use of various modelling techniques;
- Election of the candidate model with better performance;
- Implementation of the final mobile app, with the previously elected model incorporated.

1.3 PROBLEM AND ITS CHALLENGES

As the title of this dissertation indicates, this problem consists of detecting boredom through mobile phones using an *ML* approach. With that been said, perhaps the first question that comes to mind is "And how do you do that?", and the simplest answer, but the one that best clarifies this

doubt is "With data, a lot of data!". These data will serve to feed all the construction of *ML* models that can help in solving this task. Another question that may also arise is "And what does this data represent?", but for this question, the answer is no longer so trivial. The big goal in this research is to try to predict the feeling of boredom only with data related to smartphones. So a good idea would be to collect data that would represent a "print", i.e., snapshot of the state of the mobile device, at various times of the day. This state would be no more than a set of various values, from physical sensors, such as brightness, proximity or gravity, or from virtual sensors, such as the percentage of battery spent in the last minutes, the number or outgoing calls made in a certain interval of time or even the number of notifications received, in an attempt to identify usage patterns that can be associated with the feeling in question. These data just by itself is not sufficient, and it will be required a label for each instance in our hypothetical *dataset*. This label could be represented by a certain value, which could indicate whether the smartphone owner is bored or not.

The state of the art was raised on several studies that are similar to this one, and several difficulties were described, such as, for example, the ideal time to try to identify exactly when a state of boredom arises. Unfortunately for this research, this challenge remains, as there is no linear law for when this feeling occurs. For this to happen, it is necessary to find a way to approach the volunteers of this study about the lack or not of stimulation in their brain throughout the day, but this way cannot interfere in the answer itself, in other words, find a way to know if a person is bored, without bothering. An important factor to take into account is the identification of behaviours and usage patterns associated with mobile devices that may be linked to the state of boredom. As in any study that involves data, it is always a concern if that same data is skewed because if this is true, all the work developed afterwards is compromised. To prevent this from happening, the sampling of participants in this data collection will have to be widely distributed to all ages, professions or economic and social classes.

To summarize, these are the main challenges that this problem offers:

- Encounter the ideal moment to find out if a person is bored or not;
- Find the best way to aboard the participants about their boredom status and guarantee that this approach does not influence their mood;
- Recognize which cell phone sensors can offer an advantage in identifying boredom;
- Define mobile phone usage patterns and behaviours that may be associated with a state of boredom.

1.4 RESEARCH METHODOLOGY

This research will follow a methodology called *Cross-industry Standard Process for Data Mining(CRISP-DM)*. The *CRISP-DM* is a framework for translating business problems into data mining tasks and carrying out data mining projects independent of both the application area and the used technology, as indicated by Moro et al. (2011).

1.4.1 CRISP-DM

The next sections will try to demonstrate each step that belongs to this structured workflow, but before that, for a brief contextualization, Figure 1.1 shows the progress of the events along with all its phases, being taken from the website SmartVision (2015).

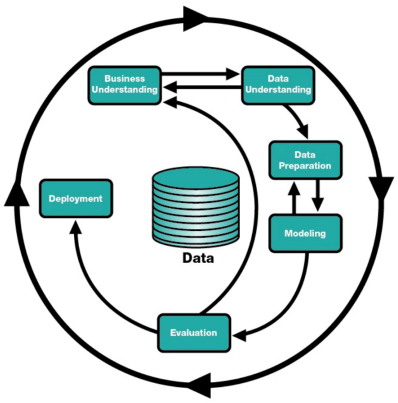


Figure 1.1: Diagram of the phases of the *CRISP-DM* methodology.

Business Understanding

The goal of the work presented in this dissertation is to predict if a certain person will be bored or not, with the help of a smartphone. For the prediction to be possible, a *dataset* is required for some of the possible approaches, and that set of data will theoretically describe various usage patterns and hardware sensors of mobile devices. Predicting if someone is bored translates in the creation of opening window for all types of stimulation signals, from a reminder for that pending task that is very urgent to conclude, to a notification for a promotion for those sneakers that the consumer loves so much. For a smartphone user, this ability would improve their experience as a consumer. Digital content providers also profit from this type of forecast, as they can save more in terms of capital when it comes to advertising, and also because they can increase their revenues since people tend to buy more when they feel bored, such as proved in the research of Close e Kukar-Kinney (2010).

Data Understating

For the data collection process, a mobile application was developed, whose sole purpose is to focus on this collection, rather than offering an improvement in the experience of the smartphone owner. The procedure is simple, whenever there is some interactivity on the part of the user with the mobile device, the application monitors several virtual and hardware sensors, and performs several calculations during that time interval, and at the end of that time sends a notification with a question "Are you bored?", after which the user answers it. To prevent the application from spamming the user, the sending of notifications is controlled so that they are only triggered when the use of the mobile phone is more intense. Figure 1.2 illustrates an iteration of the application operation.

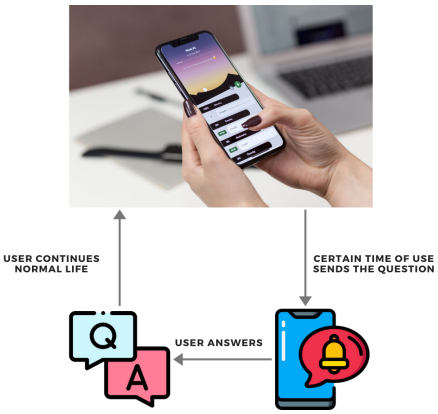


Figure 1.2: Data Collection Cycle.

Data Preparation

This stage is entirely dedicated to the processing of the data collected in the previous phase, this treatment encompasses everything from cleaning noisy data, that can be represented by missing, impossible or null values. This preparation is required to facilitate the training phase of the *ML* models. To facilitate this process, several measures were previously taken in the development of the application for the data collection, such as initializing all *features* with the value -999, so that it is possible to identify which sensors are present in the smartphone, and those that are not. Another important aspect to take into account in this step is the identification of *outliers*, and what possible resolutions can be applied to deal with these records so that their presence does not *bias* the precision of the models. Last but not least, the *dataset* will require normalizations, so that *features* that have a larger scale of values do not take on greater weight in the decision of the models, and make the optimization process less costly in time.

Modelling

The Modelling step is the entire process around the training, validation and tuning of the candidate models for integration in the final application. It is also at this stage that methods and/or algorithms are developed that are then used to optimize the hyperparameters of each model.

Evaluation

What remains from here are the final decisions. Which model is the best, and with what facts can we justify this decision, the learning metrics that will be explained in a further section may give a little help.

Deployment

This research is completed with the integration of the chosen model, in the previous phase, in the final product, an application that will be made available to the public. To guarantee a better service, monitoring of the application will be conducted in terms of performance, so that in the future, interventions can be carried out to improve and upgrade. To make this application perennial and customizable for each individual, and if a user does not agree with the performance of the application, it could be given the opportunity to recalibrate the embedded model, so that it repeats the entire process explained in this chapter, but this time only with data related to the smartphone owner.

1.4.2 Research Hypothesis

The premise for this work is the hypothesis that it is possible to detect boredom when using a smartphone. All the power of the decision will be based on the values of the physical and virtual sensors of the mobile device, and also based on recurring behaviours when using mobile phones.

1.5 DOCUMENT ORGANIZATION

This thesis is organized by the following organization:

- **Chapter 2 - State of Art:** where will be described various researches and studies on this subject, as well as the state of the scientific community regarding the notion of boredom and the ways to identify it. To facilitate the reading process, fundamental concepts related to *ML* will also be introduced;
- **Chapter 3 - Materials and Methods:** where all the resources used to carry out this investigation will be presented. From a detailed explanation of the application created for the

data collection, the exploration and preparation of that same data, as well as the technologies and frameworks used;

- **Chapter 4 - Experiments:** this chapter is dedicated to scrutiny the entire process associated with the experiments. The used setups, the computational resources, and the parameters of each model will also be presented here;
- **Chapter 5 - Results and Discussion:** after all the experiments have been explained, all results will be displayed, as well as a deep discussion and reflection about them;
- **Chapter 6 - Conclusions and Future Work:** last but now least, in the last chapter the conclusions obtained will be exposed, a debate on the good and the less positive aspects, as well as a forecast of the work to be carried out in the future.

2. STATE OF THE ART

Before the reader moves forward to the state of the scientific community about the notion of boredom and the ways to detect it, it might facilitate the understanding of this dissertation, if first is made an introduction about *ML* and its learning paradigms, the most used models, and ways of understanding whether a result is good or misleading.

2.1 MACHINE LEARNING STATUS QUO

The main reason that drove the birth of traditional computing was the need to perform tasks that are humanly impossible, either because humans lack the capacity for abstraction to perform highly complex calculations, or simply because they take a long time to perform them. With the development of the industry, the mindset of the society changes so that anything that is considered long, or even impossible for humans, is routed to computers. Similarly, the same thinking methodology is being used for tasks that are humanly simple but very difficult for a computer, such as recognizing whether a photograph is of a dog or a cat, or recognizing characters and digits in a sentence. Thus is born *ML*, which according to El Naqa e Murphy (2015), is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment. Thus, the possibility arises for a machine to perform tasks that are humanly easy, but computationally arduous.

2.1.1 Paradigms of Learning

As the name implies, *ML* is a science related with the acquisition of knowledge by machines, and this acquisition is commonly divided into 3 big areas, whose names are *Supervised Learning (SL)*, *Unsupervised Learning (UL)* and *Reinforcement Learning (RL)*. Figure 2.1 illustrates these big areas, along with the types of problems each one is used to achieve the solution. The next sections will explain with more detail which one, so that the reader gets a better sense of each paradigm and its content.

Supervised Learning

SL is defined by Chaudhari e Agarwal (2017) as a technique where the training and trained data deal with labelled instances. In order to consider *ML* modelling as *SL*, there is a need for a supervisor. Therefore this supervision is carried out by a prior classification on the input data, along with various attributes that quantify the meaning of the *target*, and the model will train with it. During the training of the model, it will be created a function, or pattern, that best reconcile the input data with the prediction goal. After the training process, the model will apply the developed function to new input data, but this time with no previous label, and will map to the *target feature* value that it thinks is more correct. This whole process is demonstrated in Figure 2.2.

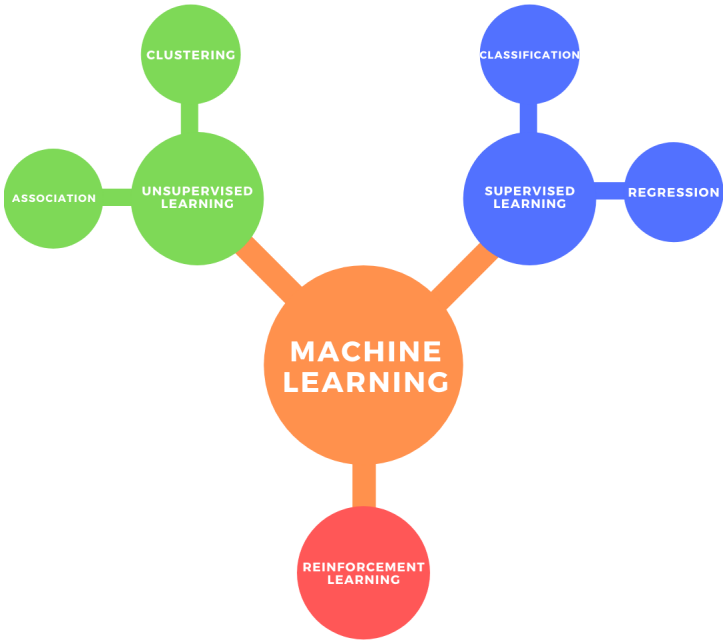


Figure 2.1: Paradigms of Learning.

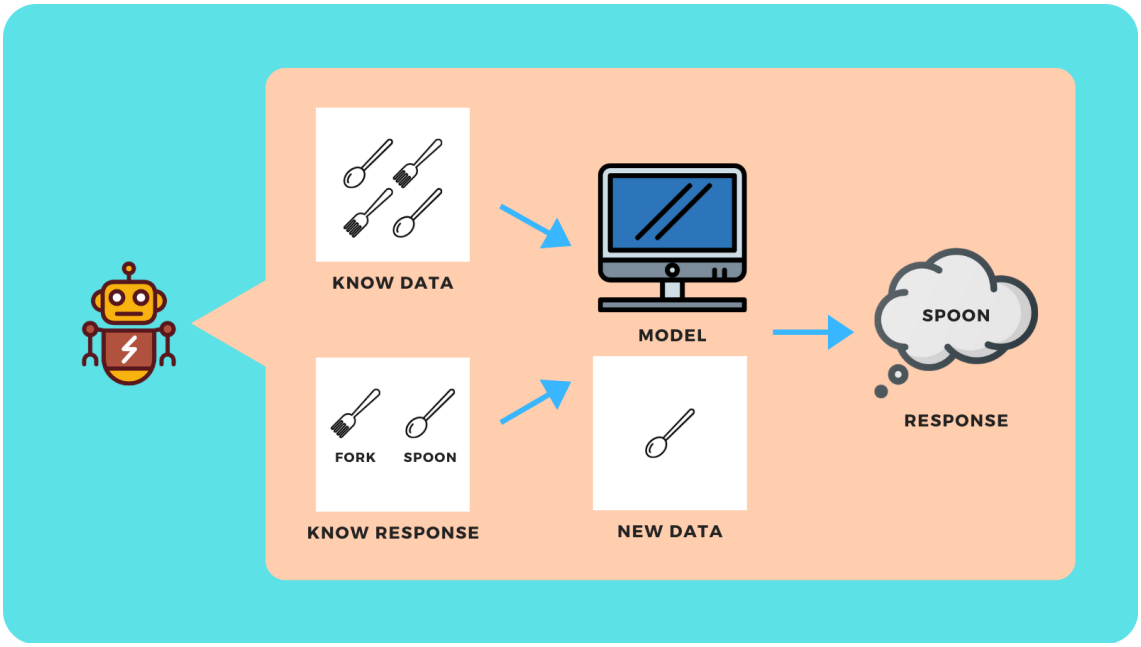


Figure 2.2: Supervised Learning.

In summary, *SL* happens when the model generates a function that maps inputs to desired outputs. One standard formulation of the *SL* task is the classification problem: the learner is required to learn (to approximate the behaviour of) a function which maps a vector into one of several classes by looking at several input-output examples of the function, like affirmed by Ayodele (2010). Depending on the nature of the prediction goal of the problem, it is possible to consider it

into two separate groups, which are *Classification* and *Regression*. Another important type of *SL* models that are worthy to be mentioned are the *Deep Learning (DL)* models. It should be noted that *DL* is not an exclusive type of the *SL* paradigm, it can also be used in *UL* and *RL*, but normally these are mostly applied to *SL*, and so it is also mentioned in this section.

- **Classification:** is a problem where the prediction *target* is a categorical variable, i.e., with two or more classes, for example ”yes” or ”no”, ”dog” or ”cat”, etc;

- **Regression:** is a problem where the prediction *target* it is a continuous variable, i.e., with a plethora of possible answers, for example, predict the amount to invest in advertising;

- **Deep Learning:** is a problem where the models try to solve classification or regression problems using methods that mimic the central nervous system of the human being.

Unsupervised Learning

This paradigm differs from *SL* primarily in the data that are provided, as it does not have any kind of labels that are previously assigned. *UL* is defined by Bowd et al. (2014) as a technique that discerns how the data are organized by learning to separate data into statistically independent groups by cluster analysis, or into representative axes by component analysis, without a priori information regarding class membership. So like the supervised, the unsupervised models also try to construct a recognition pattern through the information received, but instead of assigning it immediately to a category provided to it from the outset, it tries to aggregate the records into several different groups. Finally, the evaluation stage of these models is similar to the supervised models, and with each new case study provided, unsupervised models attempt to group this new record into one of the categories that were designed in the training phase. This whole process is demonstrated in Figure 2.3.

In summary *UL* happens when the model generates a function that maps inputs to certain outputs, but these outputs are created based on the data that are not labelled, Ayodele (2010). In this type of *ML* models it is possible to divide them into two different groups, which name are *Clustering* and *Association*.

- **Clustering:** is a problem where the model tries to divide the objects into clusters which are similar between them and are dissimilar to the objects belonging to another cluster, for example, aggregate clients into various groups, depending on their purchasing behaviour;

- **Association:** is a problem where the model tries to discover the probability of the co-occurrence of items in a collection, for example, which products will be purchased together.

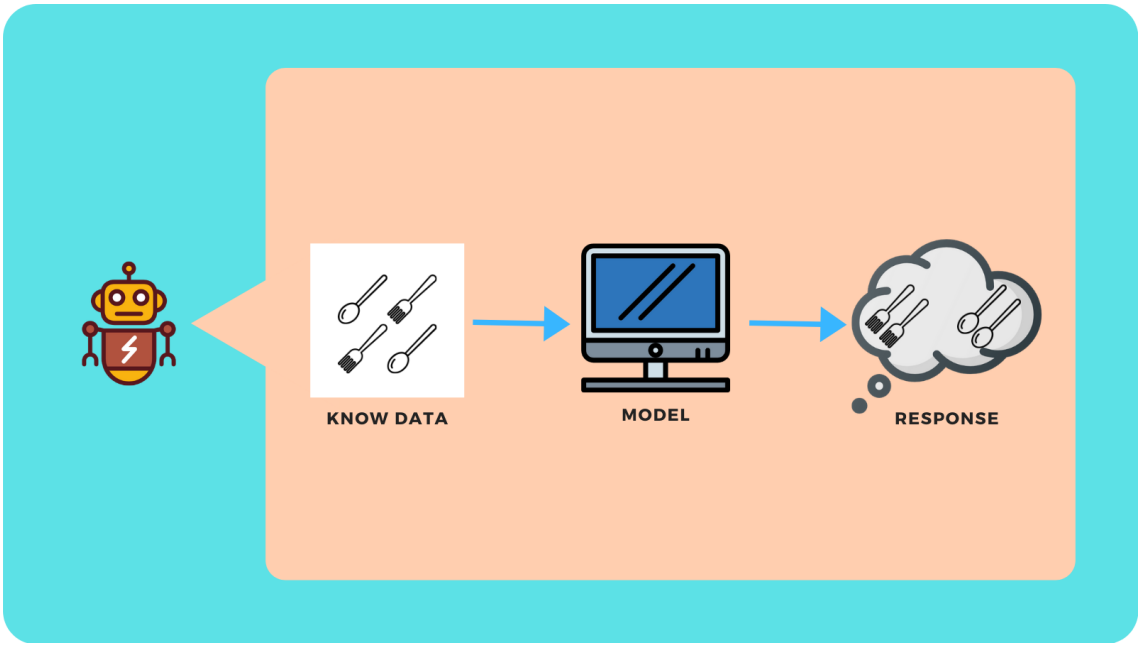


Figure 2.3: Unsupervised Learning.

Semi-Supervised Learning

Although not illustrated in Figure 2.1, recently some authors have stated the existence of a new learning paradigm, called *Semi-Supervised Learning (SSL)*. For Zhou et al. (2004), *SSL* is to design a classifying function which is sufficiently smooth concerning the intrinsic structure collectively revealed by known labelled and unlabelled points. In other words, this methodology is a kind of hybrid combination of the other two mentioned so far. Normally it is performed in three separate phases. First, the models train only with the part of the *dataset* that is labelled, through *SL* algorithms, then the trained model tries to assign the respective label to the unlabelled *entries* of the *dataset*. Finally the model is retrained but now with the entire *dataset*, that consists of the junction of truly labelled *entries*, and predicted labelled *entries*. This behaviour could be glimpsed in Figure 2.4

SSL is a learning paradigm concerned with the study of how computers and natural systems such as humans learn in the presence of both labelled and unlabelled data, as Zhu e Goldberg (2009) state, and derived from this combination, this type of models can be used in the same kind of problems in which both *SL* and *UL* models are used

Reinforcement Learning

Finally, there is the paradigm that differs most from the others, which is *RL*. This learning methodology consists of the insertion of an agent in a specific environment. This agent can perform a certain set of actions that modify the environment that surrounds it. For each action performed the agent receives a response, which can either be in the form of reward or in the form of some kind of injury. The main goal of the agent is to perform the set of actions that ultimately gives the best reward. This

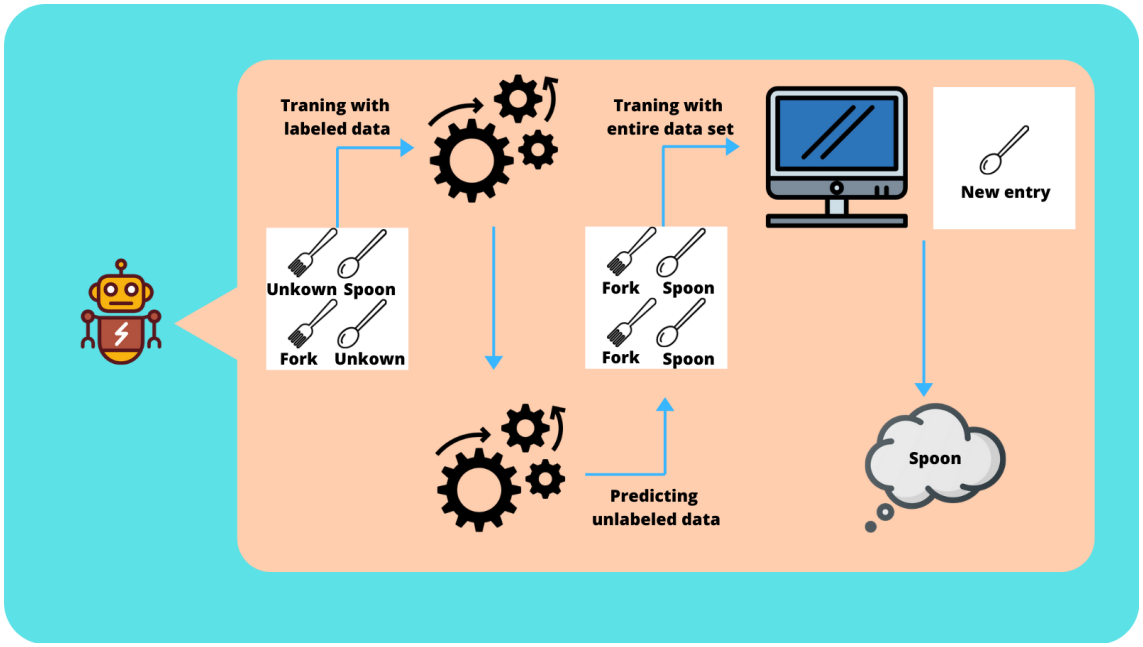


Figure 2.4: Semi-Supervised Learning.

entire process is outlined in Figure 2.5. In summary, *RL* happens when the model learns a policy of how to act given an observation of the world. Every action has some impact on the environment, and the environment provides feedback that guides the learning model (Ayodele (2010)).

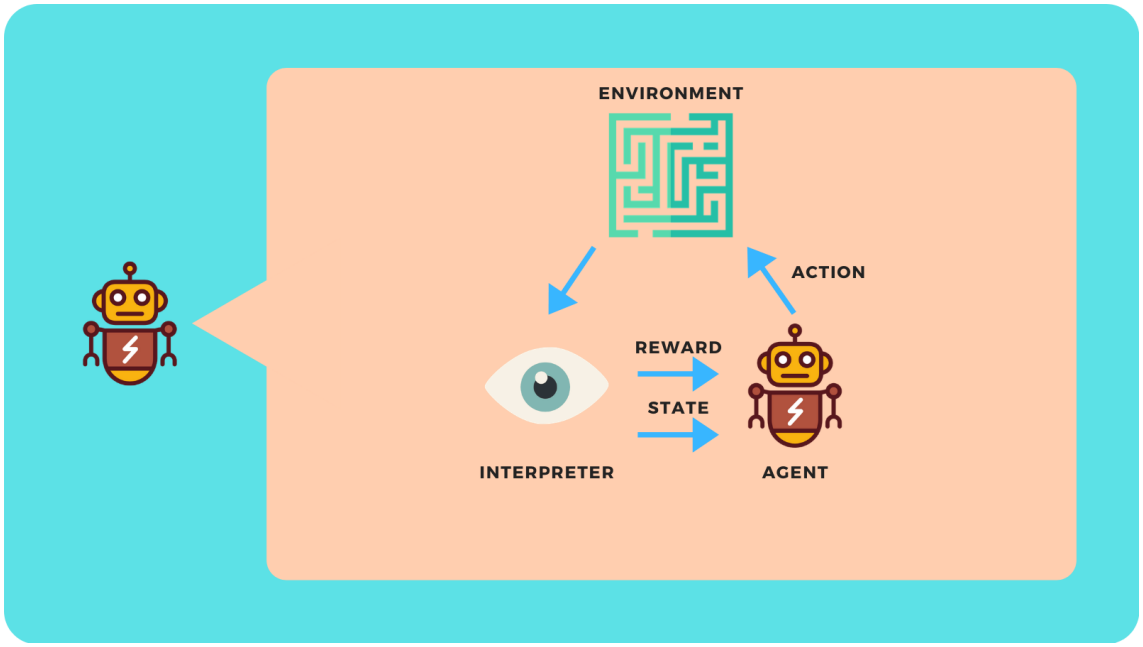


Figure 2.5: Reinforcement Learning.

2.1.2 Machine Learning Models

There are several models used in *ML*, each one works in its way, and are used in different scenarios. There is the possibility of categorizing each one into one of the paradigms that were previously referenced, as well as the possibility of belonging to more than one. But regardless of its kind, each model intends to simulate some human capacity, such that it can be attributed to a machine, either by statistical, empirical or even heuristic methods. The following sections are devoted to explaining some of the most commonly used models as well as their applicability in the real world.

Linear Regression

For a smooth beginning, let us start with one of the simplest *ML* models. During the training process, a *Linear Regression (Lin-Reg)* model tries to construct a linear function that best fits as a representation for the *dataset* given as input. After that, during the prediction phase, the new *entry* is applied to that function, and its classification goes accordingly. Like is established by Almeida et al. (2002), in a simple *Lin-Reg* model, the relationship between the *dataset* variables is established by a straight line, mathematically expressed as:

$$y = a + bx \tag{2.1}$$

where x is the abstract representation for a *entry* in our series of data. Like in the following explanations of the models enunciated in this dissertation, there will be images like Figure 2.6 that may help to imagine the process behind the model, which in this case is a *Lin-Reg*.

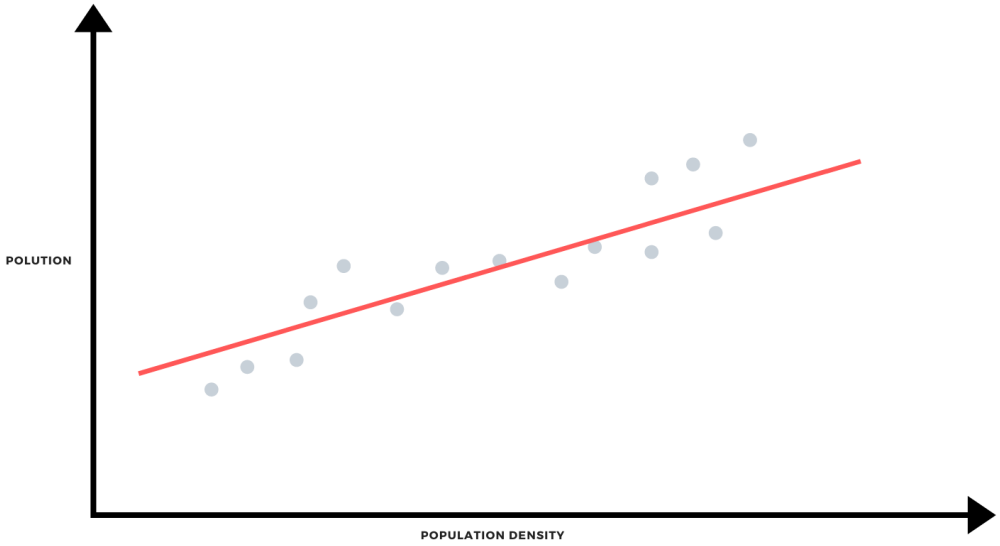


Figure 2.6: Linear Regression Operation.

Logistic Regression

Similar to a *Lin-Reg* model, a *Logistic Regression (Log-Reg)* model also tries to create a function to be applied in the classification of new cases. This new type of function is non-linear and is called by *Sigmoid* function. The name *Sigmoid* derives from the fact that the function has a "S" shape, as Hassan e Akamatsu (2004) say. A *Sigmoid* function in a *Log-Reg* problem establishes the relationship between the *dataset* variables by the "S" shape curve and is mathematically expressed as:

$$y = \frac{1}{1 + e^{-x}} \tag{2.2}$$

where x is the abstract representation for a *entry* in our series of data. As affirmed by Liu et al. (2014), *Log-Reg* is usually used as a classifier, for probabilistic binary or multivariate classification. In Figure 2.7 can be found a schematic example of a classification process of mice based on their weight. Therefore as the weight of a mouse increases, higher are the odds to be classified as obese and vice versa.

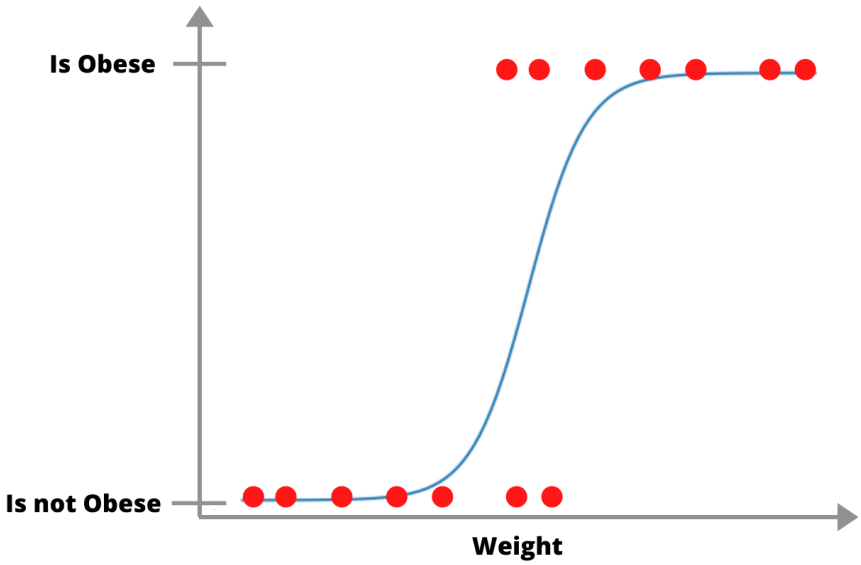


Figure 2.7: Logistic Regression Operation.

Decision Trees

Learning models inspired by *Decision Tree (DT)* are the most practical examples when it comes to *SL* since they use already classified input data to build their structure. A particularity of *DT* that makes them easy to understand and simple to implement, is the fact that their operation does not depend on the nature of the input variables, whether they are numeric or categorical. Sitanggang et al. (2013) defined a *DT* as a model expressing classification rules. It is composed of three types

of nodes: a *Root* node, internal nodes and *Leaf* or terminal nodes. Each leaf node is assigned as a class label. Classification rules can be obtained by traversing the tree from the *Root* node to the *Leaf* nodes (terminals). Each rule consists of test attributes and their value. A *DT* is nothing more than a graph with a *Root*, which represents the first question that is asked by the model when a new record is forecasted. Depending on the answer to that question, the model traces a path to a node in the next layer, which in itself also has a new question, and the whole process is repeated until the model reaches a node that has no more connections, i.e., *Leaf* nodes, and the value of that *Leaf* represents the final answer of the model to the received record. A *DT* model constructs its structure graph based on the records of the series of data given as input, where the *Root* and the internal nodes, as well as their conditions for comparison are built with the *features* present in the *dataset*, and the *Leafs* and their values are defined with the *target* label of the input data. For a better understanding, Figure 2.8 shows a simple example of how a *DT* works. A briefly definition of *DT* is offer by Myles et al. (2004), and says that a *DT* is a hierarchical model composed of discriminant functions, or decision rules, that are applied recursively to partition the *feature* space of a *dataset* into pure, single class subspaces.

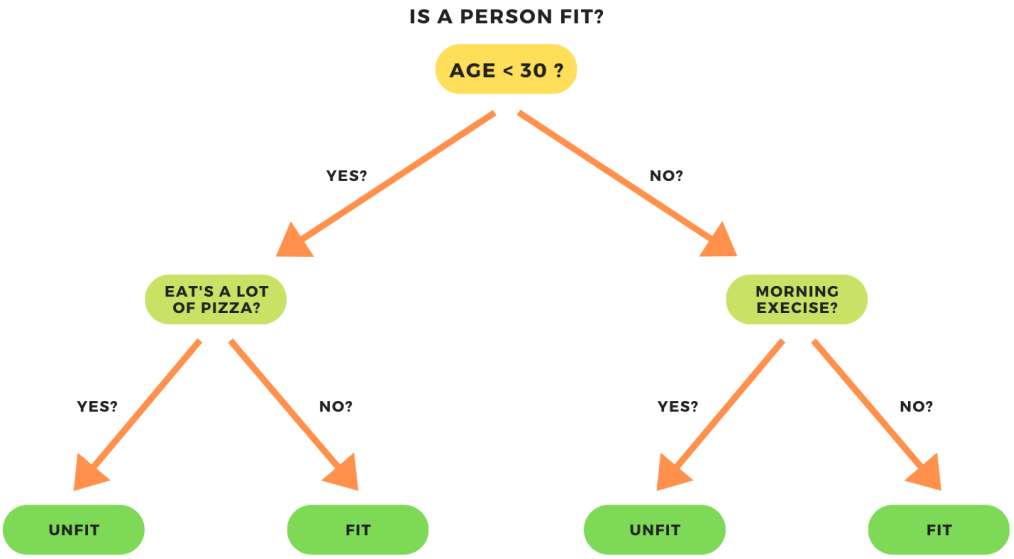


Figure 2.8: Decision Tree Operation.

K-Nearest Neighbours

The models that use *K-Nearest Neighbours (KNN)* technique are usually recognized as lazy models, as they do not have a training phase like others that are explained in this subsection. Since all *entry* of the *dataset* are place sequentially in a *n* dimensional space (*n* is equal to the number of *features* of the series of input data), and the process of finding a pattern or a function that will help

to classify new records is ignored. The *K* value is an argument that is given as an input to the model and represents the number of neighbours that are in the vicinity of the new object that we want to classify. Each new object is positioned in the sample space in question, then a circumference with the smallest possible distance is drawn so that *K* neighbours are found, and finally, the new case study is classified according to the class that represents the majority of that local neighbourhood. Figure 2.9 demonstrates how a *KNN* model works. A generic definition of this model was made by Hart (1968) and affirms that the Nearest neighbours rule assigns an unclassified sample to the same class as the nearest of *n* stored correctly classified samples. In other words, given a collection of *n* reference points, each classified by some external source, a new point is assigned to the same class as its nearest neighbour.

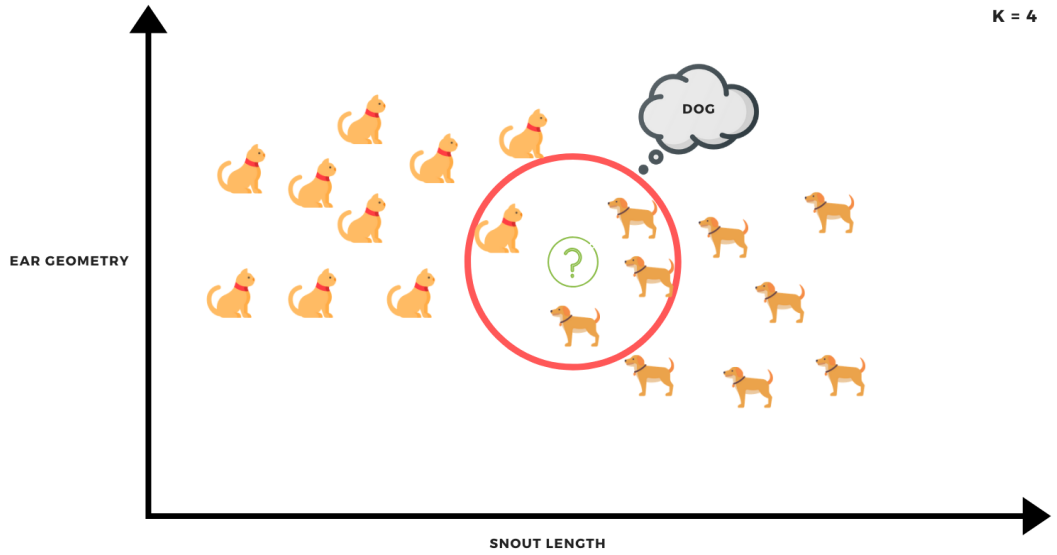


Figure 2.9: K-Nearest Neighbours Operation.

Support Vector Machines

The *Support Vector Machines (SVM)* is defined by Burbidge e Buxton (2001) as a training model for learning classification and regression rules from data, that best segregates that same data into two, or more classes. Imagine a two-dimensional plane, where each axis represents the range of possible values for a given characteristic of a given object of study. A *SVM* distributes all the study objects provided by the input data, according to the values of the respective characteristics that are being analyzed. After this phase is completed, the model in question tries to find a line that best separates the groups of objects that are being observed. This line is built with the help of the points that are closest to the group opposite to its own, and so the line is drawn so that all points belonging to it are at an equal distance to those two points in question, i.e., it is their mediator.

These points are also called *Support Vectors*. To make the reading of this section more simple and easy to understand, an example of a two-dimensional space was used, but in reality, *SVM* models are applied to spaces with *n* dimensions, where *n* is equal to the number of *features* present in the *dataset* given as input. Therefore instead of lines, *SVM* models try to design planes to separate the desired groups. Using the same example used in the section that explains the *KNN* model, Figure 2.10 schematize a simple example of the learning process associated with *SVM*.

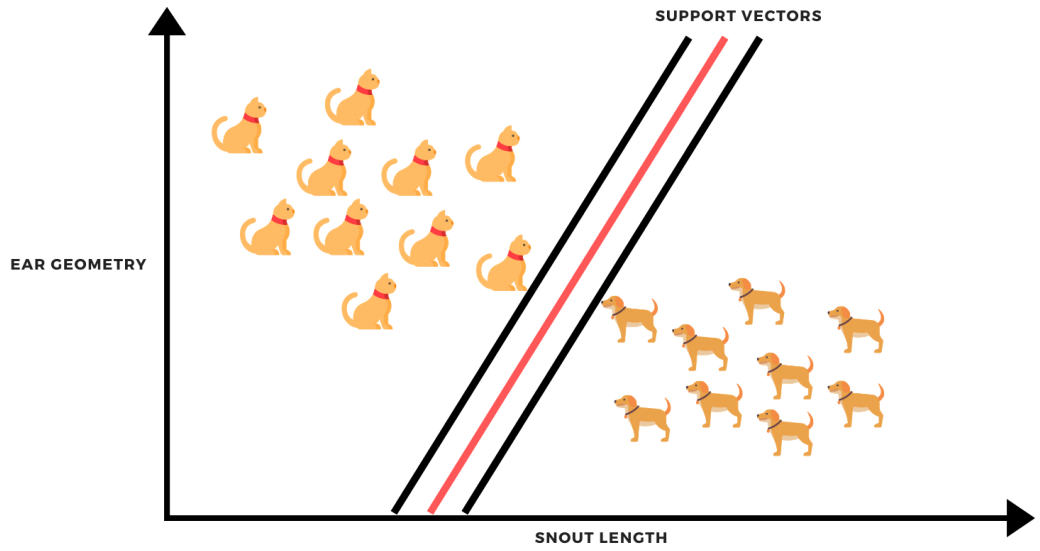


Figure 2.10: Support Vector Machine Operation.

Naïve Bayes

In this segment of the document stays the description of the *ML* models inspired in the Theorem of Bayes. Rish et al. (2001) affirms that *Naïve Bayes (NB)* classifiers assign the most likely class to a given example described by its *feature* vector, with this been said, these models make their predictions with help of the probabilities associated with the data, and this is why they are called "Naïve". If the reader looks closely at Table 2.1, it is possible to see an example of a data series in which each *entry* represents an answer to the question "Let's play football?".

For a new instance like for example, a sunny day, with a cool temperature, high humidity and strong winds. These classifiers begin by calculating the probability of each possible answer in the *dataset*, which in this case is the probability of going to play football and the probability of not going to play. For this example these values are:

$$P(Play = Yes) = \frac{9}{14} \tag{2.3}$$

$$P(Play = No) = \frac{5}{14} \tag{2.4}$$

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Table 2.1: An example of a *dataset* about "Playing football?".

The next step is to find the probabilities of each *feature* value present in this new *entry*, given the types of possible answers. For a positive answer these are the probabilities of each *feature*:

$$\begin{aligned} P(Outlook = Sunny|Play = Yes) &= \frac{2}{9} \\ P(Temperature = Cool|Play = Yes) &= \frac{3}{9} \\ P(Humidity = High|Play = Yes) &= \frac{3}{9} \\ P(Wind = Strong|Play = Yes) &= \frac{3}{9} \end{aligned} \tag{2.5}$$

For a negative answer these are the probabilities of each *feature*:

$$\begin{aligned} P(Outlook = Sunny|Play = No) &= \frac{3}{5} \\ P(Temperature = Cool|Play = No) &= \frac{1}{5} \\ P(Humidity = High|Play = No) &= \frac{4}{5} \\ P(Wind = Strong|Play = No) &= \frac{3}{5} \end{aligned} \tag{2.6}$$

Then for each answer, this model multiplies all probabilities of each given *feature* by the respective answer. For the affirmative answer it is the multiplication of the value found in (2.3) and the values in (2.5):

$$P(X|Play = Yes) * P(Play = Yes) = \frac{2}{9} * \frac{3}{9} * \frac{3}{9} * \frac{3}{9} * \frac{9}{14} = 0.0053 \tag{2.7}$$

The same is performed for the calculation for the opposite answer:

$$P(X|Play = No) * P(Play = No) = \frac{3}{5} * \frac{1}{5} * \frac{4}{5} * \frac{3}{5} * \frac{5}{14} = 0.0206 \tag{2.8}$$

To obtain the final results, the values normalized must be divided by the evidence of happening this *entry*, and that is calculated by:

$$P(X) = \frac{5}{14} * \frac{4}{14} * \frac{7}{14} * \frac{6}{14} = 0.02186 \tag{2.9}$$

To normalize the values found in (2.7) and (2.8) both are divided by the value calculated in (2.9):

$$P(Play = Yes|X) = \frac{0.0053}{0.02186} = 0.2424 \tag{2.10}$$

$$P(Play = No|X) = \frac{0.0206}{0.02186} = 0.9421 \tag{2.11}$$

The final decision is made depending on the highest value of (2.10) and (2.11), and in this case is the probability of not going to play knowing that weather conditions.

K-Means Clustering

The models based on *K-Means Clustering (K-Means)* have a simple and light segmentation process in comparison to other models that can be used in *UL* paradigm. Navaz et al. (2018) defines *K-Means* as a model based on an algorithm that surrogates between two noteworthy advances, passing on perceptions to groups and processing cluster focuses until the point when a ceasing standard is satisfied. For a quick understanding, it will be focused on a bi-dimensional problem, but it can also be used in an n-dimensional approach. These models, after distributing the data by a bi-dimensional space, place two centroids randomly amongst the data. The number of centroids varies according to the number of groups, or *clusters*, we want to find, but for a better understanding, we will continue with just two centroids. The chosen number is also known as *K argument*, like in a *KNN* model. Then a line is drawn that best represents the mediator between these two centroids, and so the first two clusters are constructed, with all data being categorized according to the side of the mediator on which they are located. Finally, the centroids are repositioned so that they are in the centre of the clusters previously built. These sequential phases are executed an arbitrary number of times, but it is normal to choose a number that assures that the position of the clusters stabilizes. In other words, the aim of the *K-Means* model is to divide M points in N dimensions into K clusters so that the within-cluster sum of squares is minimized, Hartigan e Wong (1979). For a simple visualization, Figure 2.11 illustrates a example iteration of a *K-Means* model.

Agglomerative Clustering

If we analyze the process behind a *K-Means* model, it is clear that it is a top-down method, i.e., the clusters are defined immediately through sets of points in the *dataset*. But now the reader may be asking if there are bottom-up processes, and the correct answer to that question is yes. A clear example of an *UL* model that uses a bottom-up method is *Agglomerative Clustering (AC)*, or

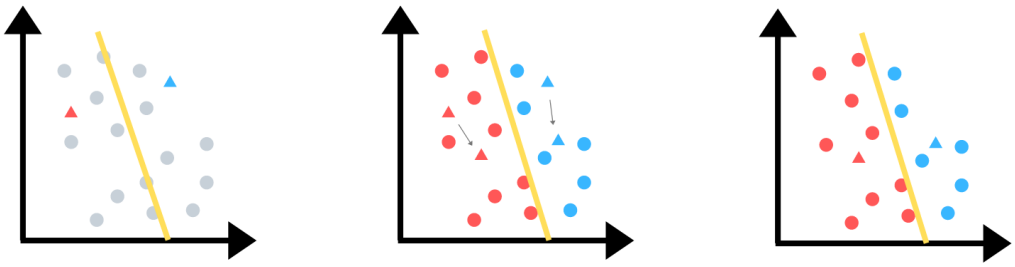


Figure 2.11: K-Means Clustering Operation.

also known as *Hierarchical Clustering*. As affirmed by Müllner (2011), *AC* schemes start from the partition of the *dataset* into singleton nodes and merge step by step the current pair of mutually closest nodes into a new node until there is one final node left, which comprises the entire *dataset*. The *AC* process begins by establishing each point in the *dataset* as a cluster. Afterwards, the closest pair of clusters is agglomerated in a single new cluster, and that is the main reason why this model is called "Agglomerative". This step is repeated iteratively until one single cluster is reached or a certain *threshold* value is surpassed. The distance is calculated through various ways, it could be *Euclidean*, *Manhattan*, among others, but to facilitate the understanding of this model, the Euclidean distance is assumed. The *threshold* value can be used in several different ways, but it can be assumed in this case, for example as the number of clusters that is intended. *AC* is a most flexible method and it is also used for clustering the web data in web usage mining, there are do not need the number of clusters as an input, as is evidenced by Katariya e Aluvalu (2014). Figure 2.12 shows a typical example of a graph representative of an abstract *dataset*, where each record is symbolized as a point, along with a possible *Dendrogram* that may illustrate the clustering process behind a *AC* model.

There are two more *UL* models used in this research, but their core is based in *Hierarchical Clustering* as *AC*, but the differences between them are highly technical, and to maintain the reading of these introduction notes simple, they are not explained in this chapter. The names of these models are *Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)* and *Spectral Clustering (SC)*, and their process will be introduced along with the parameters that were explored in a further chapter.

Label Propagation

Regarding to the *SSL* paradigm, in this work, namely the *Label Propagation (LP)*, and the *Label Spreading (LS)*. It should be noted that both algorithms are quite similar, and differ only in the calculation of the similarity matrix, so only *LP* will be introduced in this same section, and the reader should assume that the operation behind *LS* is identical. Despite in section 2.1.1.3, it is clearly expressed that a *SSL* model tries to predict the unlabelled *entries* of the *dataset*. In reality, *SSL*

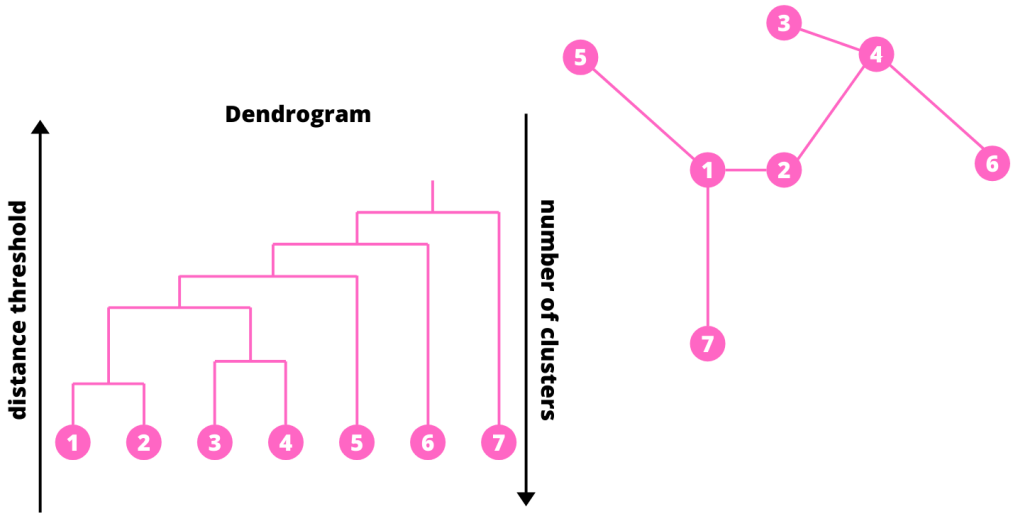


Figure 2.12: Agglomerative Clustering Operation.

models label the unknown points by comparison with the labelled *entries*. The main reason why this happens is that two training processes are too expensive in terms of time and computational resources, and might give some *bias* in the training of the entire *dataset*. Karasuyama e Mamitsuka (2013) define *LP* as one of the state-of-the-art methods for semi-supervised learning, which estimates labels by propagating label information through a graph. *LP* assumes that data points (nodes) connected in a graph should have similar labels. Consequently, the label estimation heavily depends on edge weights in a graph which represents the similarity of each node pair. In other words, imagine a collection of data, as a graph, and each *entry* in our *dataset* is a point in that graph. The edges that connect the nodes of the graph are drawn with the help of the similarity matrix, which is calculated differently if you are talking about *LP* or *LS*. Then after the connections are established, the predicting of the unknown labelled points is carried out by the propagation of the points that have known label. Known labelled points propagate their label to the nearest unknown labelled points, this process is also denominated as *Pseudo-Labeling*. Afterwards, the pseudo-known labelled points propagate their label to the nearest unknown labelled points, and the process is then repeated until all points have a label or a certain *threshold* value is surpassed. In cases where a non-labelled *entry* is equally connected to two distinct groups, the winning label is the one that the point is nearer, and this notion of distance is assigned by the weights of the edges that are constructed with the formerly mentioned similarity matrix. In Figure 2.13 can be found a simple example of the *Pseudo-Labeling* process, where the *dataset* is related to valid and invalid credit card numbers. *LP*, for Iscen et al. (2019), is a graph-based method, and the graph is constructed exploiting the embeddings obtained by the classification network itself. Thus, the proposed method alternates between two steps. First, the network is trained from labelled and pseudo-labelled data.

The second step uses the embeddings of the network trained in the previous step to construct the nearest neighbour graph. Label propagation is then used to infer pseudo-labels for unlabelled images, as well as a certainty score per image and per class. Training is performed on all data, using certainty-based weights.

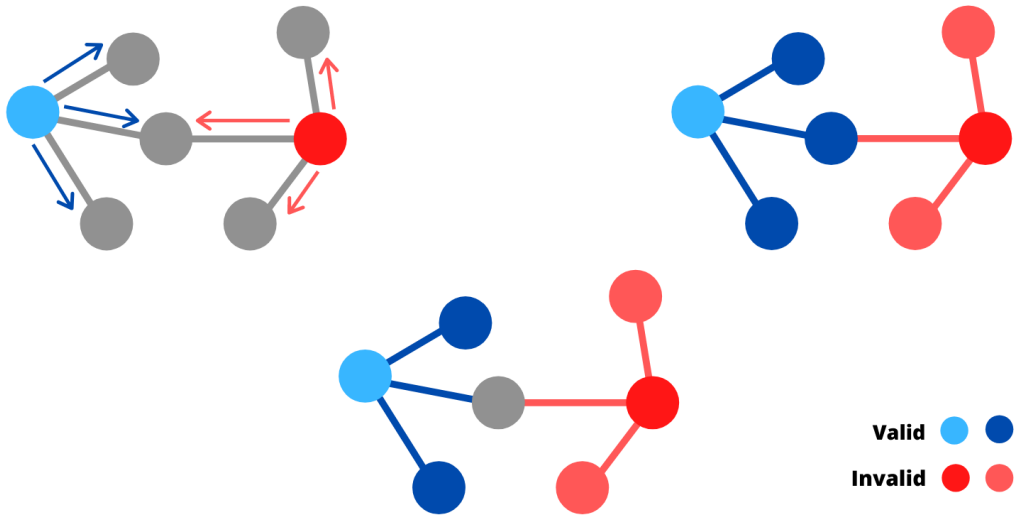


Figure 2.13: Label Propagation Operation.

Artificial Neural Networks

An *Artificial Neural Network (ANN)* is a model inspired by the structure of the brain that is well suited for complicated tasks, just like affirmed by Wang et al. (2010) and are the main focus in the *DL* paradigm. These models try to copy the thinking process of the human brain, by building a structure composed of three main layers. The first one is the input layer, this layer is constituted by an arbitrary number of nodes, to which the input *dataset* is provided. After this layer comes the several hidden layers and this number of layers can vary depending on the problem. Last but not least, comes the output layer, which symbolizes all possible answers to the problem itself, such as a node for each polygon shape. Each node, or *neuron* of a layer is connected to other nodes of the next layer, these connections are made by channels that possess a numerical value, also know as *weights*. The data that was provided to the input layer are multiplied by the *weights*, and the sum of this score is provided to the next layer. Which node in the hidden layers possesses a numerical value that is called *bias*, that is the sum to the value that was calculated in the previous layers. This result is provided to a *threshold* function called *Activation Function* that transforms the new value, according to the function equation, and then the node will transfer its information to the next connected nodes. These calculations are repeated throughout the neuronal network until reaching

the last layer and the name of this process is known as *Forward Propagation*. During the training of an *ANN*, the predicted output is compared to the actual output to realize the error in prediction, then this information is transmitted backward through the network, making adjustments to the weights, this process is named as *Back Propagation*. Figure 2.14 helps to better understand this whole process, the same image was tacked form the Simplilearn (2019) video. Just like Yegnanarayana (2009) describes, an *ANN* is a set of processing units when assembled in a closely interconnected network, offers a surprisingly rich structure exhibiting some *features* of the biological neural network.

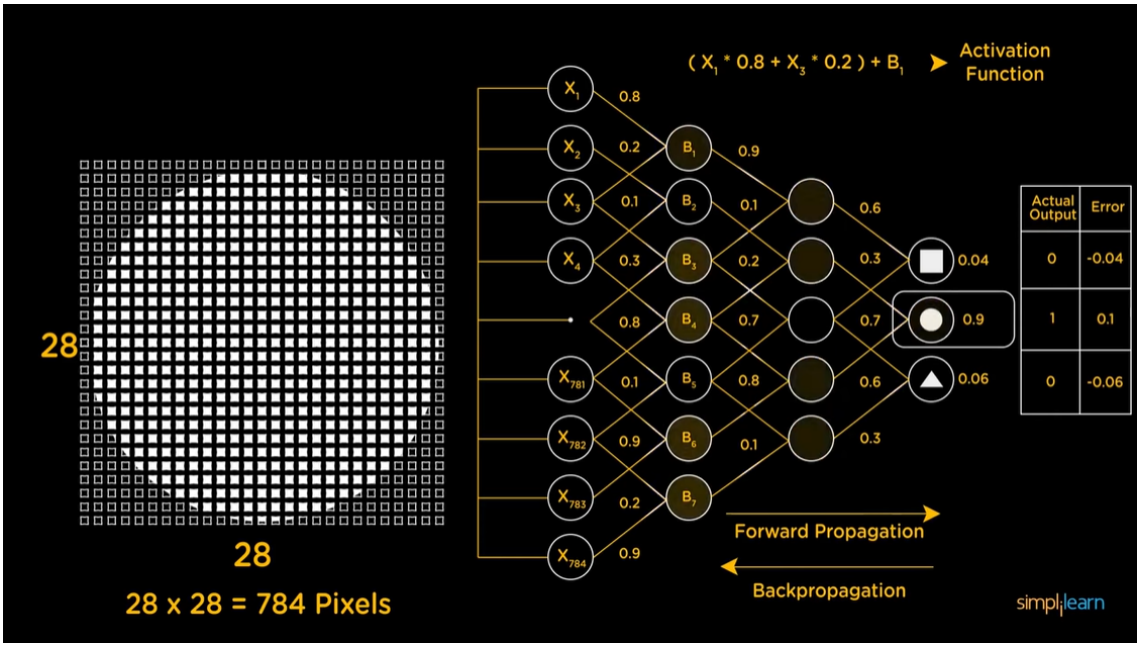


Figure 2.14: Artificial Neural Network Operation.

Random Forest

The next models that will be presented, are usually also known as meta-models. And the reason for this is that these models are not able to obtain or simulate human intelligence by themselves. Instead, they use other models as a basis and try to increase their performance in an attempt to obtain better results. We will start by introducing a meta-model that only uses a single type of base, which is the *Random Forest (RF)*. For Kulkarni e Sinha (2012) this model is an ensemble *SL* technique. Based on bagging and random *feature* selection, with several decision trees that are generated and majority voting is taken for classification. Therefore, a *RF*, as the name implies, is an aleatory collection of *DT*, the same trees that are introduced in section 2.1.2.3. The learning process in this type of cases begins with the creation of several trees with different parameters. Then, each tree trains with a random portion of the *dataset*, and this sample has all the *features* of the original input. When a new case is provided for this collection of models, each tree establishes

a forecast according to its training and its parameters. Finally, the final decision is made by the majority of votes for each tree, if it is a classification problem, or by the average vote of each tree, if it is a regression problem. An interesting particularity of these meta estimators, like is pointed by Khan et al. (2010), is that *RF* is an ensemble classifier having a quick training phase and a very high generalization accuracy. For a better understanding of ht process behind this operation, Figure 2.15 may give help.

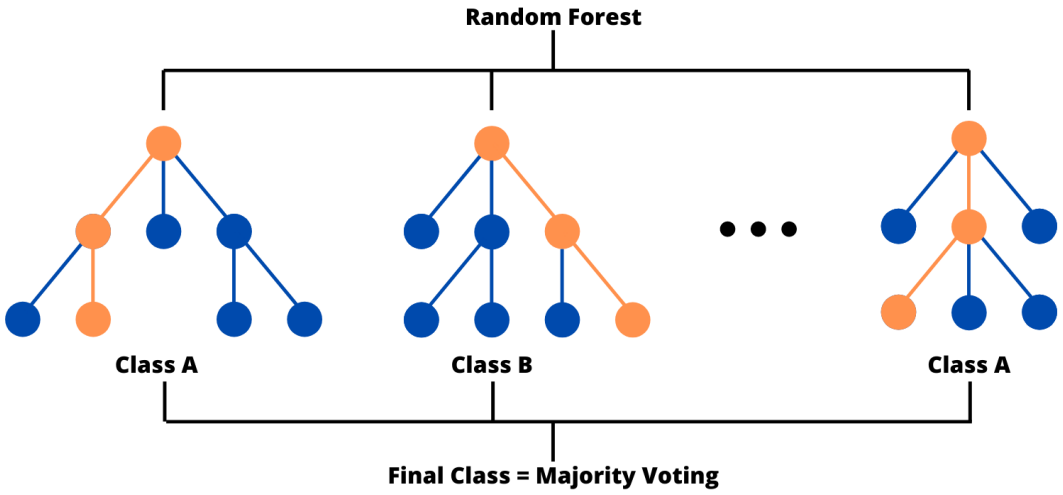


Figure 2.15: Random Forest Operation.

Adaptive Boosting

From a specific meta-model we will move on to a more generic one, the *Adaptive Boosting (AdaBoost)*. In a short definition gave by Alfaro et al. (2008), *AdaBoost* is a novel ensemble learning algorithm that constructs its base classifiers in sequence using different versions of the training *dataset*. Imagine a set of records of cancer exams. Each one indicates whether the patient has a tumour or not, along with several morphological characteristics of that same patient. The *AdaBoost* uses an arbitrary base estimator to train with the *dataset*. Then a series of copies of that base model is created, and consequently, they are trained with the same *dataset*, but this time, the records in which the base model was wrong in the forecast become more important in the new training phase, and the records in which the base model was correct, have a lower weight. This operation is repeated an arbitrary number of times, and at the end, a final classifier is built based on the joining of the weights of each record during all iterations performed. Figure 2.16 demonstrates a small execution of the cancer exams example, in a three iteration process. In the first execution, the copies of the base model, predict as positive exams all the records that are presented in the green area,

and negative exams those that are in the blue area. Notice that in the second iteration the positive records that were incorrectly predicted, have more weight in the new training phase, represented by a bigger "plus" symbol. In this phase, the models tend to predict all the region as positive in a way that all incorrect answers are covered. For the last iteration, the weight of the incorrect negative predictions increases, and the same methodology is repeated, so the more weighted *entries* are correctly predicted. The final model is a combination of all trained models and the probability of making a correct prediction for a new exam in the future increases. This example demonstrates also that, as Gao e Gao (2010) refer, *AdaBoost* is known as an effective method for improving the performance of base classifiers both theoretically and empirically.

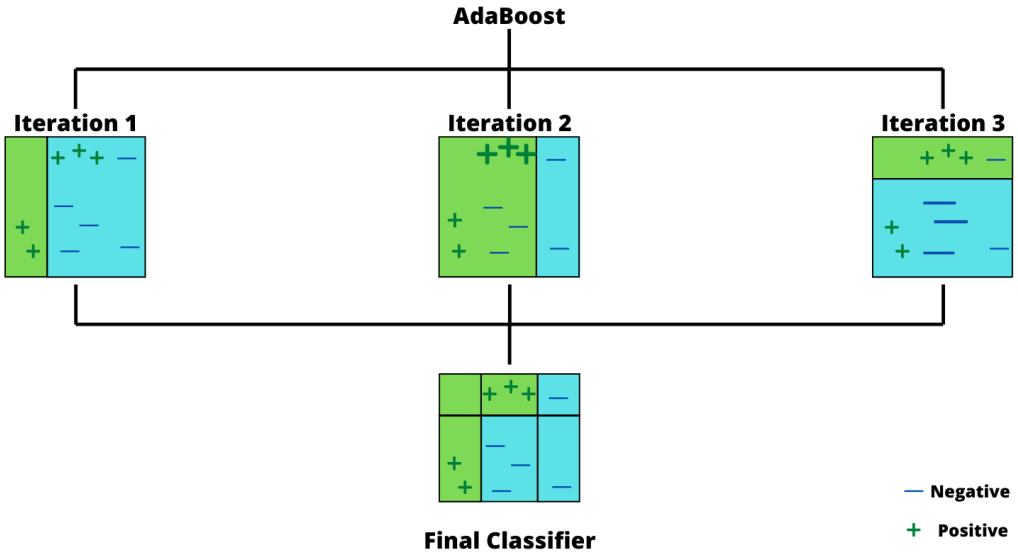


Figure 2.16: Ada Boost Operation.

Gradient Boosting

Finally, the last model introduced will be *Gradient Boosting (GB)*. Like a *RF*, a *GB* model is commonly used with *DT*. However, instead of creating a set of weak models, in a hope of obtaining a strong combination, *GB* tries to improve the performance of a base model, in a similar way like a *AdaBoost* model behaves. Zhang et al. (2019) defines *GB* as one of a set of techniques which is able to recursively fit a weak learner to the residual so as to improve model performance with a gradually increasing number of iterations. This meta-model starts with sampling the original *dataset*, and this sampling in addition to not having all the records from the original *dataset*, it may also not have all the *features* provided as input either. Then the model that is being adjusted performs a training phase with this sample. This training phase culminates in a set of residuals that indicate how well the model fits with the sample prediction. From now on, this process will be repeated,

but the residuals are used instead of the data sample during the training. In this way, the model tries to focus more on what it misses, and the process only stops when a number of iterations is reached or the residuals become small enough. The prediction of new cases of study is supplied by a weighted sum of all the iterations, according to their residual values during the training process. Xu et al. (2017) describe this tuning process as the construction of additive regression models by sequentially fitting a simple parameterized function (base learner) to current pseudo-residuals by least-squares at each iteration. The pseudo-residuals are the gradient of the loss function being minimized. It is worth noting that both the approximation accuracy and execution speed of *GB* can be substantially improved by incorporating randomization into the procedure. Figure 2.17 shows the course of the process involved in a *GB*.

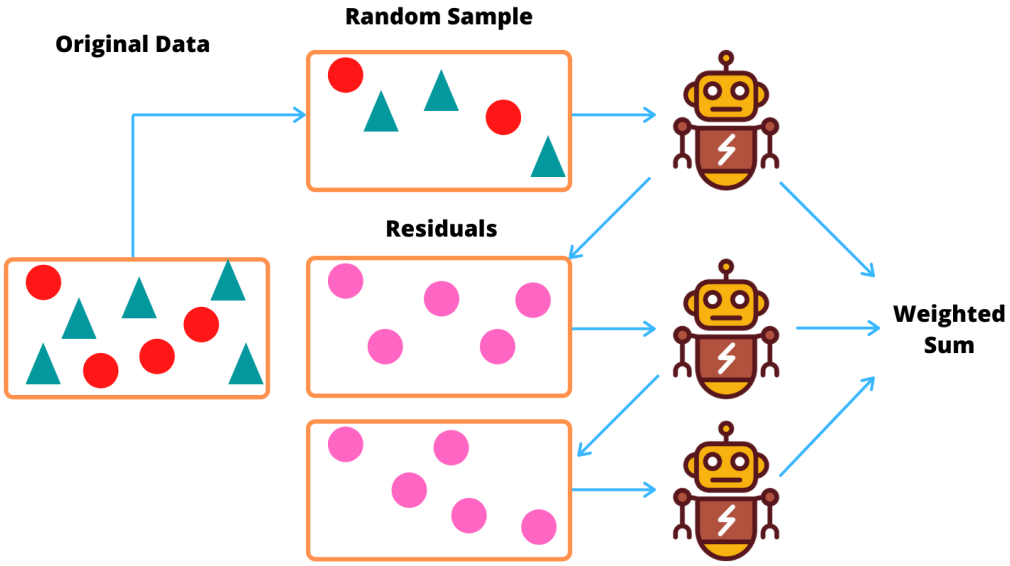


Figure 2.17: Gradient Boost Operation.

2.1.3 Learning Metrics

Usually, a modelling process is time-consuming, not only due to the training process, which can lead to a costly computational process, but also because it requires a lot of research and comparison between various types of models, such as those previously described. Thus, the reader may ask: "How do I know that a model is good?", or "Of all the models that I trained, which one is the best?". To answer these types of questions, exist several metrics that can qualify a model with regard to its quality, most of which come from the confusion matrix. A *Confusion Matrix* is defined by Parker (2001), as a matrix in which the actual class or a datum under test is represented by the matrix row, and the confusion matrix column represents the classification of that particular datum. The element M[*i*][*j*] gives the number of times that a class *i* object was assigned to class *j*. The diagonal

elements indicate correct classifications, and the other elements represent incorrect classifications. The most popular *Learning Metric* is probably the *Accuracy*, and it is defined as the ratio of the number of correctly classified items to the total number of items (Junker et al. (1999)). Relying solely on this metric can cause problems. Imagine that an *ML* model was developed to identify cases of COVID-19, and that this model was successful in 90% of cases. An *Accuracy* of 90% can be considered an excellent result, but the remaining 10% of the cases, the model had a tendency to predict cases as being negative, but in fact, they are positive cases. In real life, this could have catastrophic consequences, as there would be a percentage of patients subjected to COVID-19 tests who were considered to be healthy. Several chains of transmission could emerge, and the number of people infected globally could increase. To circumvent this type of situations, two other metrics are used, which are *Recall* and *Precision*. The *Recall* is computed by Equation (2.13), and like *Accuracy*, as long as this metric is closer to 100%, the better the performance of the model, it means that the model predicts false negatives less often. The *Precision* is calculated by Equation (2.14) and in the same way as *Accuracy* and *Recall*, as long as this metric is closer to 100%, the better the performance of the model, it means that the model predicts false positives less often. A metric that tries to combine previous two, in an attempt to get the best of both, is the *F1-Score*, as calculated by the formula indicated in (2.15). Table 2.2 represents a illustrative example of a confusion matrix.

		Actual Class	
		Cat	Dog
Predicted Class	Cat	67	4
	Dog	1	84

Table 2.2: An example of a Confusion Matrix.

$$Accuracy = \frac{\text{Correct Predictions}}{\text{Total Population}} \tag{2.12}$$

$$\text{Recall} = \frac{\text{Correct Predictions of Category X}}{\text{Total Population of Category X}} \tag{2.13}$$

$$\text{Precision} = \frac{\text{Correct Predictions of Category X}}{\text{Total Predictions of Category X}} \tag{2.14}$$

$$\text{F1-Score} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.15}$$

For the given example given in Table 2.2, the *Accuracy* of the model is $\frac{67+84}{67+4+1+84} = 0.97$. For the class *Dog*, the *Recall* is $\frac{84}{4+84} = 0.95$, the *Precision* is $\frac{84}{1+84} = 0.99$, and the *F1-Score* is $2 * \frac{0.99*0.95}{0.99+0.95} = 0.97$.

2.1.4 Cross Validation

Nowadays, the best way to evaluate the performance of a student at school throughout the year is through a series of exams. Usually, these exams are a set of exercises, which the student can get the right or a wrong resolution. The grade the student obtains is the representation of the ratio between right and wrong exercises on the exam. The reader can make an analogy, in which an *ML* model can be a student in school since it has the same great objective, to learn. Following this metaphor, for a model, each record in the data collection is like an exercise for the student. Just as a student needs to complete many tasks to study for an exam, an *ML* model needs to train with many *entries* from a *dataset* to get the best possible learning metrics. It is not correct to provide a student with the exercises that will come out on the test the day before, as the student will memorize the resolutions. Even if the student gets the highest score on the test, this result can be misleading, because it is not sure whether the student will have the same performance when facing different tasks. Therefore, it is not a good practice to evaluate a model with exercises that it already knows how to do, as such, executing a training process with all the records present in the *dataset* becomes obsolete. Simply dividing the data collection into two parts, one having the set of *entries* destined for the training phase, and the other having the set of records destined for the evaluation phase, may also not be sufficient. This division could leave *entries* that would bring great benefit to the training phase, in the input group for the evaluation phase. To solve this situation, a model evaluation technique, called Cross-Validation, appears. Dai (2013) affirms that the cross-validation technique is a standard tool in statistics which provides an appealing guiding principle to choose, within a set of candidate model structures, the ''best'' one according to a certain criterion. The Cross-Validation technique starts with a division of the *dataset* into N folds (N is an arbitrary number provided as input). In the first iteration, the model trains with N-1 folds of the *dataset* and obtains a score with the missing block. The evaluation phase generates an evaluation parameter, such as Accuracy, Recall or F1-Score, which indicates the result obtained in this iteration. This process repeats N-1 times, and in each interaction, the blocks used for training and evaluation vary, as Figure 2.18 shows. In the end, this process calculates the average of the scores attained in each iteration, and the result achieved represents the learning metric that the model obtained with the data collection.

2.1.5 Overfitting and Underfitting

These metrics are not always good enough to indicate if one model is better than another, there are cases in which despite these models having high accuracy, may not be the most suitable example for the problem, let us see the following cases:

- **Overfitting**: is the problem that a very specific model is generated while it is obvious that the *dataset* only holds example behavior, i.e., the model explains the particular sample *dataset*

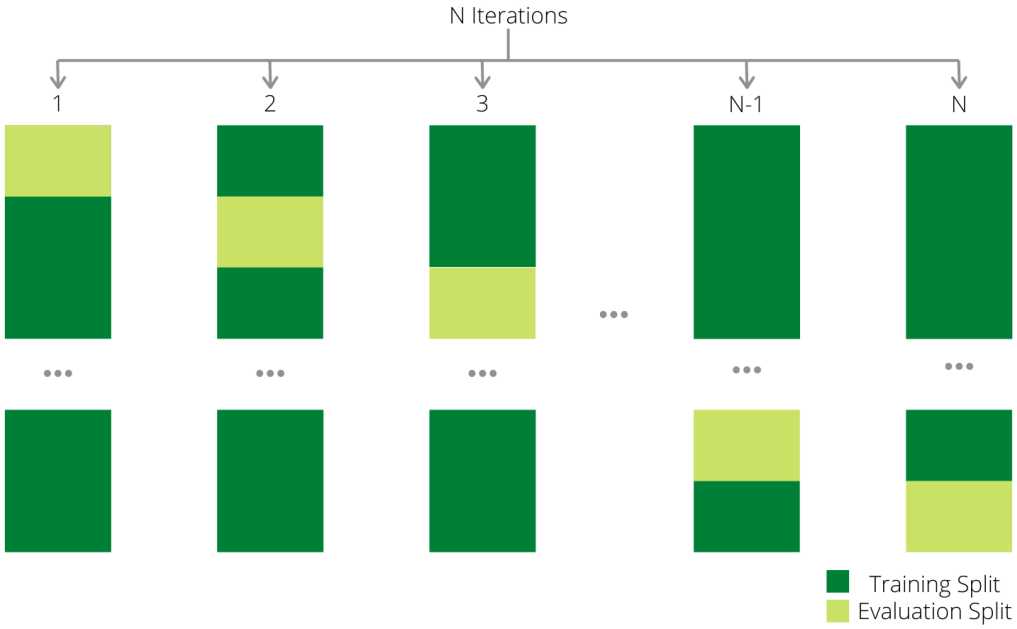


Figure 2.18: Routine of the Cross-Validation technique.

but a next sample *dataset* of the same process may produce a completely different process model (Van der Aalst et al. (2010)). The most basic solution to solve this type of problem is probably to acquire more data with more general information. However, when this is not possible, a good way to get around this obstacle is to carry out several training and validation tests, with different sub-parts of the *dataset*;

- **Underfitting**: is the problem that the model over-generalizes the example behavior in the *dataset*, i.e., the model allows for very different behaviors from what was seen in the *dataset*, Van der Aalst et al. (2010). Symmetrically to the previous difficulty, this is resolved by acquiring data that can present more unusual information, so models, during training, can develop strategies to deal with *outliers*.

2.2 LITERATURE REVIEW

Perhaps the most important definition to give for a bigger understanding of this thesis is the meaning of boredom, which is according to Mikulas e Vodanovich (1993) the state of relatively low arousal and dissatisfaction, which is attributed to an inadequately stimulation situation. With this notion in mind, and with the willing to suppress this feeling, many studies were conducted in this area, to detect this emotion, and conjecture possible approaches to gain advantage from the opportunity that this feeling may give. These are investigations that have inferred standards for deducing the level of boredom on various technology channels, such as mobile devices and computers, or the detection of other feelings in smartphone users. Traditionally, the most common

method for detecting boredom is through speech analysis, facial expressions, and text analysis (Bixler e D'Mello (2013)). Other studies use physiological measures such as body temperature, heart rate intensity, respiration, among many others (Picard et al. (2001)).

Bixler e D'Mello (2013) developed a *DT* model optimize with the *RF* algorithm that lead to an accuracy of 87%. Along with this heavy model, it was developed a more lightly one, that was a *NB* model with an accuracy of 82%, and these two different models were capable of predicting the level of boredom and engagement of a person, during writing periods at a computer, with the help of a keystroke analysis, task appraisals, and stable traits. Picard et al. (2001) constructed o program based in a *KNN* model to identify various human feelings, including boredom, with an accuracy of 81%. These two approaches require specific sensors that must be performed over a large continuous time frame, which represents a major limitation, as the required configuration may not be available when the boredom appears, as was pointed by Plarre et al. (2011) in their study. In addition to boredom, studies were also carried out with the goal of detecting other feelings, such as the level of stress, in a research that was carried out by Plarre et al. (2011). In this research, the level of stress was measured by sensors of various vital signs in the clothing of the participants, but even with this obstacle, was achieved an accuracy of 90% with a *DT* model, however, this study also points this same disadvantage, like investigations that were indicated before.

To overcome this limitation, some studies show that mobile phone sensors are a viable way to monitor usage habits by providing information about the emotional state of the owners, including boredom. For example, Pielot et al. (2015) proposed an *ML* model that can predict the boredom of smartphone owners based on *features* related to recent communication, usage intensity, time of day and demographic information. This model is a *DT* that uses the *RF* process for improvement, and it was obtained an accuracy of 82.9%. Guo et al. (2009) has discovered, with the help of a *SVM* model, that Web interaction events such as mouse movements, number of clicks, and number of times scrolling on a page allow the prediction of whether a person is willing to be distracted by something other than your main task, and may indicate a high level of boredom. Besides, it is important to highlight other related studies, but more focused on predicting more than a human feeling, with the help of smartphone usage patterns as well.

According to Bogomolov et al. (2014), the stress level can be reliably recognized based on behavioural metrics derived from mobile phone activity and other indicators such as weather conditions and personality traits, this result was attained again by a *DT* model, with an accuracy of 72.28%. Sano e Picard (2013) have reported good results for stress recognition using a combination of mobile device and clothing-associated sensor *features*, despite some restrictions, such as a limited number of subjects and acquisition of respective data, the *SVM* and *KNN* (N=14) models showed over 75% accuracy in predicting this kind of cases. LiKamWa et al. (2013) concentrated their research on building a statistical model based in *Lin-Reg*, capable of inferring the feelings of a person, on a given day. The final model attained relatively high performance, but after a two months train-

ing with personalized data that same performance registered an even higher value. The data was constructed based on communication history and application usage patterns. Another remarkable research conducted by Bogomolov et al. (2013) concluded that it is possible to automatically recognize the happiness of an individual daily life, using a large set of indicators obtained with the help of smartphones, like for example communication data, proximity sensors, Bluetooth connection, weather data, and personal characteristics. The results of this investigation show up to 80.81% of accuracy, using a *DT* model.

Seo et al. (2019a) developed a robust *ANN* that was able to classify the boredom of a person, using data from electroencephalography and galvanic skin response exams, the accuracy of this model was 79.98%. A very similar study was done again by Seo et al. (2019b), but this time only data from electroencephalography exams were used, obtaining greater precision, with a value of 86.73%, but using a *KNN* model. In this same area, Kim et al. (2018) proved the existence of a relationship between electroencephalography examinations and eye behaviour during the detection of annoyance. This team will try to develop in the future *ML* models that can achieve good results in predicting boredom, using this type of relationship. An investigation was conducted by Allen et al. (2016), on boredom and engagement during writing, with the participants volunteering to write an essay on a certain topic. During the essays, various types of information were collected, such as videos of facial expressions, screenshots, textual analysis, and various indexes of Keystrokes. The models used in this research are of an advanced *Lin-Reg* type, obtaining an accuracy of 77.3% in relation to boredom predictions during the writing period.

In the area of mental health, Torous et al. (2018) conduct a study to develop an *ML* model capable of predicting suicide. It is a *Lin-Reg* model and obtained an accuracy of around 90%. The data of this research were collected through smartphone sensors, such as the GPS signal, and the signals from other connected devices. In the area of road safety, Johnson e Rajamani (2020) have developed a learning model that is capable of predicting whether someone is using their mobile device while driving. This model is a *SVM* and achieved an extraordinary accuracy of 100%. The data used in this study are relevant to sensors in smartphones such as the accelerometer, gyroscope and GPS signal. Pimenta et al. (2016) created a *ANN* model that was capable of predicting the level of fatigue of an individual by the interaction with a computer. That same interaction was recorded with the help of accelerometers placed on chairs and other peripherals, movement of the mouses, typing in the keyboards, audio excerpts from microphones and footage from video cameras. The mentioned model was able to predict correctly this level of fatigue in 81% of the cases of study.

2.3 SUMMARY

Throughout this chapter, it stays clear the fundamental notions and key concepts of the science behind *ML*. This is a science that tries to simulate human intelligence in machines, using the envi-

ronment that surrounds them. There are three major learning paradigms in this area of study, the *SL*, the *UL* and *RL*. In addition to these, there is another paradigm that is a hybrid solution between *SL* and *UL*, which is *SSL*. In any of these paradigms, it is possible to develop *DL* approaches. witch is a methodology working around *ANN* models but is mostly used in the *SL* paradigm. All of the areas have their own characteristic models, in the supervised area some examples are the *Lin-Reg*, the *Log-Reg*, the *DT*, the *KNN*, the *SVM* and the *NB* models. The unsupervised branch posses, for example, the *K-Means* and the *AC* models. The hybrid solution that is *SSL*, has models like *LP* and the *LS*, for example. Developing *ML* models is not the last step in the whole process, it is also possible to improve them so it can be obtained better results and better performances, and one way to achieve this is through meta-models. Examples of meta-models are the *RF* and the *GB*, specifically for *DT*, and the *AdaBoost* that can be applied to any type of model. To evaluate the operation of a model, there are several metrics, of which the most used are Accuracy, Recall, Precision and F1-Score. Great results do not always indicate great solutions to the problems that these models are trying to solve, as there are situations like *Overfitting* and *Underfitting* that can compromise the work achieved. This chapter concludes with the presentation of several projects carried out in the area that this dissertation aims at. Studies that have shown that it is possible to identify different emotions, such as boredom, stress, happiness or fatigue, through various technological channels, like through the use of a smartphone, by electroencephalography exams, body sensors, or by interacting with a computer.

3. MATERIALS AND METHODS

The following chapter depicts all physical and virtual materials, whether they are tools already developed or tools that had to be built to achieve certain objectives. Regardless of the type of approach that would be used in the future to achieve the main goal of this investigation, it was necessary to acquire data. That same set of data would have to be related to the use of smartphones, and to the usage behaviours that may indicate traces of boredom from the people who use them. Since, to date, there are no public *datasets* that can portray this type of information, it may be unanimous to admit that the best way to obtain it would be through a mobile application designed exclusively for this purpose.

3.1 DATA COLLECTION

For data collection purposes, we were required to design and conceive a mobile application, entitled as *BoredomApp*, that works as depicted in Figure 1.2. Whenever someone presses the smartphone screen unlocking mechanism, the application starts a measurement period that will last for 5 minutes. During this time, various physical and virtual metrics are measured, various sensor values are collected and various non-technological aspects are examined (it should be noted that this evaluation continues even after the screen is locked, and only ceases when it reaches the duration of 5 minutes). At the end of the referred time interval, the application triggers a notification, this notification leads to an activity that shows a question to the user. That question is "Are you bored?", like its shown in the Figure 3.1, and the user can answer on a scale from 0 to 100 using a slider button (not just a binary answer, like "yes I'm bored" or "no I'm not"). When the person using the smartphone thinks that has the most appropriate answer to rate the boredom status, it should be pressed the submit button, and when that happens a new record is created in our cloud database at *Firebase*, which is later used to extract the *dataset* for this investigation.

To better explain the architecture of this app, it is necessary to have the notion that it is divided into 5 parts. Each section can be composed of several smaller elements, with more specific and atomic functionalities, these 5 parcels are:

- **Interface Activities**: this layer represents all the user interfaces with the application. The number of these interfaces is 5. One interface for the question that is launched with the notification, the main interface that allows navigation to the others. One interface that explains all the sensors measured during the evaluation period, and two interfaces for the visualization of the data from these same sensors for the last evaluation periods;

- **Sensor Listeners**: group of modules responsible for measuring various physical aspects of real life. Using the hardware sensors present in a smartphone, these listeners can measure the brightness of the environment, gravity, magnetic fields, among others;

Figure 3.1: BoredomApp question interface, this question in Portuguese is "Está aborrecido?", the answer "Pouco" means "Little" and "Muito" means "Very", the button "Responder" it is used for submitting the final answer.

- **Broadcast Receivers**: set of components with the function to detect virtual events during the evaluation period. The events can be outgoing calls, incoming text messages, screen activation, Bluetooth connection status, among others;
- **Background Service**: this element of the application is primarily responsible for the management and temporal operation of the application. Prepares the launch of the notifications with the question "Are you bored?". It assembles all signals collected by the Broadcast Receivers, and the values form the Sensor Listeners, processes that information throughout the evaluation period and starts to create an *entry* of our *dataset*. Sends the record to the Database Handler as soon as the participant of this research answers the question. It is also the entity that handles the data provided to the Interface Activities;
- **Database Handler**: Part of the application that establishes the connection with the database. This Handler determines the structure of the records, along with the name and type of each *feature*. Send the *entries* to the database for the formation of the *dataset* used for this investigation.

Figure 3.2 shows a scheme of *BoredomApp* architecture, in an attempt to give the reader help to understand and memorize.

Is worth mentioning during the design of the *BoredomApp* an important decision was made. So that the application does not become too repetitive, and ends up being the main responsible for the

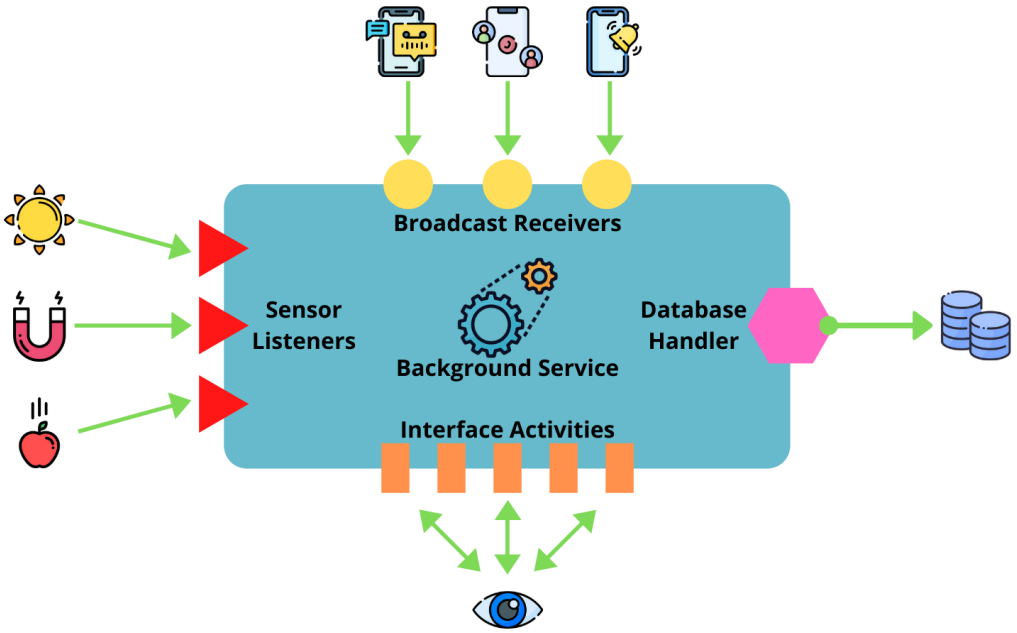


Figure 3.2: *BoredomApp* Architecture.

annoyance of the participant, the evaluation period runs for 5 minutes but does not start again in the following 30 minutes of activity. So the application will not consecutively begin the evaluation period during a car trip where GPS is required. The application will not start the evaluation period either if the smartphone user is not interacting with his device for a long time, thus avoiding to start the evaluation period during sleep time. This way, the application itself does not add a *bias* to the answers of the participants, thus ensuring the integrity of the *dataset*.

In an attempt to avoid missing values, at the beginning of the evaluation period, the Database Handler initialises the *entry* with all *features* with -999 as value. So if the mobile device does not have a particular sensor or a *Broadcast Receiver* is unable to perform its task, it is guaranteed that no record has a missing value. The reason for choosing this value is because no *feature* can naturally assume that value, henceforth, this number is considered as the null value.

BoredomApp obtained a relatively good adhesion, acquiring 50 installations on its first day. This initial influx allowed the application to reach fifth place in the trends leaderboard of *Play Store*, as shown in Figure 3.3. Even though the data collection period has ended, this application is still available on the *Play Store*.

3.2 DATA EXPLORATION

The *dataset* used for this research is composed of around 1500 records that were collected by the *BoredomApp* during 1st November of 2019 to 29th February of 2020. A population of 66 participants supplied all the *entries*, and no additional information was given, such as their age,

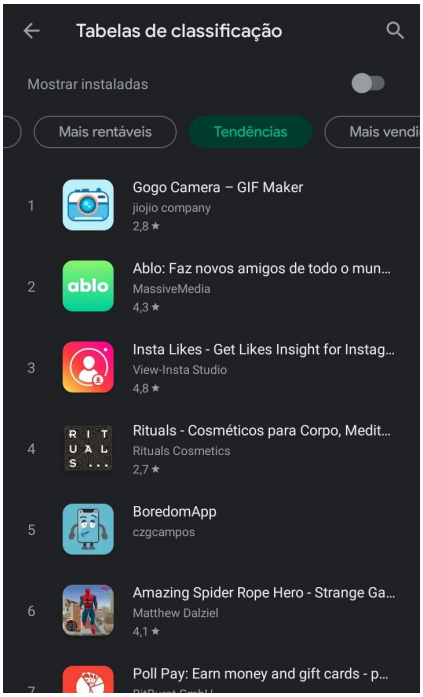


Figure 3.3: Trends leaderboard.

nationality, or profession. Each register describes the state of the smartphone when the participant answered the question. These are the *features* that the application measures:

- **Audio Jack**: it is a Boolean value that informs about the connection of the Audio Jack, at the time that the participant answered the question, where the value 0 means "not connected", and the value 1 indicates "it is connected";
- **Battery Level 1**: it is the value of the battery level of the mobile device, at the time that the evaluation period started;
- **Battery Level 2**: it is the value of the battery level of the mobile device, at the moment that the participant answered about the boring status;
- **Bluetooth Connection**: it is a flag that represents the state of the Bluetooth connection, at at the moment that the participant answered about the boring status. The value 0 means "Bluetooth not available", the value 1 indicates "Bluetooth off", 2 signifies "Bluetooth on and paired", and 3 implies "Bluetooth on but not paired";
- **First Date**: is the date of the start of the evaluation period, this date is composed by day, month, year, hour, minute, second and day of the week;
- **Second Date**: represents the moment that the participant answered about the boring status. This date has the same composition as the other date mentioned;

- **Flight Mode**: it is a Boolean value that informs about the Flight Mode, at the moment that the participant answered about the boring status, the value 0 means "it is off", and the value 1 implies "it is on";

- **Charging State**: it is a Boolean value that informs about the smartphone charging state, at the moment that the participant answered about the boring status, the value 0 means "it is not connected", and the value 1 implies "it is connected";

- **Orientation State**: it is a Boolean value that informs about the orientation of the mobile device, at the moment that the participant answered about the boring status, the value 0 means "it is not connected", and the value 1 implies "it is connected";

- **Current Notifications**: is the number of all notifications received during the evaluation period;

- **Chatting Notifications**: is the number of chatting notifications received during the evaluation period. The applications that contributed to this counting are Whatsapp, Messenger, Snapchat and others;

- **Social Notifications**: is the number of notifications related to Social Networks, received during the evaluation period. The applications that contributed to this counting are Facebook, Instagram, Twitter and others;

- **Other Notifications**: it is the number of the notifications received that do not fit in the other categories, during the evaluation period. applications like Gmail, Calander or Android System contributed to this counting;

- **Notifications Removed**: it is the number of removed notifications by the smartphone user, during the evaluation period;

- **Incoming Calls**: it is the number of incoming calls received during the evaluation period;

- **Outgoing Calls**: it is the number of outgoing calls effected by the participant during the evaluation period;

- **Accelerometer**: represents the value of the accelerometer sensor manufactured within the smartphone, at the time that the participant answered about the boring status;

- **Gravity Sensor**: represents the value of the gravity sensor manufactured within the smartphone, at the time that the participant answered about the boring status;

- **Gyroscope**: represents the value of the gyroscope sensor manufactured within the smartphone, at the time that the participant answered about the boring status;

- **Luminosity Sensor**: represents the value of the luminosity sensor manufactured within the smartphone, at the time that the participant answered about the boring status;
- **Magnetic Sensor**: represents the value of the magnetic sensor manufactured within the smartphone, at the time that the participant answered about the boring status;
- **Pressure Sensor**: represents the value of the pressure sensor manufactured within the smartphone, at the time that the participant answered about the boring status;
- **Proximity Sensor**: represents the value of the proximity sensor manufactured within the smartphone, at the time that the participant answered about the boring status;
- **Home Button Press**: it is the number of times that the participant pressed the home button during the evaluation period;
- **Recent Apps Button Press**: it is the number of times that the participant pressed the recent apps button during the evaluation period;
- **Ringer Mode**: it is a flag that represents the state of the ringer mode, at the time that the participant about the boring status, the value 0 means "silence", the value 1 implies "vibration only", and the 2 signifies "normal";
- **Screen Activations**: it is the number of screen activations during the evaluation period;
- **SMSes Received**: it is the number of text messages received, during the evaluation period;
- **Mobile Data Connection**: it is a Boolean value that informs if the cell phone had the mobile data connection on or off, at the time that the participant answered about the boring status. The value 0 means "it is off", and the value 1 implies "it is on";
- **Wi-fi Connection**: it is a Boolean value that informs if the cell phone had the Wi-fi connection on or off, at the time that the participant answered about the boring status. The value 0 means "it is off", and the value 1 implies "it is on";
- **Boredom status answer**: it is the *target feature* and represents the quantification of the participant boring status.

For each *feature*, it is possible to consult its distribution concerning the *target* label in Appendix A. For a mere example, the distribution graph for the variable "Current Notification" is shown in Figure 3.4, which makes visible the relationship of the higher number of notifications received, the less tendency for someone to be bored.

All *features* of this *dataset* have a numerical nature, except for the two mentioned dates. Therefore the remaining variables belong to one of these two distinct groups:

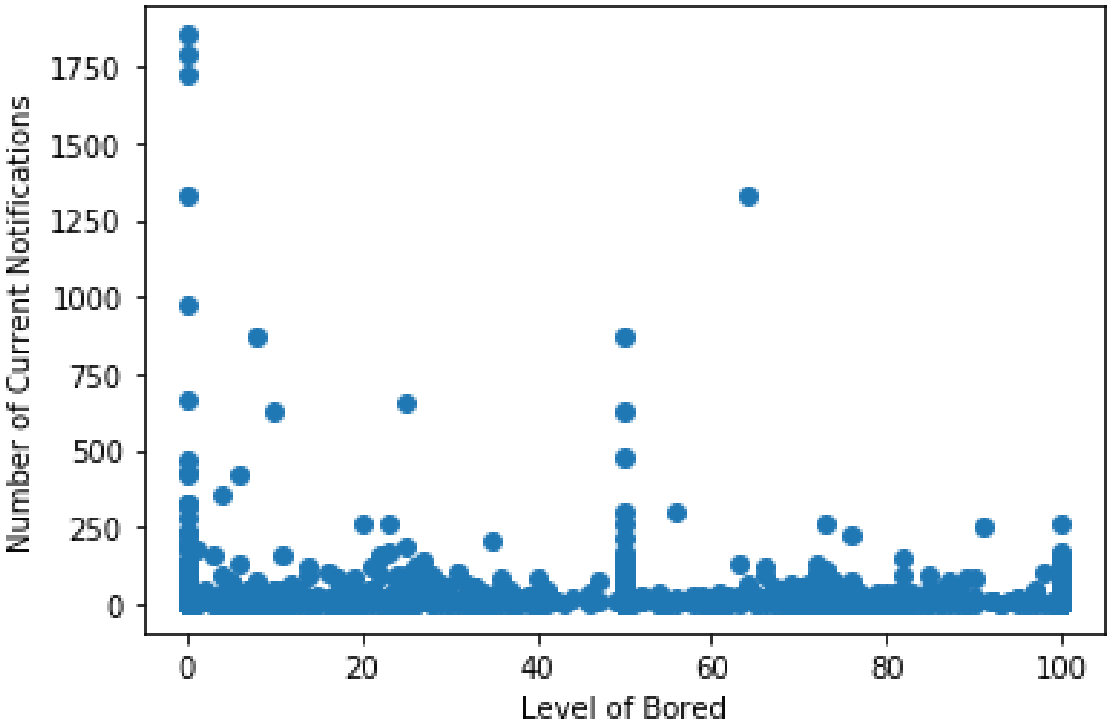


Figure 3.4: Distribution chart of the number of notifications received.

- **Discrete variables**: such as counters that indicate the number of calls made, the number of calls received, the total number of notifications received, the number of notifications received from social networks, the number of notifications received from chatting applications, the number of notifications received from other types of applications, number of text messages received, the number of notifications removed by the user, the number of times that the user pressed the Home button, the number of times that the user pressed the Recent Apps button and the number of activations of the smartphone screen. Also, the flags that inform about the status of the Bluetooth connection, the Ringer mode and the smartphone orientation have this same nature. The Boolean values that appoint if the audio jack is connected, if the charger is connected, if the flight mode is "on" or "off", if the mobile data is "on" or "off", and if the Wi-Fi network connection is "on" or "off" are also part of this group. Lastly, the battery levels are also part of this cluster;
- **Continuous variables**: such as the values that represent the accelerometer, gyroscope, gravity, luminosity, proximity, pressure and magnetic sensors.

Table 3.1 outlines the behaviour of each variable, showing the number of records found, statistical mean, standard deviation, first, second and third quartiles, and also shows the maximum and minimum values.

A correlation matrix was built, being possible to recognize some relevant values that may point to some interesting conclusions:

	count	mean	std	min	25%	50%	75%	max
Accelerometer	1526	-15.77	124.37	-999	-0.99	-0.06	0.68	30.80
Audio Jack	1526	-379.66	485.11	-999	-999	0	0	1
Battery Level 1	1526	54.31	75.96	-999	39	59	81	100
Battery Level 2	1526	42.72	131.37	-999	37	58	80	100
Bluetooth Connection	1526	-13.92	121.90	-999	1	1	1	3
Chatting Notifications	1526	11.61	28.96	0	0	0	10	481
Current Notifications	1526	34.45	114.46	0	2	10	30	1853
Flight Mode	1526	-15.05	121.76	-999	0	0	0	1
Gravity Sensor	1526	-194.43	395.65	-999	-3.53	-0.17	0.31	9.77
Gyroscope	1526	-224.54	417.15	-999	-0.66	0	0.01	7.47
Home Button Press	1526	4.45	5.78	0	1	2	6	60
Incoming Calls	1526	0.20	0.75	0	0	0	0	8
Charging	1526	-14.96	121.77	-999	0	0	0	1
Luminosity Sensor	1526	-34.84	1055.99	-999	0	9	62	20961
Magnetic Sensor	1526	-190.54	397.47	-999	-33.20	-8.58	13.69	897.80
Mobile Data	1526	-14.72	121.80	-999	0	0	1	1
Notifications Removed	1526	6.72	8.64	0	1	4	9	107
Orientation	1526	-14.04	121.88	-999	1	1	1	2
Other Notifications	1526	22.27	109.48	0	0	2	12.75	1832
Outgoing Calls	1526	0.21	0.82	0	0	0	0	12
Pressure Sensor	1526	-174.67	982.12	-999	-999	-999	993	1023.87
Proximity Sensor	1526	-9.34	122.50	-999	5	8	8	10
Recent Button Press	1526	1.62	3.71	0	0	0	2	48
Ringer Mode	1526	-13.97	121.89	-999	1	1	2	2
Screen Activations	1526	7.18	7.24	0	2	5	10	61
SMSes Received	1526	0.75	2.13	0	0	0	0	24
Social Notifications	1526	0.57	2.01	0	0	0	0	23
Wifi	1526	-14.50	121.83	-999	0	1	1	1
Boredom status answer	1526	40.86	35.50	0	1	41	71	100

Table 3.1: Descriptive matrix of the *dataset*.

- **0.4737**: it is the correlation value between magnetic sensor values and the gyroscope. There are other hardware sensors with higher correlation values, such as the gravity sensor and the accelerometer, and this is normal since both are physically related. In this case, the gyroscope and the magnetic sensor are independent of each other, so this principle is not applied;

- **0.5304**: it is the correlation value between the gravity sensor and the accelerometer. Earlier mentioned, these are two physically related sensors. However, their correlation value is not as high as expected, which may indicate that different mobile phone manufacturers may have these sensors working differently;

- **0.5631**: it is the correlation value between the number of notifications removed, and the

number of notifications received related to chatting applications during the evaluation period. This value implies that the user removed a good part of the chatting notifications;

- **0.5803**: it is the correlation value between the number of total notifications received and the number of chatting notifications during the evaluation period. This value shows that the majority of the notifications received are as chatting notifications;
- **-0.8163**: It is the correlation value between the Wi-Fi connection state and the mobile data connection state. Since this value is negative and so close to -1, it is logical to conclude that if on connection is "on", the other is "off", and vice-versa. Which implies that most people are online most of the day.

It is good practice to test if the *features* have a Gaussian distribution. For this attempt, all *features* went through two different types of tests. The Kolmogorov-Smirnov test, and the Shapiro-Wilk test, both trials had an alpha equal to 0.05. In both experiments, all *features* rejected the null hypothesis, concluding that they all have a non-Gaussian distribution. Therefore techniques like Rochac et al. (2019) that permits to augment the original *dataset* are impossible to use. Comparing the correlation values between the *features* and *target* label is not sufficient to conclude which have more influence on the response given by participants. The reason why is because the correlation value only indicates the behaviour of the *target* label according to the *feature* value. Therefore the *dataset* went through an F-test, to find out which *feature* could play a more dominant role over the *target*. The alpha of the F-Test was equal to 0.05, and these are the highest results:

- **15.30**: it is the value attributed by the F-Test to the number of other notifications and the level of boredom;
- **26.03**: it is the value attributed by the F-Test to the mobile data connection status and the level of boredom;
- **38.58**: it is the value attributed by the F-Test to the magnetic sensor and the level of boredom;
- **45.69**: it is the value attributed by the F-Test to the Wi-Fi connection status and the level of boredom;
- **68.85**: it is the value attributed by the F-Test to the Bluetooth connection status and the level of boredom.

3.3 DATA PREPARATION

The *dataset* in its raw state has a lot of information that can be used and extracted. Various *ML* models could use the original *dataset* and consequently obtain decent results. However, before ex-

ecuting any modelling tasks, the collected data went through several techniques of processing. The main reason for this step is to take the maximum benefit from the data collected by the *BoredomApp* and ensure better results right from the start.

3.3.1 Unwanted values

The original *dataset* has 1526 records and 29 *features*. As mentioned, during the collection of data, the mobile app ensures the non-existence of *entries* with missing values. Consequently the *dataset* continued with 1526 records and 29 *features*. At this phase, no decision will be taken yet on the null values assuming a -999 quantification, because models such as *DT* or *ANN* may take advantage of null values of this nature. Later in this document, these values are treated and the results obtained compared. Even after this step, the size of the *dataset* remains the same.

The internal functioning of a *Broadcast Receiver* consists of receiving specific virtual signals from the smartphone, after the catch of that signal, the broadcast performs determined action. *Broadcast Receiver* responsible for the apprehension of the audio jack connection signal only change the *feature* value to 1 or 0, if the participant plugs in an extension to the mobile device. Several participants did not connect any extensions to the audio jack of their smartphone during the data collection period. Therefore those -999 values were converted to 0, with a certain level of security, as it is unlikely that, for example, a person will always carry a headset connected to their mobile device. Table 3.2 shows the comparison between this *feature* distribution before and after this transformation. The result obtained by the F-test and the correlation value between the *target* and this *feature* did not get a wide variation.

	count	mean	std	min	25%	50%	75%	max
Old A.J. distribution	1526	-379.66	485.11	-999	-999	0	0	1
New A.J. distribution	1526	0.04	0.19	0	0	0	0	1

Table 3.2: Comparison between the old and the new distribution of the Audio Jack (A.J.) *feature*.

3.3.2 Unwanted entries

A total of 23 responses from participants have only the level of boredom. All the remaining *features* contained null values. Thus these *entries* will only be noise for the training process of the models. The data are completely anonymous, but these 23 lines have a high chance of being related. Because of their amount, there is a strong probability that they belong to the same person, who has a smartphone with a distinct version of Android. In this case, the most sensible thing to do is undoubtedly to delete these lines from our data. Therefore, the *dataset* goes from 1526 *entries* to 1503. Table 3.3 presents the descriptive matrix of the data collection after the conclusion of this process.

	count	mean	std	min	25%	50%	75%	max
Accelerometer	1503	-0.72	25.97	-999	-0.90	-0.06	0.71	30.80
Audio Jack	1503	0.04	0.19	0	0	0	0	1
First battery level	1503	59.07	25.62	4	39	59	81	100
Second battery level	1503	58.66	25.49	3	38.50	59	80	100
Bluetooth connection	1503	1.16	0.53	1	1	1	1	3
Chatting notifications	1503	11.73	29.10	0	0	0	10	481
Current notifications	1503	34.87	115.26	0	2	10	31	1853
Flight mode	1503	0	0.06	0	0	0	0	1
Gravity sensor	1503	-182.12	385.84	-999	-2.97	-0.16	0.33	9.77
Gyroscope	1503	-212.69	409.09	-999	-0.48	-0	0.01	7.47
Home button press	1503	4.51	5.80	0	1	2	6	60
Incoming calls	1503	0.21	0.75	0	0	0	0	8
Charging	1503	0.10	0.30	0	0	0	0	1
Luminosity sensor	1503	-20.08	1057.23	-999	0	10	66	20961
Magnetic sensor	1503	-178.17	387.61	-999	-31.06	-8.28	13.96	897.80
Mobile data	1503	0.34	0.47	0	0	0	1	1
Notifications removed	1503	6.81	8.68	0	1	4	9	107
Orientation	1503	1.04	0.19	1	1	1	1	2
Other notifications	1503	22.56	110.29	0	0	2	13	1832
Outgoing calls	1503	0.21	0.82	0	0	0	0	12
Pressure sensor	1503	-162.06	984.25	-999	-999	-999	993.15	1023.87
Proximity sensor	1503	5.80	2.86	0	5	8	8	10
Recent button press	1503	1.64	3.74	0	0	0	2	48
Ringer mode	1503	1.11	0.75	0	1	1	2	2
Screen activations	1503	7.27	7.25	0	2	5	10	61
SMSes received	1503	0.76	2.15	0	0	0	0	24
Social notifications	1503	0.58	2.02	0	0	0	0	23
Wi-Fi	1503	0.57	0.50	0	0	1	1	1
Boredom status answer	1503	41.25	35.48	0	3	46	72	100

Table 3.3: Descriptive matrix of the *dataset* after filtering the unnecessary lines.

3.3.3 Feature Engineering

Cleaning up the noise of a *dataset* is not enough to obtain the max benefit from it. It is often possible to extract information and knowledge from existing data, and thus get even more data. This process is called *Feature Engineering*. The two dates collected are in date format, are not useful for the training process of the *ML* models, so each date has been divided into six numeric fields. These fields are year, month, day of the month, day of the week, hour and minute. With the help of the date of the answer, arise a new *feature* which is a Boolean value for the question "Is it a weekend?", where the value 0 represents "no", and the value 1 represents "yes". A new *feature* also emerged that represents the time difference between the two dates in seconds.

A great way of entertainment during the most tedious hours is undoubtedly to play a game on a

smartphone. But because of the privacy policy of *Google*, it would be impossible to publish *BoredomApp* if it could know which application is in the foreground when it is in background. Therefore it is impossible to know when a participant is playing or not. Trying to discover through notifications received related to games also becomes a difficult task, because it would be difficult to know what games the participants would have. However, it is quite common for a game to spend a higher percentage of the battery of the mobile phone, as it usually uses more physical resources such as processor or graphics. Since there are two levels of battery in the data collection, the first refers to the beginning of the evaluation period, and the second refers to the moment when the participant answers the question, a new *feature* appears that indicates the percentage of battery spent between these two moments.

A new *feature* emerged to represent a qualification of the mobile phone in terms of technology. The objective for this idea is to find some pattern between the level of boredom and the price of the device (since the more sensors it posses, the higher the manufacturing cost). This new column represents the number of sensors present on the mobile device. The value of this column in each row is equal to the number of sensors that had not a null value. Table 3.4 shows the distribution of the new *fatures*. The reader may notice that in this table exists only one column referring to the month and one column referring to the year. The reason for this to happen is because in every record the two dates have the same month and the same year, thus repeated and unnecessary information, is not kept.

	count	mean	std	min	25%	50%	75%	max
First minute	1503	29.69	17.56	0	15	29	45	59
First hour	1503	14.09	5.88	0	11	15	19	23
First day of month	1503	12.27	7.40	1	6	10	17	31
First day of week	1503	4.32	1.96	1	3	5	6	7
Second minute	1503	28.16	17.21	0	13	28	43	59
Second hour	1503	14.22	5.91	0	11	15	19	23
Second day of month	1503	12.28	7.40	1	6	10	17	31
Second day of week	1503	4.32	1.96	1	3	5	6	7
Month	1503	10.93	3.22	1	12	12	12	12
Year	1503	2019.10	0.30	2019	2019	2019	2019	2020
Is weekend?	1503	0.34	0.48	0	0	0	1	1
Time diff. in seconds	1503	1022.04	1744.09	0	300	540	1020	41220
Battery spent	1503	-0.41	2.65	-8	-1	-1	0	33
Number of sensors	1503	1.33	1.76	0	0	1	1	5

Table 3.4: Descriptive matrix of the new *features*.

The distribution of the new *features* in relation to the *target* variable is found in Appendix A. Like the other columns, these do not have a Gaussian distribution either. The shape of the data collection goes from 1503 records and 29 columns to 1503 records and 43 columns. When it comes to correlation, these are the values that stand out:

- **-0.528**: it is the correlation value between the number of sensors and the gyroscope, i.e., as the value detected by the gyroscope increases, the number of sensors present in the mobile phone decreases, and vice versa. The low presence of null values in the gyroscope column indicates that the gyroscope should be one of the sensors with the lowest manufacturing cost, since when a smartphone has fewer sensors, at least one of them is the gyroscope.

- **-0.8698**: it is the correlation value between the number of sensors and the pressure sensor, i.e., as the value detected by the pressure sensor decreases, the number of sensors present in the mobile phone increases, and vice versa. Contrary to the previous example, this sensor should be one of the sensors with the highest manufacturing cost. The high number of null values in the pressure sensor column corroborates this conclusion.

These new *features* went through the F-Test, and these are the ones with the highest results:

- **16.54**: it is the value attributed by the F-Test to the number of sensors and the level of boredom;

- **8.69**: it is the value attributed by the F-Test to the weekend Boolean value and the level of boredom.

3.3.4 Normalization

By viewing the various tables in this chapter, the reader can notice that the multiple *features* of the *dataset* have different variations. There are columns like the luminosity sensor that reaches the tens of thousands, and there are columns like the flight mode that only have 0 or 1. Some *ML* models may assume that due to their size, the luminosity sensor *feature* may be more important than the flight mode *feature*. Nothing indicates that it is more important to know the value of the luminosity sensor than to know if the flight mode is on or off, to discover if a person is bored. It is necessary to make all the columns vary in the same range of magnitude, to prevent this from happening. Since the smallest interval found in the data collection is 0 and 1, then it is logical to choose this to be the new range of magnitude. This process is known as the *normalization* of a *dataset*. Table 3.5 shows the descriptive matrix of the data collection after this technique. This method also allows the optimization of time spent in the models training process, as well as the computational resources spent.

3.3.5 Target testing

To this point, the reader may already have noticed that the *target feature* plays a different role than the others present in the *dataset*. This column is used differently during the model training phase. Consequently, it will have more deep processing compared to the procedures performed so

far. Like any human feeling, it is hard to quantify boredom in a general way. So it is irrational to say something like: "Right now I am at a 75% level of boredom". None of the participants thought with that level of precision during the data collection period. The user of the smartphone did not know what value the answer had because the interface of the question did not have a visible scale. With this in mind, it is perhaps inadvisable to predict the level of boredom rather than whether the person is bored or not. But since the possible values of the responses concerning the state of boredom vary between 0 and 100, it is necessary to category these values into bins.

Some hypotheses have emerged to make this possible. How many possible states of boredom exists and which ones are the desirable ones. Given these circumstances, it is unanimous categorising a "boring" and a "very boring" states as the same. In contrast, an "entertained" or "very entertained" feeling can also be identical. One hypothesis based on this type of situation, to convert the answers from 0 to 100 to "is bored" or "is not bored". However, there may be people who do not know how to qualify their state of boredom/entertainment. Another hypothesis is to divide the various answers into three categories, one being "bored", another being "not bored" and finally one being "doesn't know" or "neither". Dividing the various responses into these categories also brings several alternatives. For the two categories hypothesis, one choice is to divide the answers from 0 to 49 as being "not boring" and from 50 to 100 as being "boring". Another option is to divide the categories so that each has the same number of records, and logically the division is different. This division can make a big difference, because in general, it is fundamental that the models do not have too much training in just one of the categories, thus avoiding the problem of *overfitting*. If applying this same notion to the three category hypothesis, there are four different approaches to divide the data collection answers. These are the results:

- **Two categories with symmetrical intervals**: 758 answers for "not bored" and 745 answers for "bored". The "not bored" category covers the answers from 0 to 49 and the "bored" category retreats the answers from 50 to 100;
- **Two categories with the same number of answers**: 755 answers for "not bored" and 748 answers for "bored". The "not bored" category covers the answers from 0 to 45 and the "bored" category retreats the answers from 46 to 100;
- **Three categories with symmetrical intervals**: 699 answers for "not bored", 389 answers for "not sure" and 415 answers for "bored". The "not bored" category covers the answers from 0 to 33, the "not sure" category incorporates the answers form 34 to 66 and the "bored" category retreats the answers from 67 to 100;
- **Three categories with the same number of answers**: 504 answers for "not bored", 520 answers for "not sure" and 479 answers for "bored". The "not bored" category covers the answers from 0 to 13, the "not sure" category incorporates the answers form 14 to 49 and the "bored" category retreats the answers from 50 to 100.

Figure 3.5 illustrates all the divisions, along with the absolute frequency of each category so the reader can have a better visualization of the different divisions.

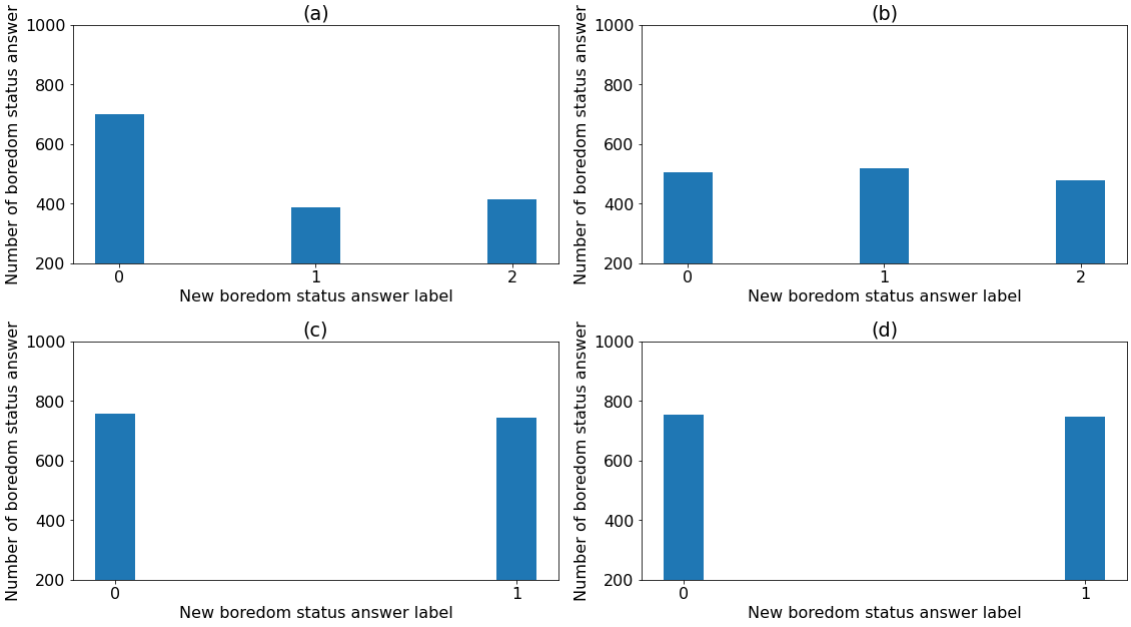


Figure 3.5: The (a) chart shows the absolute frequencies of the three categories with symmetrical intervals, where the value 0 means "not bored", the value 1 implies "not sure", and the value 1 indicates "bored". The (b) chart shows the absolute frequencies of the three categories with the same number of answers, where the value 0 means "not bored", the value 1 implies "not sure", and the value 1 indicates "bored". The (c) chart shows the absolute frequencies of the two categories with symmetrical intervals, where the value 0 means "not bored", and the value 1 implies "bored". The (d) chart shows the absolute frequencies of the two categories with the same number of answers, where the value 0 means "not bored", and the value 1 implies "bored".

The best way to find out which of the four divisions is the most suitable for this research is through the development of an *ML* model. This model has to have the same setup in the training process of the four different approaches so that the results are intrinsic. The model chosen was a *SVM*, and like all the models presented in this dissertation, it has several parameters dedicated to the training process. Later the reader will found a more detailed explanation of how a *SVM* works according to the various values of some of these parameters. For now, follows a list of the parameters used in this pre-modelling as well as their possible values:

- **Kernel**: [rbf, sigmoid, poly, linear];
- **Degree**: [1,2,3,4,5,6,7];
- **C**: [0.001, 0.01, 0.1, 1]
- **Gamma**: [0.001, 0.01, 0.1, 1, scale, auto]

In this phase, using an algorithm called *Grid Search*, the models went through a quick optimization. This algorithm created an *SVM* for each possible combination of the mentioned parameters. Then each *SVM* trains with the data and its corresponding *target*. In all models, the cross-validation is equal to 5. In the end, each hypothesis reaches the best possible configuration and its respective accuracy. Table 3.6 presents these results.

In conclusion, for the models used in the *SL*, *SSL* and *DL* paradigms, we decided to divide the *target* label into two categories, one category representing the "not boring" answers, ranging from 0 to 49, and another bin representing the "boring" answers, ranging from 50 to 100. The fact that this is the hypothesis that has obtained the best accuracy using *SVM* models supports the final decision.

3.4 TECHNOLOGIES AND FRAMEWORKS

Just as a sculptor needs his utensils to make a statue, a software engineer also needs tools to carry out such an investigation. Up to this point, all the work done becomes possible with the help of various tools and utensils.

- **Java**: is the programming language used to build the *BoredomApp* and its respective elements such as *Sensor Listeners*, *Broadcast Receivers*, *Interface Activities* and *Background Service*;
- **Android Studio**: is the *Integrated Development Environment (IDE)* used for the construction of *BoredomApp*. The target version of the *Software Development Kit (SDK)* is 29, and the minimum version is 19 so that the application works on as many smartphones as possible;
- **Firebase**: is the tool responsible for connecting and managing the Realtime Database used to maintain the data;
- **Python**: is the programming language used for the visualization and processing techniques mentioned so far;
- **Spyder**: is the *IDE* used for the visualization and processing of the data. This tool is also responsible for the construction of some tables and figures present in this dissertation.

	count	mean	std	min	25%	50%	75%	max
Accelerometer	1503	0.97	0.03	0	0.97	0.97	0.97	1
Audio Jack	1503	0.04	0.19	0	0	0	0	1
First battery level	1503	0.57	0.27	0	0.36	0.57	0.80	1
Second battery level	1503	0.57	0.26	0	0.37	0.58	0.79	1
Bluetooth connection	1503	0.08	0.26	0	0	0	0	1
Chatting notifications	1503	0.02	0.06	0	0	0	0.02	1
Current notifications	1503	0.02	0.06	0	0	0.01	0.02	1
First day of month	1503	0.38	0.25	0	0.17	0.30	0.53	1
First day of week	1503	0.55	0.33	0	0.33	0.67	0.83	1
Second day of week	1503	0.55	0.33	0	0.33	0.67	0.83	1
Second day of mont	1503	0.38	0.25	0	0.17	0.30	0.53	1
Flight mode	1503	0	0.06	0	0	0	0	1
Gravity sensor	1503	0.81	0.38	0	0.99	0.99	0.99	1
Gyroscope	1503	0.78	0.41	0	0.99	0.99	0.99	1
Home button press	1503	0.08	0.10	0	0.02	0.03	0.10	1
First hour	1503	0.61	0.26	0	0.48	0.65	0.83	1
Second hour	1503	0.62	0.26	0	0.48	0.65	0.83	1
Incoming calls	1503	0.03	0.09	0	0	0	0	1
Charging	1503	0.10	0.30	0	0	0	0	1
Luminosity sensor	1503	0.04	0.05	0	0.05	0.05	0.05	1
Magnetic sensor	1503	0.43	0.20	0	0.51	0.52	0.53	1
First minute	1503	0.50	0.30	0	0.25	0.49	0.76	1
Second minute	1503	0.48	0.29	0	0.22	0.47	0.73	1
Mobile data	1503	0.34	0.47	0	0	0	1	1
Month	1503	0.90	0.29	0	1	1	1	1
Notifications removed	1503	0.06	0.08	0	0.01	0.04	0.08	1
Orientation	1503	0.04	0.19	0	0	0	0	1
Other notifications	1503	0.01	0.06	0	0	0	0.01	1
Outgoing calls	1503	0.02	0.07	0	0	0	0	1
Pressure sensor	1503	0.41	0.49	0	0	0	0.98	1
Proximity sensor	1503	0.58	0.29	0	0.50	0.80	0.80	1
Recent button press	1503	0.03	0.08	0	0	0	0.04	1
Ringer mode	1503	0.55	0.37	0	0.50	0.50	1	1
Screen activations	1503	0.12	0.12	0	0.03	0.08	0.16	1
SMSes received	1503	0.03	0.09	0	0	0	0	1
Social notifications	1503	0.03	0.09	0	0	0	0	1
Wi-Fi	1503	0.57	0.50	0	0	1	1	1
Year	1503	0.10	0.30	0	0	0	0	1
Time diff. in seconds	1503	0.02	0.04	0	0.01	0.01	0.02	1
Battery spent	1503	0.19	0.06	0	0.17	0.17	0.20	1
Number of sensors	1503	0.27	0.35	0	0	0.20	0.20	1
Is weekend?	1503	0.34	0.48	0	0	0	1	1
Boredom status answer	1503	41.25	35.48	0	3	46	72	100

Table 3.5: Descriptive matrix of the *dataset* after the normalization.

Hypothesis	Accuracy	Kernel	Degree	C	Gamma
Two categories with symmetrical intervals	59.89%	Poly	2	1	0.1
Two categories with the same number of answers	59,75%	Poly	2	0.01	1
Three categories with symmetrical intervals	51.30%	Poly	2	1	Scale
Three categories with the same number of answers	44.70%	Poly	2	0.001	1

Table 3.6: Results of the *SVM* Grid Search.

4. EXPERIMENTS

Once the data processing is complete, it is necessary to choose the best *ML* model to incorporate it into the final mobile application. It is critical to approach as many models as possible, from several different paradigms, to make that decision. Several models of all paradigms have gone through various experiments, with the execution of *RL*. The following sections demonstrate the whole process carried out, step by step, to choose the final model.

4.1 EXPERIMENTAL SETUP

All experiments were submitted to the same computational environment so that there would be no model that would be favoured, and so everyone would be on an equal stand. All models have gone through the same optimization algorithm to find their best hyper-parameter setting. This algorithm will be explained in detail so that the reader understands the procedure to improve each model, independently of its learning paradigm. There are three different approaches to the use of the data processed in the previous chapter. The *dataset* went through more techniques and processes in each of these approaches to achieve even better results.

4.1.1 Computational resources

When it comes to*ML* modelling, as in Chapter 3, Python is the programming language used. There are several libraries in this language that provide implementations of the various models. For the construction of the *ANN*, the imported library is *Tensorflow*, and in the other cases, the imported library is *Scikit-Learn*. One of the approaches uses the *XGBoost* library for the import of the *GB* optimization meta-model. *Google Colab* is an 24/7 online platform that provided computational resources such as RAM and CPU remotely, thus allowing a long and interrupted execution of the optimization algorithms.

4.1.2 Hyper-parameters tuning

A parameter used for the construction of a *DT* is the criterion used for the separation of a node, which in the implementation of the *DecisionTreeClassifier* provided by *Scikit-Learn* can only assume two values, "Gini" or "Entropy". If all hyper-parameters used in the development of the models were like this, it would be enough to use a *Grid Seacrh* with all possible combinations, and the best configuration would be found, just like in Section 3.3.5. However, some models have parameters that allow an infinity possibility of values, such as the degree of the function used in the hyperplane separation in an *SVM* with the polynomial kernel, which can be any integer. In this case, a *Grid Search* becomes obsolete because it is impossible to test all possibilities. This research uses a more robust optimization algorithm to get around this difficulty.

Genetic Algorithm

Ikotun Abiodun et al. (2011) cites that the idea of *Genetic Algorithm* stemmed from the evolutionary theory of Charles Darwin called Darwinian evolution. The algorithm starts with the creation of a *Population*, and a *Chromosome* represents each one of these individuals. Time goes by, and each *Generation* gives birth to a new one. In each iteration, all elements of the *Population* suffer an appraisal by an objective function which receives as input the respective *Chromosome*. This process is called *Evaluation*. The grades provided by the goal function serve to indicate which individuals are the most suitable to reproduce among themselves. This stage is also known as *Selection*. The chosen elements of the *Population* generate new beings among themselves, i.e., a *Crossover* occurs, and the *Chromosome* of each new-born individual is a combination of the *Chromosomes* of their parents. The elements of the *Population* excluded from the *Selection* phase are eliminated and replaced by these new individuals. During each *Generation*, there is a probability of changing a *Chromosome* content, like a *Mutation* does to our genetic code. Instead of a textual explanation, Figure 4.1 schematises an iteration of a *Genetic Algorithm* so that the reader can understand a more intuitive way.

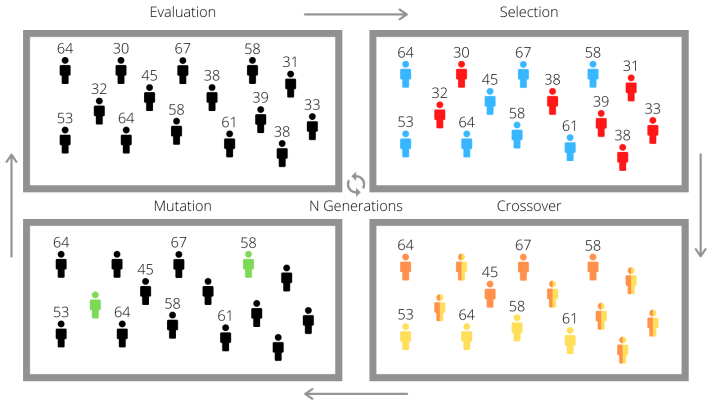


Figure 4.1: The first frame describes the *Evaluation* phase, where each individual is appraised and supplied with a score. The second frame represents the *Selection* phase, where the blue elements have the best scores and will reproduce, and the red individuals have the worst scores and will disappear. The frame bellow depicts the *Crossover* step, where each newborn individual is a combination of two *Chromosomes* selected in the previous phase. The last frame shows the *Mutation* step, where some individuals may suffer a change in their genetic code. This process repeats an arbitrary number of *Generations*.

Regarding the *Genetic Algorithm* used to optimize the various models in this research, here is a list with the variables and their respective values, and the several aspects of each phase:

- **Population**: consisting of 70 individuals initiated with random *Chromosomes*;
- **Chromosome**: composed of the set of hyper-parameters used in the construction of a given model and for this reason every *Chromosome* in a population has the same size. The following sections will detail each *Chromosome* composition in detail;

- **Generations**: the number of iterations used for this algorithm is equal to 50;
- **Evaluation**: in this phase, each *Chromosome* represents a possible configuration of parameters for a model. Each model trains with the prepared *dataset*, with *Cross-Validation* equal to 10. The final score for the chromosome is the mean of the *F1-Scores* obtained in each *Cross-Validation* step;
- **Selection**: the selection method in this algorithm is the election of the 24 models with the highest *F1-Score*. Therefore, the *Selection Rate* of this algorithm is approximately 35%;
- **Crossover**: the 24 elected models reproduce at this stage and do it in the following order, i.e., the model with the best score crosses with the second-best, the second-best crosses with the third-best, and onwards. The *Crossover* point is random and varies between 0 and the *Chromosome* size (remember that both parent *Chromosomes* have the same size). From this mating, two new individuals are born, one is the combination of the parent 1 *Chromosome* from position 0 to the *Crossover* point and the parent 2 *Chromosome* from the *Crossover* point to the end. The other individual has an inverse composition, i.e., it is composed by the combination of the parent 2 *Chromosome* from position 0 to the *Crossover* point and the parent 1 *Chromosome* from the *Crossover* point to the end. Thus no genetic material is lost, and each iteration generates 46 new individuals. Therefore this algorithm has a *Crossover Rate* of about 65%;
- **Mutation**: an individual undergoes a *Mutation* when a parameter is changed. The *Mutation Rate* is 15%, i.e., in one hundred generations, fifteen will present a change on a *Chromosome*.

4.1.3 Approaches

To this point, using the data processed in Chapter 3, any model would obtain promising results after being optimised by the *Genetic Algorithm* described in Section 4.1.2. To be content with these results would not be enough, as it is always a good practice to try to achieve more and better. Three new approaches emerge, which both take as their starting point the state of the *dataset* after the treatment referred to in the previous chapter.

First Approach

The starting point for experimentation is the development of a series of models, all trained with the data collection processed in Chapter 3. In summary, the *dataset* has 1503 records and 43 columns. All *entries* are normalized, i.e., they range from 0 to 1, and the *target feature* is composed of two categories, one category for "bored" and another for "not bored". The *Genetic Algorithm* provided in section 4.1.2 has allowed the optimization of all *ML* models, thus obtaining the best *F1-Score* possible.

Second Approach

If the reader remembers, section 3.3.1 reports that no decision has been made concerning the null values, because some models can take advantage of such variables. With the first approach, this hypothesis gains an opportunity to test its veracity. In the second approach, there is a prediction of null values before the execution of the *Genetic Algorithm*. Unlike the main problem of this research, which is to classify at a given moment whether a person is bored or not, predicting a continuous variable is a regression problem. Of all the columns in the data collection, only six contain null values after the data processing performed. These *Features* are the *Accelerometer*, the *Gravity Sensor*, the *Gyroscope*, the *Luminosity Sensor*, the *Magnetic Sensor* and the *Pressure Sensor*. For the *Accelerometer* case, the model executes the training process with all the records where the value is known. Worth mention that the *target* label is also part of the input for the models. Once concluded the training process, the models predict the null values and replace them, and therefore this methodology repeats for the other five columns. Observing some results obtained in the first strategy, which will appear later, three possible models show promising scores, becoming strong candidates to solve this problem, and these are *DT*, *SVM* and *KNN*.

About the optimization of these three different models, a new optimization algorithm called *RandomSearch* rises. This algorithm has a mode of operation very similar to the *GridSearch*, receiving as input a grid of parameters and their possible values, and for continuous variables can assume a range in which the parameter can take any number within those limits. Since it is difficult and costly to build a model for all the values within a continuous interval, *RandomSearch* takes as input the number of iterations. During the optimization process, the algorithm creates random models based on the input grid, and when the number of iterations exceeds, the process ends. The number of iterations chosen for the three models is 1000. The cross-validation used for this algorithm is equal to 10, as in the *Genetic Algorithm*. Table 4.1 shows all the possible values for each hyper-parameter used for this process. A detailed explanation of these hyper-parameters will follow later.

In a classification situation, for a new *entry*, an *ML* model is right if it predicts its category, or is wrong if it categorizes a class that does not correspond to reality. For a regression problem, the predicament is not quite like that. Imagine an example where a model is trying to predict the height of a person based on other biometric information. For a new case of study, the person measures 1.78cm, and the model predicts 1.80cm. The model is not wrong as if it forecast 2.10cm, because it is quite close to the right answer. In a regression problem, metrics like *Accuracy* and *F1-Score* lose their meaning and purpose. A widely used metric in these situations is the *Root-Mean-Square Error (RMSE)*, and Equation 4.1 indicates the way of how to calculate.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(prediction_i - real_i)^2} \tag{4.1}$$

Model	Parameter	Values
Decision Tree	Criterion	{MSE, FRIEDMAN-MSE, MAE}
	Splitter	{Best, Random}
	Max Depth	[1:42] step of 1 unit
K-Nearest Neighbours	Number of Neighbours	[1:30] step of 1 unit
	Leaf Size	[1:50] step of 1 unit
	Weights	{Uniform, Distance}
	Algorithm	{Auto, Ball Tree, KD Tree, Brute}
	P	{1, 2}
Support Vector Machine	C	[0.001:100]
	Kernel	{Linear, Poly, RBF, Sigmoid}
	Degree	[1:10] step of 1 unit
	Gamma	[0.0001:0]

Table 4.1: Hyper-parameters used for *DT*, *KNN* and *SVM*, and respective values.

Looking at the equation, it is clear that the closer the *RMSE* value is to 0, the more it is indicative of how well the model fits the data, just like Yeniay e GÖKTAŞ (2002) affirms. Table 4.2 presents the results of each model in forecasting the *Accelerometer* column. Appendix A contains the remaining tables that exhibit the scores for the other five columns. The model that obtained the lowest *RMSE* value is responsible for the forecast of the null value forecasts for each corresponding column. For this to happen, the model with the best configuration replaces the null values by the values it predicted. All of this process occurs before the application of the *Genetic Algorithm*.

Model	RMSE	Parameter	Value
Decision Tree	2.460	Criterion	MSE
		Splitter	Best
		Maximum Depth	4
k-Nearest Neighbours	3.165	Number of Neighbours	30
		Leaf Size	41
		Weights	Distance
		Algorithm	Ball Tree
		P	1
Support Vector Machine	3.159	Kernel	Polynomial
		Degree	5
		C	3.054
		Gamma	0.067

Table 4.2: Results from the prediction of the null values present in the Accelerometer *feature*.

Third Approach

Sometimes the performance of *ML* models is hampered by the high complexity of the *dataset* provided as input. In other situations, it is the optimization process that is impaired, because the time of the training phase of some models becomes too time-consuming and thus impossible to execute. A good practice to reduce the complexity of a data collection is to decrease its number of *features*, also known as dimensionality reduction. The latest strategy in this research is related to this technique. The third approach is to reduce the number of columns in the *dataset* in an attempt to improve the performance of the models and consequently improve their results. A combination of *RandomSearch* and *GB* is the method used to choose which *features* remain in the original *dataset*. As in the previous approach, this process takes place before the execution of the *Genetic Algorithm*.

This selection is at the expense of the meta-model since, during its training process, it only uses *dataset* samples, in which these samples do not hold all the records either all the columns of the original data collection. In the end, the model can inform which *features* played a more important role during the optimization. Another parameter used to build the meta-model allows the control of the number of *features* used in each sample for each iteration. In the end, taking these two factors into account, it is possible to find out how many and which columns can disappear before the execution of the *Genetic Algorithm*. The main reason for using *RandomSearch* is to optimize the *GB* training stage. The following list describes each parameter used for the creation of each *DT* optimized by *GB*, and Table 4.3 shows the grid of parameters used for the development of the meta-model.

- **Number of boosting rounds**: number of boosting rounds that the base model trains over the residuals;
- **Maximum Depth**: maximum allowed depth for the *DT*;
- **ETA**: learning rate used for the boosting process;
- **Gamma**: minimum loss reduction required to make a further partition on a leaf node of the tree;
- **Minimum Child Weight**: minimum sum of instance weight needed in a child of the base model;
- **Column Sample by Tree**: sample ratio of columns when training the initial *DT*.

For the more attentive reader, it is intuitive to notice that a meta-model, such as an optimization algorithm, tries to improve the performance of a base model. Whenever a technique occurs to improve this performance, *cross-validation* always appears to validate the results. If this strategy uses a meta-model within an optimization algorithm, it is logical that this situation has two nested

Parameter	Values
Number of boosting rounds	[50:2000] step of 1 unit
Maximum Depth	[1:42] step of 1 unit
ETA or Learning Rate	[0.00001:0.1]
Gamma	{0.02, 0.04, 0.08}
Minimum Child Weight	{4, 6, 8}
Column Sample by Tree	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3}

Table 4.3: List of parameters used for the *GB* model and their respective list of possible values.

cross-validations. The external *cross-validation*, which is used for the *RandomSearch*, is equal to 10. However, the internal *cross-validation*, which is used by the *GB*, is equal to 5. The main reason for this difference is if the inner *cross-validation* has a higher value, the base *DT* used in the meta-model will have to few *entries* to train. Appendix B shows the tables with the scores and best settings for each iteration, and Table 4.4 summarizes the results along with the quantity of *features* used.

C.V. Iteration	Inner F1-Score	Outer F1-Score	Column Sample
10	0.595	0.742	0.2
8	0.587	0.696	0.25
9	0.609	0.641	0.2
4	0.581	0.627	0.3
6	0.585	0.607	0.3
5	0.601	0.602	0.25
7	0.590	0.583	0.3
3	0.565	0.517	0.25
2	0.608	0.477	0.25
1	0.605	0.382	0.15

Table 4.4: Results of the *GB* optimized by a *RandomSearch*. This table is order by the *F1-Score* obtained in the exterior *cross-validation*.

The model chosen was the one created in the last iteration, because it is the model with the highest *F1-Score* in the outer *Cross-validation*, and it is the fourth-best in the inner *Cross-validation*. The results of the external *Cross-validation* are more relevant than the counterpart, and the reason why is the score bases in evaluation with data not provided for the training process. Therefore the most relevant columns for the *feature* selection are eight since 0.2 * 42 is approximately equal to 8. Figure 4.2 shows the *heatmap* produce based on the importance values of the *dataset* columns assigned by the best *GB* model.

Comparing the results obtained by the *F-Test* used in Section 3.2, with the scores provided by the *Heatmap*, it is worth to note that both gave the *Bluetooth Connection* column the most relevant role when it comes to predicting boredom using a smartphone. *F-test* indicated the *Magnetic Sensor feature* and the *Other Notifications feature* as important columns concerning the *target* label.

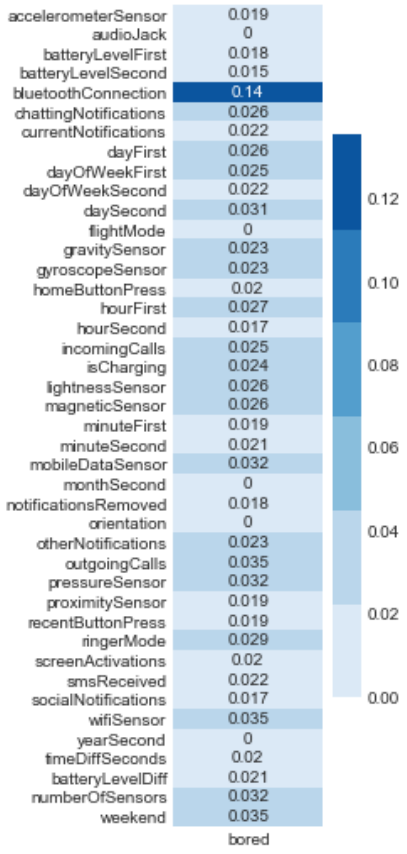


Figure 4.2: *Heatmap* that shows the *feature* importance assigned by the elected *GB*.

However, this final strategy excluded both. After the conclusion of the *Feature Selection*, the final *dataset* consists of the following columns: *Bluetooth Connection, First Day, Mobile Data Status, Number of Outgoing Calls, Pressure Sensor, Wi-Fi Status, Number Of Sensors, Weekend* and of course, the *target* label. The shape of the data collection end with 1503 *entries* and 9 columns.

4.2 DESIGNED MODELS

Section 4.1.2 contains all the details of each step of the *Genetic Algorithm* used in this research. However, a core part of this optimizing program has yet to be explained, which is the *Chromosome* used in each *ML* model.

4.2.1 Supervised learning

The area of *SL* has been the most studied in this research in comparison with the others. Six different models are used in this work: *Lin-Reg, Log-Reg, NB, KNN, SVM* and *DT*. In addition to these, for each approach, two more meta-models were added: *RF* and *AdaBoost*. For each model, this document presents the constitution of the *Chromosome* used in the optimization process. Along

with the *Chromosome* composition, is also detailed the purpose of each parameter.

Linear Regression

Due to the simplicity of this model, the implementation provided by *Scikit-Learn* only has one optimization parameter, which accepts a boolean value. Therefore, there are only two different types of *LR* models possible to build, making the *Genetic Algorithm* obsolete. However, this section will keep the same structure as the others for consistency. The constitution of the hypothetical *LR chromosome* is as follows:

- **Fit Intercept**: boolean value which indicates whether to calculate the intercept for the *Lin-Reg* model. If set to False, no intercept will be used in calculations, in other words, data is expected to be centred. One of the models has this parameter set to True, and the other to False.

Logistic Regression

Unlike the previous model, this one has a more complex implementation, so the use of the genetic optimization algorithm is already useful. The following is the list with the parameters that compose the *chromosome* used to optimize the *Log-Reg* model:

- **Penalty**: used to specify the norm used in the penalization. There are four possible values to this parameter, which are L1, L2 and Elasticnet and None. However, only the L2 is used for the optimization process, because only this one is compatible with all solvers;
- **C**: is a positive real number and is a regularization parameter. The strength of the regularization is inversely proportional to C. For the optimization process can assume values between 0.001 and 1000;
- **Solver**: indicates which algorithm is used for the regularization process and can be the Newton-CG, LBFGS, LibLinear, SAG and SAGA.

Another parameter used in optimization is the maximum number of iterations. To force the Solver to converge and not to exceed the number of iterations, the maximum number of iterations is equal to one million in all the individuals created by the genetic algorithm.

Gaussian Naive Bayes

The implementation provided by *Scikit-Learn* for this model is also quite simple as it only has one optimization parameter. However, this parameter can assume a multitude of values, which does

not prevent the use of the *Genetic Algorithm*. The chromosome composition of this model is the following.

- **Variance Smoothing**: real number that indicates the portion of the largest variance of all *features* that is added to variances for calculation stability. This variance can take a value between 10^{-10} to 1.

K-Nearest Neighbours

The following list details the composition of the chromosome used for the optimization of the *KNN* models. For each parameter its function is indicated as well as the values it can assume.

- **Number of Neighbours**: indicates the number of neighbours to use by default for k-Neighbours queries. The number is in the range between 1 and 30;
- **Leaf Size**: dictates the number of the leaf size. This can affect the speed of the construction and query, as well as the memory required to store the tree. The number is in the range between 1 and 50;
- **Weights**: it represents the weight function used in prediction, and it can be Uniform or Distance. If the function is Uniform than all points in each neighborhood are weighted equally. If is Distance then the weight of the points are inverse of their distance. In this case, closer neighbours of a query point will have a greater influence than neighbours which are further away;
- **Algorithm**: it implies the algorithm used to compute the nearest neighbours, and the algorithm may be Auto, Ball Tree, KD Tree or Brute. If the algorithm is Ball Tree the model will use the Ball Tree algorithm, as the name implies. If KD Tree the same logic is applied. Then if is Brute the model will use a brute-force search. Finnaly if is auto the model will attempt to decide the most appropriate algorithm based on the values passed to the training method;
- **P**: can be equal to 1 or 2 and indicates the power for the Minkowski metric. If P = 1, then the model is using Manhattan Distance. If P = 2, then the model is using the Euclidean Distance.

Support Vector Machine

The following list details the composition of the chromosome used for the optimization of the *SVM* models. For each parameter its function is indicated as well as the values it can assume.

- **Kernel**: specifies the kernel type to be used in the algorithm. It can be one of Linear, Polynomial, RBG or Sigmoid;

- **Degree**: is the degree of the polynomial kernel function, and is ignored by the other kernels. This number can assume values between 1 and 10;

- **C**: like in the *Log-Reg* model is a regularization parameter. The strength of the regularization is inversely proportional to C. Is real number strictly positive and varies between 0.001 and 100. The penalty is a squared l2 penalty;

- **Gamma**: is the coefficient for RBF, Polynomial and Sigmoid kernels. This parameter is a real number and varies between 0.0001 and 1.

Decision Tree

The following list details the composition of the chromosome used for the optimization of the *DT* models. The *DT* is the last model of this paradigm used in this investigation. For each parameter its function is indicated as well as the values it can assume.

- **Criterion**: represents the function to measure the quality of a split of a node. Supported criteria are Gini for the Gini impurity and Entropy for the information gain;

- **Splitter**: indicates the strategy used to choose the split at each node. Supported strategies are Best to choose the best split and random to choose the best random split;

- **Maximum Depth**: implies the maximum depth of the tree. The value for this parameter varies between 1 and 42 (42 is equal to the number of *features* of the *dataset*).

Random Forest

After the presentation of the *SL* paradigm models, it is the turn of the meta-models. The first meta-model to be detailed is *RF*. It is worth to remind the reader that the base model is a *DT*. The following list shows the constitution of the *Chromosome* used in the optimization process, and for each parameter, its role and the values it can take.

- **Number of estimators**: is the number of trees in the forest. For the *Genetic Algorithm* can assume a value between 1 and 200;

- **Criterion**: represents the function to measure the quality of a split. Supported criteria are Gini for the Gini impurity and Entropy for the information gain;

- **Maximum Depth**: implies the maximum depth of the tree. The value for this parameter varies between 1 and 42 (42 is equal to the number of *features* of the *dataset*);

- **Maximum Number of Features**: signifies the maximum number of *features* of the data collenction to consider when looking for the best split. The value for this parameter varies between 1 and 42 (42 is equal to the number of columns of the *dataset*);
- **Minimum Sample Split**: is the minimum number of samples required to split an internal node. The value of this parameter varies between 2 and 100;
- **Minimum Samples Leaf**: is the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least "Minimum Samples Leaf" training samples in each of the left and right branches. For the optimization process the value of this parameter ranges between 1 and 10;
- **Bootstrap**: is a Boolean value that indicates whether bootstrap samples are used when building trees. If False, the whole *dataset* is used to build each tree;
- **Number of Maximum Samples**: if bootstrap is True, this parameter implies the number of samples to draw from the data to train each base estimator. The value varies between 1 and 1352 (the reason behind the number 1352 is the fact that this is the exact number of records used for the training process).

Ada Boost

Finally, this is the meta-model that closes the section of the *SL* paradigm, and contrary to the previous one, *AdaBoost* can use any base model. The following list details the composition of the *Chromosome* used for the *Genetic Algorithm*.

- **Number of estimators**: is the number of trees in the forest. For the *Genetic Algorithm* can assume a value between 50 and 200;
- **Learning Rate**: is the a real number that shrinks the contribution of each classifier. There is a trade-off between the Learning Rate and the Number of estimators, i.e., situations with more estimators have to have low learning rates, and situations with lesser estimators have to have higher learning rates.

The base model used in the three approaches is different because, for each strategy, the model that obtained the best *F1-Score* within the *SL* paradigm was used as the base estimator, to gain even better results and test the performance of the meta-model.

4.2.2 Unsupervised learning

This document presents four different models for the *UL* paradigm: *K-Means, AC, BIRCH* and *SC*. An important note to remind the reader is that *BIRCH, AC* and *SC* are models based on *Hierarchical*

Clustering. For each model, this section presents the constitution of the *Chromosome* used in the *Genetic Algorithm*. Along with the *Chromosome* structure, it is also detailed the purpose of each parameter.

K-Means Clustering

The following list presents all the parameters used for the *K-Means* optimization process.

- **Tolerance**: a real number relative to the tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. This parameter varies between 10^{-9} and 1;
- **Algorithm**: it is the parameter that implies the used algorithm for the clustering process. There are two possible choices Full and Elkan. The Full algorithm is the classical EM-style. The Elkan variation is more efficient on data with well-defined clusters, by using the triangle inequality.

The number of clusters used is two, one for representing the "bored" answers and the other for the "not bored" responses. Consequently, this is the number of centroids to generate for each iteration. The maximum number of iterations establish is one million to force the model to converge and overpass the tolerance value.

Agglomerative Clustering

When it comes to hierarchical clustering, the first model to be presented is the *AC*. The following list introduces all the parameters that compose the *Chromosome* used in the *Genetic Algorithm*, their functions and the respective values they can assume.

- **Linkage**: indicates which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion. There are four options, the Average, the Complete, the Single and the Ward. The Average criterion uses the average of the distances of each observation of the two sets. The Complete criterion or maximum linkage uses the maximum distances between all observations of the two sets. The Single criterion uses the minimum of the distances between all observations of the two sets. The Ward criterion minimizes the variance of the clusters being merged;
- **Affinity**: metric used to compute the linkage. Can be Euclidean, L1, L2, Manhattan or Cosine.

Like in the previous model, the number of clusters used is two, one for representing the "bored" answers and the other for the "not bored" responses.

BIRCH

The following list introduces all the parameters that compose the *Chromosome* used in the optimization process, their role and the respective values they can assume.

- **Threshold**: a real number that limits the radius of the subcluster obtained by merging a new sample and the closest subcluster. If this values is exceed a new subcluster is started and setting this value to be very low promotes splitting and vice-versa. This parameter varies between 10^{-4} and 1;
- **Branching Factor**: maximum number of subclusters in each node. If a new samples enters such that the number of subclusters exceed the Branching Factor then that node is split into two nodes with the subclusters redistributed in each. The parent subcluster of that node is removed and two new subclusters are added as parents of the 2 split nodes. This parameter can assume values from 2 to 200;
- **Compute Labels**: Boolean value that implies whether or not to compute labels for each fit.

The clustering process of a *BIRCH* bases on the same principle of the *AC*, so this model has two clusters as well. One cluster representing the "bored" answers and other expressing the "not bored" responses.

Spectral Clustering

To conclude the *UL* paradigm, the next lines describe the model with more optimization parameters of this research. The following list presents all the parameters used for the *SC* optimization process.

- **Affinity**: defines how to construct the affinity matrix. Can be Nearest Neighbours, or RBF. The Nearest Neighbours Affinity construct the affinity matrix by computing a graph of nearest neighbours. The RBG Affinity construct the affinity matrix using a radial basis function kernel;
- **Eigen Solver**: implies the eigenvalue decomposition strategy to use. Can be None (no strategy) or Arpack;
- **Number of Components**: is the number of eigen vectors to use for the spectral embedding. This parameter varies from 1 to 100;

- **Gamma**: is a real number that represents the kernel coefficient for the RBF Affinity. This parameter is ignored when Affinity is Nearest Neighbours and can assume values between 10^{-4} and 1;
- **Number of Neighbours**: defines the number of neighbours to use when constructing the affinity matrix using the nearest neighbours method. This parameter is ignored for the RBF Affinity and varies between 1 and 30;
- **Eigen Tolerance**: is real number used as the criterion to stop the Eigen decomposition of the Laplacian matrix when Eigen Solver is Arpack and can assume values from 10^{-9} to 1;
- **Assign Labels**: represents the strategy to use to assign labels in the embedding space. There are two ways to assign labels after the Laplacian embedding. K-Means can be applied and is a popular choice. But it can also be sensitive to initialization. Discretization is another approach which is less sensitive to random initialization;
- **Degree**: is the degree of the RBF Affinity. Ignored when Affinity is Nearest Neighbours and varies from 0 to 10;
- **Coefficient**: is the zero coefficient the RBF Affinity. Ignored when Affinity is Nearest Neighbours and varies from 0 to 10.

Like in the previous models, the number of clusters used is two, one for representing the "bored" answers and the other for the "not bored" responses.

4.2.3 Semi-supervised learning

This essay presents two different models for the *SSL* paradigm: *LP* and *LS*. For each model, this section displays the structure of the *Chromosome* used in the algorithm of optimization. Along with the *Chromosome* composition, is also detailed the purpose of each parameter.

Label Propagation

LP is the first model of this hybrid paradigm, and this is the list of its parameters.

- **Kernel**: this parameter identifies the kernel function to use. There are only two variations, RBF and KNN;
- **Gamma**: is a real number used for the RBF Kernel. This parameter varies between 10^{-4} to 1;
- **Number of Neighbours**: is the number of neighbours used by the KNN kernel. This value ranges from 1 and 30;

- **Tolerance**: is a real number that indicates the convergence tolerance and can assume values from 10^{-9} to 1.

The maximum number of iterations establish in this model is one million to force it to converge and overpass the tolerance value.

Label Spreading

An *LS* model is very similar to an *LP* model, the parameters used for the optimization process are the same, except for the presence of an additional one in the *LS Chromossome*. The following list indicates the parameters used in the *Genetic Algorithm* along their roles and respective values.

- **Kernel**: this parameter identifies the kernel function to use. There are only two variations, RBF and KNN;
- **Gamma**: is a real number used for the RBF Kernel. This parameter varies between 10^{-4} to 1;
- **Number of Neighbours**: is the number of neighbours used by the KNN kernel. This value ranges from 1 and 30;
- **Tolerance**: is a real number that indicates the convergence tolerance and can assume values from 10^{-9} to 1;
- **Alpha**: is a real number that represents the clamping factor. A value between 0 and 1 that specifies the relative amount that an instance should adopt the information from its neighbours as opposed to its initial label. If Alpha equals to 0 means keeping the initial label information, and if Alpha equals to 1 means replacing all initial information.

The maximum number of iterations establish in this model is one million to force it to converge and overpass the tolerance value.

4.2.4 Deep learning

This dissertation presents only one *DL* model, which is the *ANN* or *Multilayer Perceptron (MLP)*. Both *Scikit-Learn* and *Tensorflow* have libraries for the creation of *ANNs*. However, the implementation provided by *Tensorflow* is the one displayed in this research. The factor influencing this decision is that the incorporation of an *ANN* implemented by *Tensorflow* is more accessible. This section presents the structure of the *Chromosome* used in the *Genetic Algorithm* for the *ANN*. Along with the *Chromosome* composition, is also detailed the purpose of each parameter.

Artificial Neural Network

The level of interoperability of the library used for the development of this model, namely *Tensorflow*, is so high that it allows the models to execute their training process using *datasets* similar to this research, where each record is a set of numerical data that identifies a situation/entity. Before listing the parameters of the *Chromosome* used by the optimization process, there are some decisions related to two parameters that need to be detailed.

Evaluate the performance of an *ANN* through its learning curve is a solid practice. A learning curve is a plot of model learning performance over experience or time like Brownlee (2019) affirms. For example, if someone was learning a musical instrument, the skill on the instrument could be evaluated and assigned a numerical score each week for one year. A plot of the scores over the 52 weeks is a learning curve and would show how the learning of the instrument has changed over time. Having a reasonable learning curve avoids the model to over or *Underfitting*, and Figure 4.3 shows an example of a good learning curve.

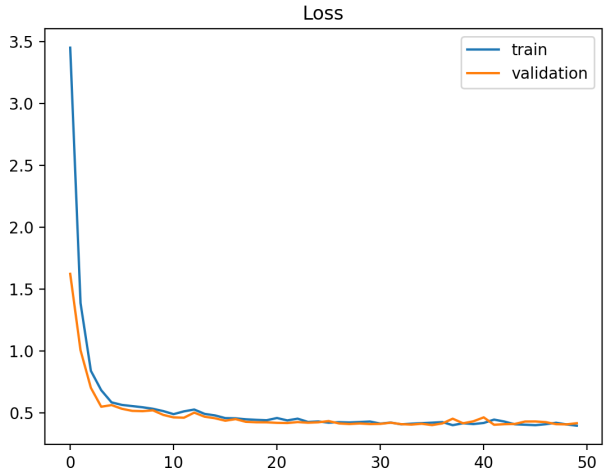


Figure 4.3: The x-axis represents time, and the y-axis represents a score (in this example the score is the loss). The blue line represents the performance of the fictional model during training, and the orange line represents the performance of the fictional model during the validation phase.

Two important aspects that influence this type of plots are the epochs and the batch size. The Epoch is an iteration over the entire *dataset*. The batch size is the parameter that controls the sample size provided in each training iteration. To obtain a reasonable learning curve, after a hit and miss process, the parameters obtained were 10 Epochs and a batch size equal to 8. Figure 4.4 shows the graph obtained with the *dataset* from this search using a standard model.

With that said, for all the *ANN* created during the *Genetic Algorithm*, these values remain the same. The other parameters that compose the *Chromosome* are as follows.

- **Layers**: indicates the number of layers that embodies the neural network. This parameter

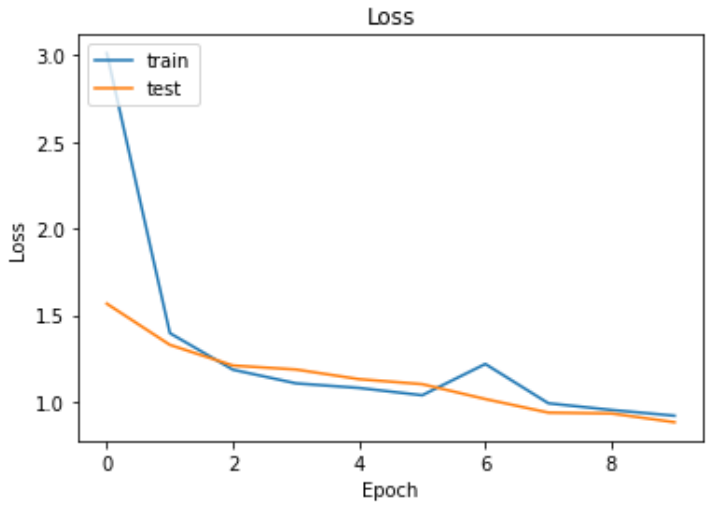


Figure 4.4: The x-axis represents time, and the y-axis represents a score (in this plot the score is the loss). The blue line represents the performance of the fictional model during training, and the orange line represents the performance of the fictional model during the validation phase.

varies between 1 and 6;

- **Nodes**: implies the number nodes per layer. At the programming level, it is an array with a size equal to the number of layers, wherein each position it has the number of nodes of the respective layer. The array values can be 2, 4, 8, 16 or 32;
- **Dropouts**: suggests the existence of a dropout between layers. At the programming level, it is an array with a size equal to the number of layers, wherein each position it has Boolean value that indicates if the Dropout exists or not;
- **Dropout Values**: if the dropout exists, the respective value is given by this parameter. At the programming level, it is an array with a size equal to the number of layers, wherein each position it has the value of the respective dropout. The dropout values vary from 0.3 to 0.5;
- **Optimizer**: shows the optimizer used for the compiling process. The optimizer can be one of the following: SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax or Nadam;
- **Learning Rate**: designates the laerning rate used by the optimizer and varies between 10^{-6} and 10^{-2}.

The compilation of all networks are with Binary Crossentropy as loss function, and the output layer consists of only one node, which indicates the probability of the person being bored.

5. RESULTS AND DISCUSSION

This chapter portrays all the conclusions obtained during the experimentation presented in the previous chapter. For the sake of consistency, this chapter has a similar structure to the previous one. The first section shows an analysis of the results obtained to make it possible to elect of the model to incorporate in the final application. After discussing all the results, this dissertation presents the final product built with the fruits of this research.

5.1 DESIGNED MODELS

As in the previous chapter, this section is divided into learning paradigms, and consequently, each paradigm presents the results of the various models that belong to it. For each model, a graph illustrates its evolution during the three methodologies approached. In addition to the plot, exists a table with the optimal configuration for each approach. In each section, this dissertation displays a reflection of the graph and table obtained for each model.

5.1.1 Supervised learning

This dissertation begins by reflecting the results obtained by the models that belong to the *SL* paradigm. It starts with the presentation of the results of the six different models, *Lin-Reg, Log-Reg, NB, KNN, SVM* and *DT*, and concludes with the results of the two meta-models, *RF* and *AdaBoost*.

Linear Regression

As mentioned above, the *Lin-Reg* implementation provided by *Scikit-Learn* is very simplistic. The only parameter used is the *Fit Intercept*, which only admits two possible values. So the optimization process for this model was to create two different models for each approach, one model with the value True and the other with the value False. Figure 5.1 illustrates the evolution of the optimization performance of the model over the methodologies performed.

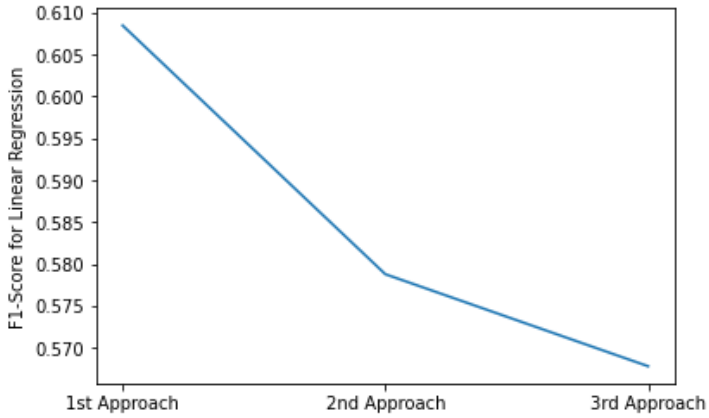


Figure 5.1: *F1-Scores* of the *Linear Regression* model for each approach.

By looking at the graph, it is clear to conclude that both the null value prediction approach and the *features* selection approach have not contributed to improve *F1-Score* of the *Lin-Reg* model. With the *dataset* treated by the procedures indicated in Chapter 3, this model has achieved an *F1-Score* of nearly 61%. After predicting the null values in the second approach, the best *Lin-Reg* model scored 58%. Finally, with the *Features Selection*, the highest registered *F1-Score* was about 57%. From the first to the second approach F1-Score fell by approximately 3%, and from the second to the third by about 1%. These results suggest that this model presents better results with more complex data collections and it adapts well with the presence of *outliers* since the null values are equal to -999. Table 5.1 shows the parameters of the best models per approach.

Approach	Parameter	Value	F1-Score
First	Fit Intercept	False	0.608
Second	Fit Intercept	True	0.579
Third	Fit Intercept	True	0.568

Table 5.1: Parameter setting of the *Lin-Reg* model obtained in each approach.

In two cases, the model with the best score has Fit Intercept marked as True. However, this happens in the last two procedures, where the lower *F1-Scores* happens. The first approach is the one with the best score, which is an *F1-Scores* of approximately 61%, and the Fit Intercept equals False.

Logistic Regression

Contrary to the previous model, *Log-Reg* used the *Genetic Algorithm* as an optimization process. Due to the similarity of the two models presented so far, it is not surprising that the evolution of *F1-Score* of the two models along the three procedures is similar too. Figure 5.2 illustrates this same evolution about the *Log-Reg* model.

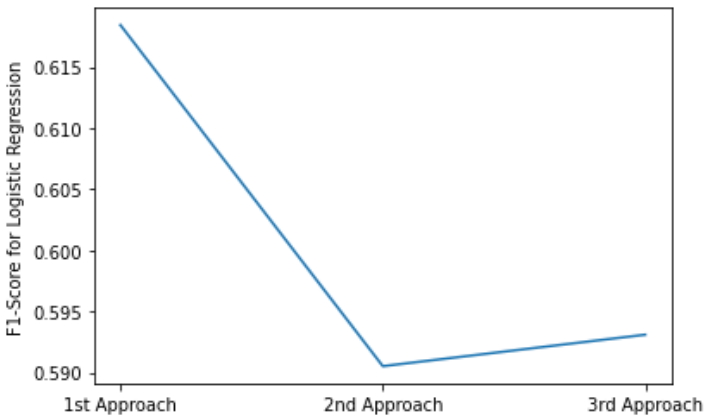


Figure 5.2: *F1-Scores* of the *Logistic Regression* model for each approach.

In the first procedure, the optimization algorithm generated a *Log-Reg* model that reached an *F1-Score* of 62%. Afterwards, in the second methodology, the best model scored close to 59%. With the last approach, the best *F1-Score* obtained was 59%. The *F1-Score* of this model has significantly lowered with the forecast of null values in the second procedure. From the first to the second approach it fell by 3%. Nevertheless, it has relatively decreased less with the selection of *features* in the third methodology, where the score rises 1%. It is possible to attribute to this model conclusions similar to those inferred in the previous section, with the detail that the *Log-Reg* model deals better with more complex *datasets* than the *Lin-Reg* model. Comparing the graph of the *Lin-Reg* results present in Figure 5.1, with the plot of this model, an increase in *F1-Scores* is evident. Table 5.2 outlines the best parameter settings resulting from each procedure.

Approach	Parameter	Value	F1-Score
First	Penalty	L2	0.618
	C	3.675	
	Solver	LibLinear	
Second	Penalty	L2	0.590
	C	1.105	
	Solver	LibLinear	
Third	Penalty	L2	0.593
	C	0.0367	
	Solver	LibLinear	

Table 5.2: Parameter setting of the *Log-Reg* model obtained in each approach.

There is an improvement in the performance of this model in the third approach compared to the second. However, similarly to the previous model, the best result obtained was in the first approach with an *F1-Score* of 62%. The model that achieved this result uses a Penalty equal to L2, C equal to 1,105, and the Solver is LibLinear.

Gaussian Naive Bayes

The only parameter used in this research for its construction is the Variance Smoothing. Nonetheless, due to its mathematical nature, the use of the *Genetic Algorithm* was still necessary. Figure 5.3 shows the evolution of the performance of the *NB* model over the three different approaches.

The analysis of the graph concludes that the model improves with the different approaches, unlike the two models presented so far. In the first approach the *Genetic Algorithm* developed a model that obtained an *F1-Score* of 40%. In the second approach, where the *dataset* has no null values, the best model recorded a score of 48%. In the procedure where the complexity of the *dataset* reduces, the best *F1-Score* achieved was 61%. From the first to the second procedure, there is an improvement in *F1-Score* of approximately 10%. From the second to the last approach occurs an increase of the *F1-Score* of more than 10%. Important notes to be taken with these outcomes

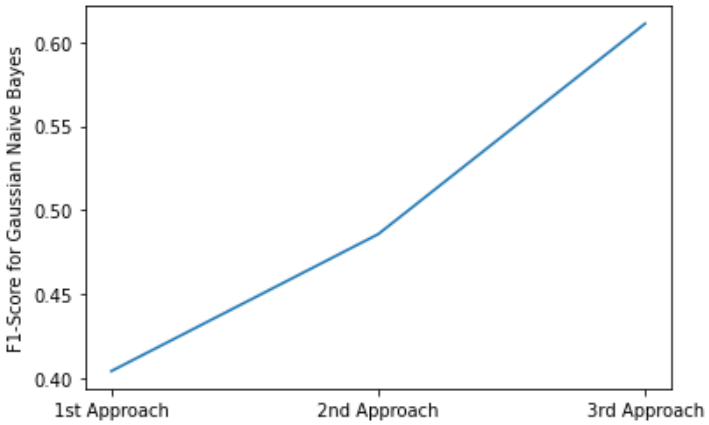


Figure 5.3: *F1-Scores* of the *NB* model for each approach.

is that this model fits better with less complex and better-treated *datasets*. Table 5.3 contains the configuration resulting from the optimization process executed.

Approach	Parameter	Value	F1-Score
First	Variance Smoothing	0.002	0.404
Second	Variance Smoothing	6.268e-05	0.486
Third	Variance Smoothing	0.0356	0.611

Table 5.3: Parameter setting of the *NB* model obtained in each approach.

Despite a less significant start, this model managed to obtain an *F1-Score* of 61% in the *Features Selection* procedure, thus matching the *Lin-Reg* model. The model that obtained this score has a Variance Smoothing of 0.0356

K-Nearest Neighbours

For the model that does not have an explicit training process, Figure 5.4 illustrates the progress of *F1-Score* obtained in each approach through the optimization process. The *KNN* model is one of the fastest to execute all generations of the *Genetic Algorithm*, due to its training operation.

In the first attempt of optimization, the best score was 61%, in the second attempt the best model registered an *F1-Score* of almost 62%, and in the last approach the model generated by the *Genetic Algorithm* reached a score nearly 60% The *KNN* model has improved slightly by 0.5% with the procedure for predicting the null values. However, its *F1-Score* fell by nearly 2% with the *feature* selection approach. This type of results may infer that these types of models do not fit well with the presence of *outliers*, since the null values in this research are -999, and as such, the model improves when these values are no longer present in the *dataset*. The presence of *outliers* in the *dataset* does not have much influence on the performance of the *KNN* model. Nevertheless, it is

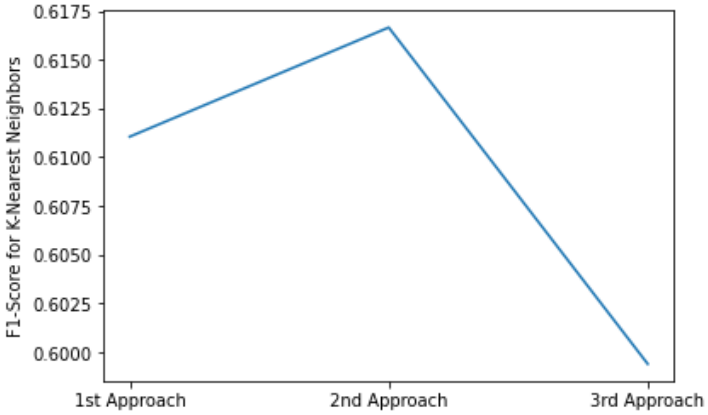


Figure 5.4: *F1-Scores* of the *KNN* model for each approach.

noticeable that the same declines for less complex *datasets*. In Table 5.4 it is possible to find the hyper-parameters provided by the *Genetic Algorithm* during the optimization of this model.

Approach	Parameter	Value	F1-Score
First	Number of Neighbours	13	0.611
	Leaf Size	8	
	Weights	Distance	
	Algorithm	KD Tree	
	P	2	
Second	Number of Neighbours	11	0.617
	Leaf Size	18	
	Weights	Uniform	
	Algorithm	KD Tree	
	P	2	
Third	Number of Neighbours	26	0.599
	Leaf Size	23	
	Weights	Uniform	
	Algorithm	Brute	
	P	1	

Table 5.4: Parameter setting of the *KNN* model obtained in each approach.

Through the analysis of the parameters, it is possible to point an inversely proportional relationship between the Number of Neighbours and *F1-Score*, the higher the number, the lower the score. The best *KNN* model obtained a final *F1-Score* of bout 62%. The best parameter configuration is the one achieved by the second approach and is the Number of Neighbours as 11, the Leaf Size as 18, the Weight function as the Uniform, the Algorithm used as the KD Tree, and the value of P as 2.

Support Vector Machine

As in the model presented above, the *SVM* model showed a slight improvement with the second approach, but relatively lower performance in the latter. Figure 5.5 shows the evolution of the F1-Score attained through the Genetic Algorithm in the three different approaches.

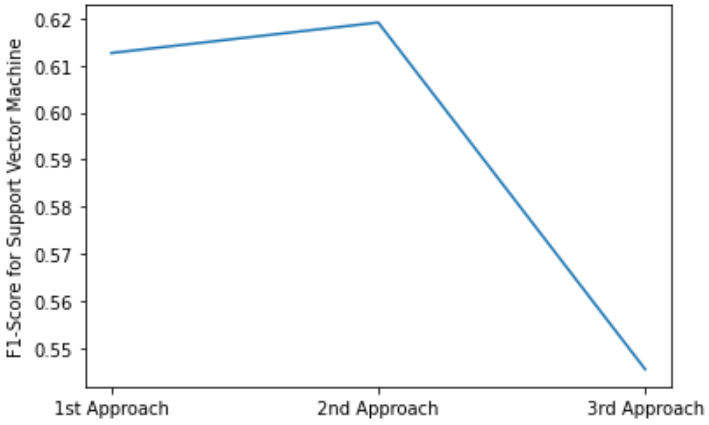


Figure 5.5: *F1-Scores* of the *SVM* model for each approach.

When the *SVM* model went through the optimization process using the initial *dataset*, the best result was 61%. In the second approach, where happens the prediction of the null values, the best model achieved a score of almost 62%. After the *feature* selection, the *Genetic Algorithm* developed a model that reached a *F1-Score* of 55%. During the transition to the second approach, the *F1-Score* obtained in the first approach increases by 1%. Although, in the latter approach, there is a sudden drop of 7%. These factors indicate that this model has a reasonable sensitivity to the presence of *outliers* in the *dataset*, and has a better ability to handle data collections with more *features*. Table 5.5 holds the hyper-parameters and respective values resulting from the optimization process carried out by the *Genetic Algorithm*.

After the execution of the *Genetic Algorithm* for optimizing the *SVM* model, the best *F1-Score* achieved was close to 62%. This score appears during the approach of predicting the null values, and this is the composition of the best model found: C equals 0.741, Gamma equals 0.344, and the Kernel is the RBF.

Decision Tree

This section presents the results obtained for the latest model of the *SL* paradigm. Figure 5.6 shows the F1-Scores achieved after all generations of the *Genetic Algorithm* for the optimization process of the *DT* model. The line shape of this graph is very similar to the line present in the plot of the *SVM* model present in the previous section.

In the first approach, the best *F1-Score* obtained is around 64%. In the second approach, the result attained was 65%, representing an increase of 1% over the previous procedure. For the

Approach	Parameter	Value	F1-Score
First	Kernel	Polynomial	0.613
	Degree	2	
	C	11.428	
	Gamma	0.0530	
Second	Kernel	RBF	0.619
	Degree	4 (*)	
	C	0.741	
	Gamma	0.344	
Third	Kernel	RBF	0.545
	Degree	1 (*)	
	C	1.000	
	Gamma	0.887	

Table 5.5: Parameter setting of the *SVM* model obtained in each approach. The values marked with (*) are ignored by the Kernel.

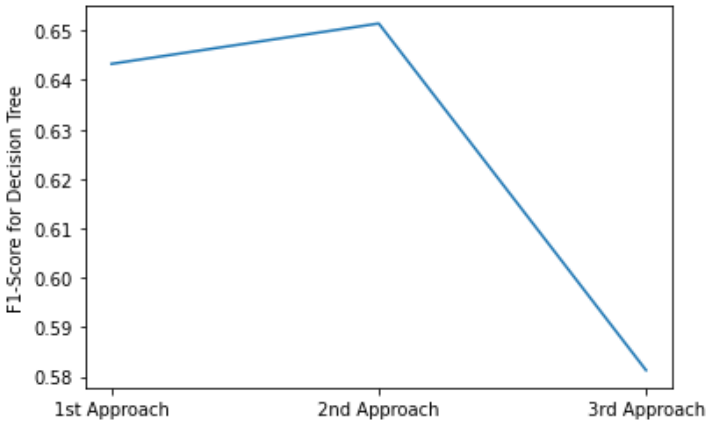


Figure 5.6: *F1-Scores* of the *DT* model for each approach.

latter methodology, the score marks 58%, representing a relatively large drop of 7%. The *F1-Score* variations of the *DT* model are very similar to those observed in the *SVM* model. However, the model presented in this section has higher scores. Based on these scores, it possible to say that this model becomes more efficient when this *dataset* has no null values or *outliers*. However, its performance decreases when the number of *features* of the data collection decreases. Table 5.6 contains the ideal configuration resulting from the optimization process of the *DT* model.

About the table with the parameters, the fact that Criterion in the three approaches is always Entropy is highlighted. Another interesting fact is that Maximum Depth is relatively small compared to the maximum value it could achieve, suggesting that the trees with the best results are relatively balanced. It was during the second approach that the highest *F1-Score*, with a value of 65%, was registered and these are the parameters used for the construction of the respective model: Criterion is Entropy, Splitter is Random, and Maximum Depth is equal to 5.

Approach	Parameter	Value	F1-Score
First	Criterion	Entropy	0.643
	Splitter	Random	
	Maximum Depth	6	
Second	Criterion	Entropy	0.651
	Splitter	Random	
	Maximum Depth	5	
Third	Criterion	Entropy	0.581
	Splitter	Best	
	Maximum Depth	5	

Table 5.6: Parameter setting of the *DT* model obtained in each approach.

Random Forest

Figure 5.7 shows the progress of the *F1-Score* over the three different approaches. There was a relatively high probability that the *RF* model would achieve better results than those observed in the previous section since this model uses the *DT* model as the base estimator. However, this situation did not occur.

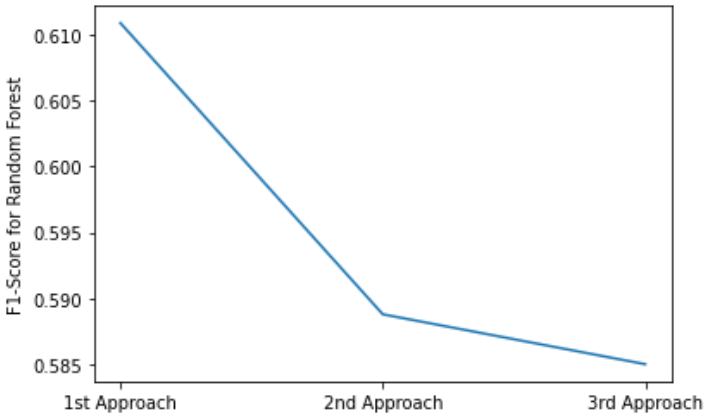


Figure 5.7: *F1-Scores* of the *RF* meta-model for each approach.

The graph shows that, unlike its base model, the progress of this meta-model declines over the approaches. The *RF* model obtained an *F1-Score* of 61% with the *dataset* initially processed. After replacing the null values, the meta-model reached a score of 59%, indicating a decrease in its performance of 2%. Finally, after selecting the most important features, the optimization algorithm generated a model that reached an *F1-Score* of 58%, decreasing even more 1% in comparison with the previous approach. After analyzing the data present in the graph, it is possible to indicate that this meta-model becomes less efficient when this data collection decreases its number of columns. However, its performance is higher when the *dataset* has the presence of null values. Table 5.7 contains the parameter composition of the best models for each procedure.

Approach	Parameter	Value	F1-Score
First	Number of estimators	603	0.611
	Criterion	Gini	
	Maximum Depth	20	
	Maximum Number of Features	22	
	Minimum Sample Split	8	
	Minimum Samples Leaf	8	
	Bootstrap	False	
	Number of Maximum Samples	1112 (*)	
Second	Number of estimators	119	0.589
	Criterion	Gini	
	Maximum Depth	22	
	Maximum Number of Features	7	
	Minimum Sample Split	10	
	Minimum Samples Leaf	8	
	Bootstrap	False	
	Number of Maximum Samples	989 (*)	
Third	Number of estimators	1707	0.585
	Criterion	Gini	
	Maximum Depth	8	
	Maximum Number of Features	5	
	Minimum Sample Split	87	
	Minimum Samples Leaf	8	
	Bootstrap	False	
	Number of Maximum Samples	199 (*)	

Table 5.7: Parameter setting of the *RF* model obtained in each approach. The values marked with (*) are not used because Bootstrap is False

By looking at the table, one fact emerges that it is the Criterion parameter in all three approaches to be the Gini, which is the opposite of the *DT* model. In comparison with the base model, in this case, Maximum Depth has risen slightly, which may indicate that several trees in the forest can be unbalanced. These two factors may be at the origin of the reason why the meta-model obtained lower results than the base estimator. Another indicator to note is that in all three approaches, the most suitable models all have the Bootstrap parameter as False, which indicates that the models that trained with the full *dataset*, had better results than the models that practised with samples. The best model achieved a score of 61%, and this is parameter configuration: the Number of estimators equals to 603, the Criterion is Gini, the Maximum Depth equals to 20, the Maximum Number of Features equals to 22, the Minimum Sample Split equals to 8, and the Minimum Samples Leaf equals to 8.

Ada Boost

This dissertation closes the *SL* paradigm with the results acquired after the *AdaBoost* meta-model optimization process. Figure 5.8 depicts the curve showing the development of *F1-Score* along with the different approaches.

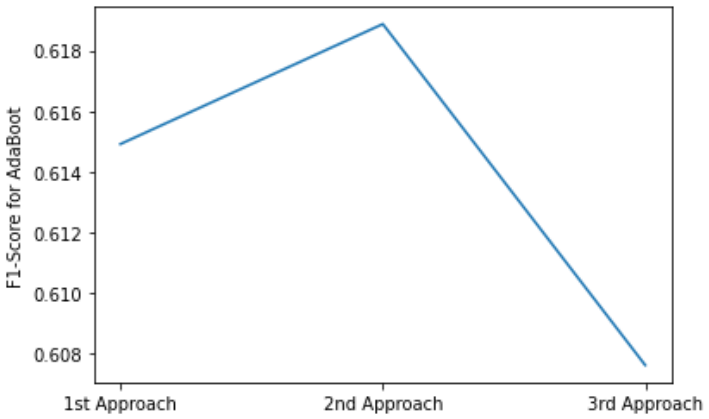


Figure 5.8: *F1-Scores* of the *AdaBoost* meta-model for each approach.

In the first approach, the basic estimator used is the *DT* model. The reason for this decision is that this was the model with the highest *F1-Score* within this first learning paradigm. With the *dataset* initially processed, the optimization process generated a meta-model which attained a score of almost 62%. In the second procedure, the basic estimator used is again the *DT* model, as it was this model that achieved the best performance when it came to models of the *SL* paradigm. By replacing the null values, the *Genetic Algorithm* has developed a model that has reached a level of 62%. In the last strategy, the basic estimator used is the *NB* model, as this was the model with the best performance. After removing the less relevant columns from the data collection, the chosen model obtained an *F1-Score* of 61%, thus lowering its score by 1% compared to the other two approaches. Both the second and the third approach failed to change the performance of the meta-model, and on neither occasion did the *AdaBoost* model achieve a better *F1-Score* than the base model. Table 5.8 shows the parameters and respective values that compose the models elected after the optimization process.

After a study of the table, a fact emerges that is a worthy mention. The meta-model with the lowest Learning Rate is also the meta-model with the lowest number of estimators. The best meta-model appears in the second approach, with a F1-Score of 62%, and this is its list of parameters: the Base Model is the *DT* model, the Number of estimators equals 1311, and the Learning Rate equals 0.006.

Approach	Parameter	Value	F1-Score
First	Base Model	Decision Tree	0.615
	Number of estimators	1580	
	Learning Rate	0.009	
Second	Base Model	Decision Tree	0.619
	Number of estimators	1311	
	Learning Rate	0.006	
Third	Base Model	Gaussian Naive Bayes	0.608
	Number of estimators	81	
	Learning Rate	3.712e-4	

Table 5.8: Parameter setting of the *AdaBoost* model obtained in each approach.

5.1.2 Unsupervised learning

Having concluded the *SL* paradigm, this document presents a reflection on the results obtained in *UL*. As previously done, there is a section for each of the four models studied, *K-Means, AC, BIRCH* and *SC*.

K-Means

The *F1-Scores* of the *K-Means* model are the first results of this learning paradigm. Figure 5.9 illustrates the progress of the optimization of this model through the three different approaches.

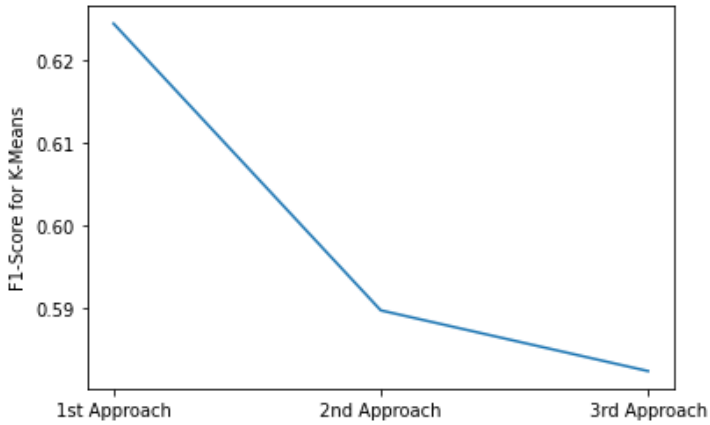


Figure 5.9: *F1-Scores* of the *K-Means* model for each approach.

In the first approach, where the *dataset* is the one processed during Chapter 3, the best model obtained an *F1-Score* of 62%. In the second approach, where the data collection has its null values replaced, the *Genetic Algorithm* generated a model that reached 59%, thus registering a decrease of 3%. In the last approach, where the most important *features* are selected, the optimization process developed a model that obtained a score of 58%, decreasing its result again by 1%. The review of the data present on the plot allows to state that this model is more efficient when the number of

columns in this data collection is higher and holds null values. Table 5.9 contains all the parameters that allow the construction of the ideal models by approach.

Approach	Parameter	Value	F1-Score
First	Tolerance	1.492e-05	0.624
	Algorithm	Elkan	
Second	Tolerance	4.962e-08	0.590
	Algorithm	Full	
Third	Tolerance	2.462e-09	0.582
	Algorithm	Elkan	

Table 5.9: Parameter setting of the *K-Means* model obtained in each approach.

Thanks to the data in the table, it is possible to indicate a directly proportional relationship between the result obtained by the best model and the value of its Tolerance, since for higher scores, the higher the Tolerance value is. The model with the highest *F1-Score* was obtained in the first approach and achieved a score of 62%, and this is its parameter configuration: the Tolerance value is equal to 1.492e-05, and the clustering Algorithm is the Elkan.

Agglomerative Clustering

This section introduces the results obtained by the first hierarchical clustering model and is during the optimization of this model that occurs the lowest value of *F1-Score* in all this research. Figure 5.10 depicts the evolution of the optimization performed on the *AC* model.

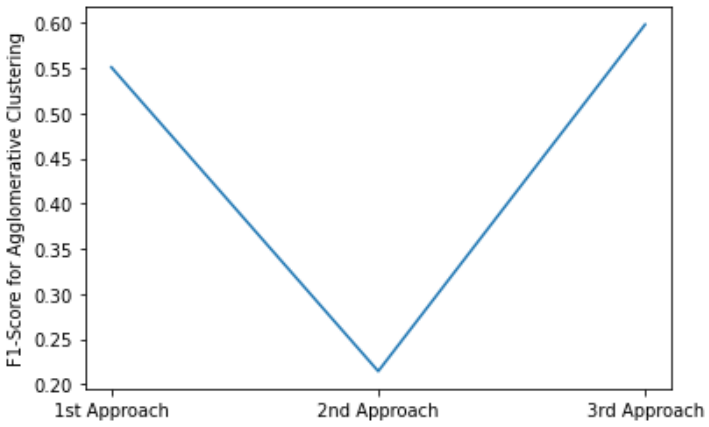


Figure 5.10: *F1-Scores* of the *AC* model for each approach.

During the first approach, the *Genetic Algorithm* generated a model that obtained a score of 55%. With the second procedure, the optimization process created a model that achieved an *F1-Score* of 21%. This outcome represents the lowest result of this research, and an increase of 34% compared to the previous approach. With the last methodology, the chosen model achieved a score close to

60%, thus increasing its value by 39% compared to the second approach. Based on these results, it is possible to conclude that it is this model which has suffered the most declining in its performance from the prediction of the null values. However, it is the model that has benefited the most from the selection of the most important columns. Table 5.10 presents the parameter settings of the best models obtained by the optimization process.

Approach	Parameter	Value	F1-Score
First	Linkage Affinity	Complete Cosine	0.551
Second	Linkage Affinity	Complete Euclidean	0.215
Third	Linkage Affinity	Average Manhattan	0.598

Table 5.10: Parameter setting of the *AC* model obtained in each approach.

The *Genetic Algorithm* used in the third approach generated the best model, its *F1-Score* is about 60%, and this is its parameter setting: the Linkage criterion is Average, and the Affinity metric is Manhattan.

BIRCH

The results obtained during the optimization process of the *BIRCH* model are presented in this section. Figure 5.11 depicts the evolution of the *F1-Score* obtained at the end of all generations of the *Genetic Algorithm*. This model is another example, like the *NB* model, which has benefited from the realization of the two additional approaches.

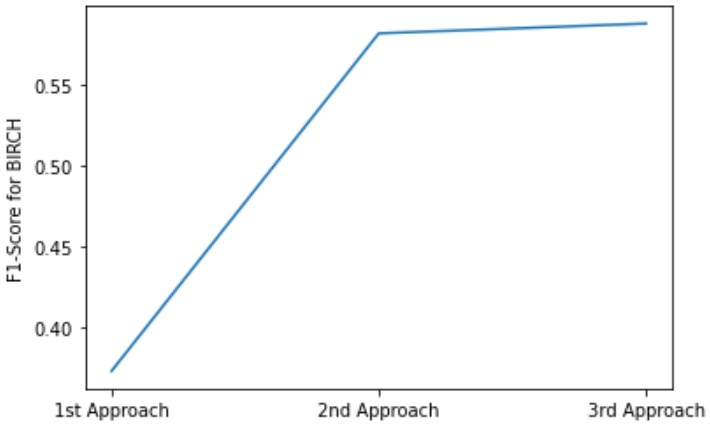


Figure 5.11: *F1-Scores* of the *BIRCH* model for each approach.

The optimization process carried out in the first approach, generated a *BIRCH* model that obtained a score of 37%. With the null values forecast in the second approach, the *Genetic Algorithm*

developed a model that reached an *F1-Score* of 58%, thus registering a relatively high increase of 21%. In the last strategy, where the less important columns of the *dataset* are eliminated, the best model attained a score of about 59%, thus rising another 1% compared to the previous approach. By examining these results, it is possible to conclude that the *BIRCH* model adapts much better when there are no null values in the data collection, and when the its complexity is more reduced. Table 5.11 provides the configurations of the hyper-parameters of the best models obtained in each approach.

Approach	Parameter	Value	F1-Score
First	Threshold	0.875	0.373
	Branching Factor	150	
	Compute Labels	True	
Second	Threshold	0.253	0.581
	Branching Factor	123	
	Compute Labels	True	
Third	Threshold	1.000	0.588
	Branching Factor	16	
	Compute Labels	True	

Table 5.11: Parameter setting of the *BIRCH* model obtained in each approach.

Through an analysis of the table, it is possible to conclude that in all approaches, the selected model after the optimization process always has the Compute Label as True. The best model has an *F1-Score* of 59% and was generated during the optimization process after removing the least relevant columns, and this is its constitution: Threshold is about 1, Branching Factor is 16, and Compute Labels is True.

Spectral Clustering

Finally, this dissertation presents the results of the latest hierarchical model and the latest model of the *UL* paradigm. The graph that appears in Figure 5.12 demonstrates the attained performances.

With the *dataset* initially processed, the *Genetic Algorithm* developed a model that reached an *F1-Score* of 68%. In the second approach, where null values are replaced, the dominant model registered a score of almost 68%, maintaining its amount in comparison to the first strategy. With the selection of the most relevant *features*, the best model also recorded a score of almost 68%, remaining its value in comparison to the other two approaches. In short, the performance of the *SC* model did not suffer many variations with the different treatments performed to the data collection. However, it was one of the models with the best results in this research. Table 5.12 shows the set of parameters used to the modelling of the best estimators.

It is possible to indicate that in the three approaches, the best model utilises K-Means as the strategy to use to assign labels in the embedding space. The best model obtained an *F1-Score*

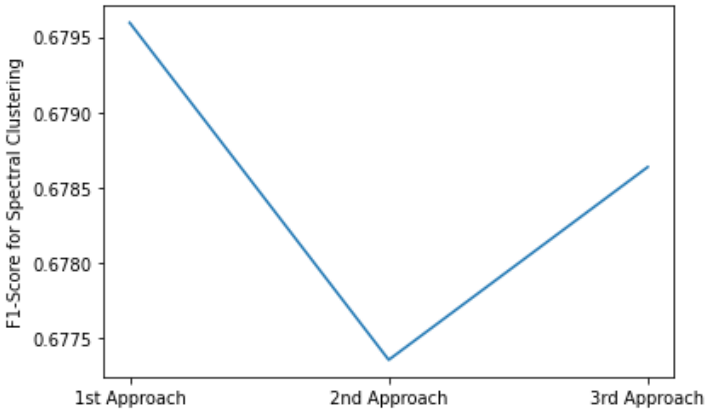


Figure 5.12: *F1-Scores* of the *SC* model for each approach.

of 67.96%, it was developed in the first approach, and this is its set of parameters: the Affinity is RBF, the Eigen Solver is Arpack, the Number of Components is equal to 48, the Gamma is equal to 0.154, the Eigen Tolerance is equal to 9.047e-05, the Assign Labels is the K-Means, the Degree is equal to 2, and Coefficient is equal to 10.

5.1.3 Semi-supervised learning

Below are two sections on the two models belonging to the hybrid paradigm, *LS* and *LP*.

Label Propagation

The first section presents the results of the optimization process of the first model of this paradigm. Figure 5.13 represents the progress of the *F1-Score* of the *LP* model through the *Genetic Algorithm* execution in the three different approaches.

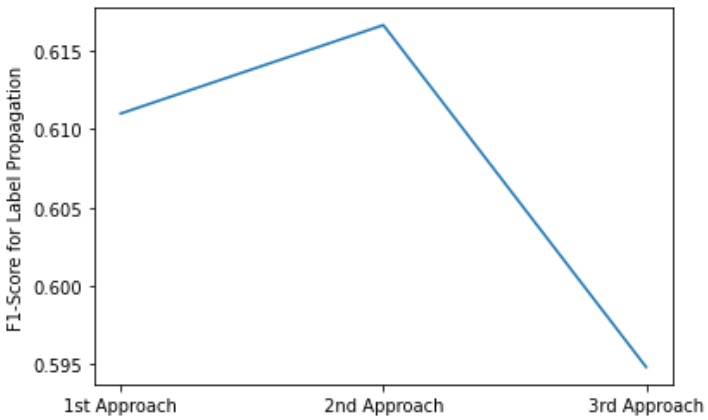


Figure 5.13: *F1-Scores* of the *LP* model for each approach.

In the first approach, the optimization process developed a model that achieved a score of 61%. With the second data processing, the *Genetic Algorithm* generated a model that got a rating close

Approach	Parameter	Value	F1-Score
First	Affinity	Nearest Neighbours	0.680
	Eigen Solver	None	
	Number of Components	84	
	Gamma	0.022 (*)	
	Number of Neighbours	14	
	Eigen Tolerance	2.228e-4	
	Assign Labels	K-Means	
	Degree	1 (*)	
	Coefficient	9 (*)	
Second	Affinity	Nearest Neighbours	0.677
	Eigen Solver	Arpack	
	Number of Components	6	
	Gamma	1.143e-4 (*)	
	Number of Neighbours	19	
	Eigen Tolerance	0.038	
	Assign Labels	K-Means	
	Degree	6 (*)	
	Coefficient	8 (*)	
Third	Affinity	RBF	0.679
	Eigen Solver	Arpack	
	Number of Components	48	
	Gamma	0.154	
	Number of Neighbours	16 (*)	
	Eigen Tolerance	9.047e-05	
	Assign Labels	K-Means	
	Degree	2	
	Coefficient	10	

Table 5.12: Parameter setting of the *SC* model obtained in each approach. The values marked with (*) are ignored by the Affinity.

to 62%, so with the null values predicted the model registered a 1% increase in its score. With the last strategy, the chosen model reached an *F1-Score* of 59%. So with the reduction of the *dataset* columns, the model registered a 3% decrease. With these results, it is possible to conclude that this hybrid model obtained the best outcome with the *dataset* without null values. However, the selection of the most relevant *features* lowered the performance of the model. Table 5.13 contains the parameters obtained by the optimization algorithm in each approach.

In all three approaches, the Kernel chosen is KNN. Interestingly, the same *LP* model that the optimization process generated in the first approach is the same generated in the third. However, it was with the second strategy that the best model was developed, which achieved a score of 62% and these are its parameters: the Kernel is KNN, the Number of Neighbours is equal to 15, and Tolerance is equal to 0.548.

Approach	Parameter	Value	F1-Score
First	Kernel	KNN	0.611
	Gamma	0.061 (*)	
	Number of Neighbours	21	
	Tolerance	0.248	
Second	Kernel	KNN	0.617
	Gamma	0.009 (*)	
	Number of Neighbours	15	
	Tolerance	0.548	
Third	Kernel	KNN	0.595
	Gamma	0.061 (*)	
	Number of Neighbours	21	
	Tolerance	0.248	

Table 5.13: Parameter setting of the *LP* model obtained in each approach. The values marked with (*) are ignored by the Kernel.

Label Spreading

This document concludes the *SSL* paradigm with the results obtained by the *LS* model. Figure 5.14 presents the progress of the scores obtained by the best models generated by the optimization process in each approach.

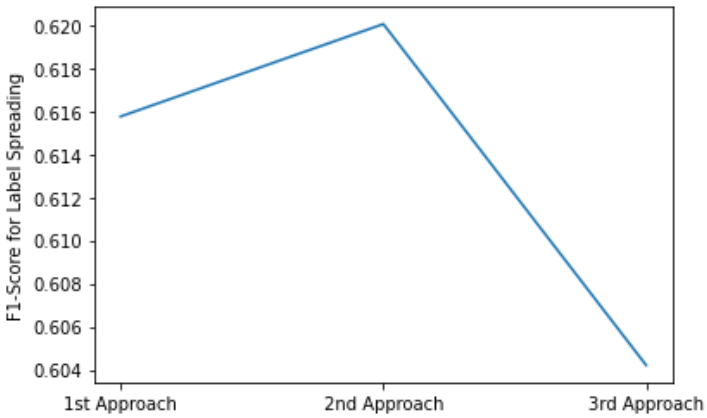


Figure 5.14: *F1-Scores* of the *LS* model for each approach.

From the graph analysis, the first fact that stands out is that the behaviour of this model is very identical to the previous model, which makes sense since its implementation is very similar. With the initial preparation made to the *dataset*, the *Genetic Algorithm* developed a model that achieved a score close to 62%. With the second approach, the optimization process generated a model that obtained an *F1-score* of 62%, indicating a slight increase as can be seen by the graph curve. In the last methodology, the chosen model registered a decrease in its performance of 2%, that is, it obtained a score of 60%. With these results, it is possible to conclude that this model is slightly

better than the other hybrid model. The origin behind the value of the second approach can be due to a greater sensitivity to the presence of null values. The result in the third strategy can indicate that the *LS* model adapts best when the data collection has a larger number of columns. Table 5.14 shows the parameter values obtained in each optimization process.

Approach	Parameter	Value	F1-Score
First	Kernel	KNN	0.616
	Gamma	0.301 (*)	
	Number of Neighbours	13	
	Tolerance	0.514	
	Alpha	0.067	
Second	Kernel	KNN	0.620
	Gamma	0.230 (*)	
	Number of Neighbours	13	
	Tolerance	0.514	
	Alpha	5.484e-07	
Third	Kernel	KNN	0.604
	Gamma	0.016 (*)	
	Number of Neighbours	21	
	Tolerance	0.780	
	Alpha	0.406	

Table 5.14: Parameter setting of the *LS* model obtained in each approach. The values marked with (*) are ignored by the Kernel.

From the table report, it is possible to indicate that, as in the previous model, the Kernel in the three different approaches is KNN. The configuration obtained in the first procedure is very similar to the one obtained in the second, with the execution of the Alpha parameter value. As the best score appears in the second approach, the Alpha parameter may play a more relevant role than the others, thus justifying the difference in results, and also the reason why this model is slightly higher than the other hybrid model presented. The best model obtained an *F1-Score* of 62%, and these are its parameters: the Kernel is the KNN, the Number of Neighbours is equal to 13, the Tolerance is equal to 0.514, and the Alpha value is equal to 0.067.

5.1.4 Deep learning

Finally, it follows the latest learning paradigm, which contains the most promising results. To maintain the writing coherence of this document, although there is only one model in this paradigm, it follows a section containing the *F1-Scores* graph obtained in each approach, as well as the table with the neural network configurations obtained during the optimization process.

Artificial Neural Network

This section contains the latest analysis of the results obtained in this dissertation. In Figure 5.15 lays the graph that shows the evolution of the performance of the models obtained by the optimization process, through the different procedures.

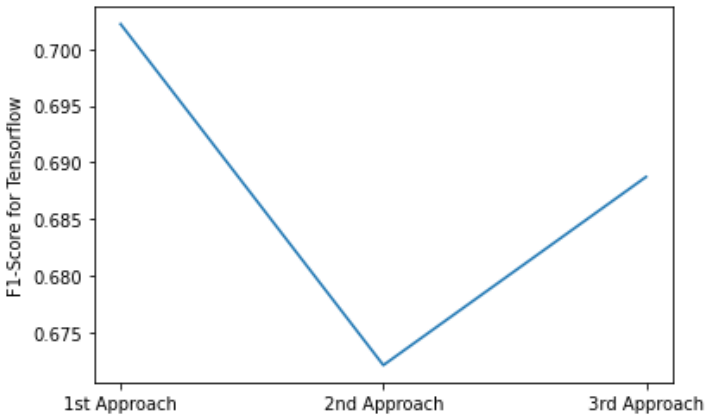


Figure 5.15: *F1-Scores* of the *ANN* model for each approach.

The *Genetic Algorithm* generated a model that scored 70% in the first approach. This result is the best obtained in all this research. In the second procedure, the optimization process developed a model that achieved an *F1-Score* of 67%, thus registering a 3% decrease in performance. In the last strategy, the chosen model reached a score close to 69%, thus increasing 2% compared to the previous approach. With these results, it is possible to reflect that this model adapts better with the presence of null values in the *dataset*, and with a larger number of *features*, not taking advantage of the two approaches performed. Table 5.15 shows the parameters and their respective values, which are essential to the modelling of the networks in each strategy.

Approach	Parameter	Value	F1-Score
First	Layers	[32, 16, 16, 8, 32]	0.702
	Dropouts	[0.494, None, 0.460, 0.387, None]	
	Optimizer	Nadam	
	Learning Rate	0.009	
Second	Layers	[32, 32, 32, 2]	0.672
	Dropouts	[0.453, 0.318, None, 0.318]	
	Optimizer	Nadam	
	Learning Rate	0.007	
Third	Layers	[4, 16, 2]	0.689
	Dropouts	[None, None, 0.460]	
	Optimizer	Adam	
	Learning Rate	0.009	

Table 5.15: Parameter setting of the *ANN* model obtained in each approach.

With the data shown in the table, it is possible to conclude that there may be a relationship between the number of columns in the *dataset*, and the number of layers in the neuronal network, because the optimal model found in the third approach, is the smallest. Although the best result appears with the most complex model, the worst result does not emerge with the model with lesser layers. This fact invalidates the idea that the more layers a network can have, the better its performance will be. The best model appears in the approach where the *dataset* is in its initial processing state. The final *F1-Score* is 70%, and this is the composition of the network that achieved it: The first layer consists of 32 nodes, followed by a Dropout with 0.494. The second and third layers are composed of 16 nodes, and after the third layer is a Dropout with 0.460. The fourth layer has 8 nodes, followed by a Dropout of 0.387. The penultimate layer has 32 nodes, and the last one has only one exit node. The Optimizer is Nadam, and its Learning Rate is 0.009. The optimal model trained with 10 Epochs, the Batch Size equals 8, and the loss function is Binary Crossentropy.

5.2 SUMMARY

Lin-Reg, Log-Reg, RF, K-Means, SC and *ANN* models attained the best results when trained with *dataset* in their initial processing state. *KNN, SVM, DT, AdaBoost, LS* and *LP* models achieved the best performance after replacing null values with the predictions made by the chosen regressors in Section 4.1.3.2. The *NB, AC* and *BIRCH* models generated the best model after the selection of the most relevant *features* by the *GB* meta-model. Although meta-models are an optimization process in their own right, in this research, it has never happened to obtain better performances than their respective base estimators. The interval of *F1-Scores* ranging from 60% to 62% encompasses the outcomes achieved in 11 of the 15 models studied in this research. The *BIRCH* model obtained the worst result of 59%. The *DT* model lays in the third position of the best models, with a score of 65%. The *SC* model stays in the second place, with an *F1-Score* of 67%. The best model in this research is the *ANN*, with a score of 70%. Naturally, this was the model chosen to be incorporated into the final application. Table X contains the overall results obtained during this research.

5.3 TAEDIUM

This dissertation ends with the presentation of the final product, which this investigation culminates in. Figure 5.16 is the logo created for the application that will predict the boredom state of the person using the smartphone. The name chosen for this application is *TaeDIUM* which derives from the Latin word that gives origin to the term annoyance. The core of this application is very similar to the application used in data collection, with some differences in the way it works. This application starts its evaluation period every time the screen is activated, the evaluation time is also 5 minutes, just like it is in the first application. After the 5 minutes mentioned, *TaeDIUm* analyses

Model	Approach	F1-Score
ANN	First	0.702
SC	First	0.680
DT	Second	0.651
K-Means	First	0.624
LS	Second	0.620
AdaBoost	Second	0.619
SVM	Second	0.619
Log-Reg	First	0.618
KNN	Second	0.617
LP	Second	0.617
RF	First	0.611
NB	Third	0.611
Lin-Reg	First	0.608
AC	Third	0.598
BIRCH	Third	0.588

Table 5.16: Overall results order by descending F1-Score.

the data collected during this time interval, and using its incorporated *ANN* to infer whether the person using the smartphone is bored or not. The *ANN* was pre-trained and deployed using the *TensorFlow Lite* framework. The application triggers a notification with recommended content if the prediction made by the neuronal network is higher than 50%. In this way, there is the possibility that by stimulating the person, the feeling of bored may disappear. The recommended content could be something like a new series that debuts on Netflix, or a new product on Amazon, thus opening a window here to monetize the project.

Figure 5.16: *TaeDIUM* Logo.

One of the objectives of this research was to guarantee the possibility of calibrating a model only with the data generated by the cell phone. At the time of writing this dissertation, *Tensorflow* has not yet developed a way to train a neural network incorporated into a mobile phone. However, during the implementation of this application, there were some measures to make possible this goal in the future. Whenever a recommendation notification disappears from the smartphone status bar, the

application registers an *entry* in a database. The structure of the new *entry* is similar to the *dataset entries* used in this investigation, with only two differences. This time, it is the neural network that provides the *feature* that indicates the level of boredom and not the user. The other difference is the existence of a new column, which designates if the user clicked on the notification (receiving the value 2), if removed it from the status bar (getting the value 1), or if the network predicted that the person was not bored (obtaining the value 0). So in the future, when the training process is available, the neural network will know when it failed and when it got it right.

To better explain the architecture of this app, it is necessary to have the notion that it is divided into six parts. Each part is composed of several smaller elements, with more specific and atomic functionalities.

- **Interface Activities**: this layer represents all the user interfaces with the application. The number of these interfaces is four. One interface is the central interface that allows navigation to others. One interface that explains all the sensors measured during the evaluation period, and two interfaces for the visualization of the data from these same sensors for the last evaluation periods;

- **Sensor Listeners**: the group of modules responsible for measuring various physical aspects of real life. Using the hardware sensors present in a smartphone, these listeners can measure the brightness of the environment, gravity, magnetic fields, among others;

- **Broadcast Receivers**: the set of components with the function to detect virtual events during the evaluation period. The events can be outgoing calls, incoming text messages, screen activation number, Bluetooth connection status, among others;

- **Background Service**: the functions of this component are the same as in the first application. Manages the evaluation period, collects all values from Sensor Listeners and signals collected by Broadcast Receivers. However, before presenting the new record to the Database Handler, perform the same techniques present in Section 3.3. Subsequently, it provides this processed input to the *ANN* model, so that it can predict whether the person is bored or not;

- **Database Handler**: is the part of the application that establishes the connection with the database. This Handler determines the structure of the records, along with the name and type of each *feature*. Send the *entries* to the database for the formation of the new *dataset*;

- **Artificial Neural Network**: it is the best model obtained in this chapter, incorporated in the application to perform the prediction of whether the person using the smartphone is boredom or not.

Along with this textual explanation, it also follows Figure 5.17, which shows a scheme of *TaeDIUM* architecture, in an attempt to give the reader help to understand and memorize.

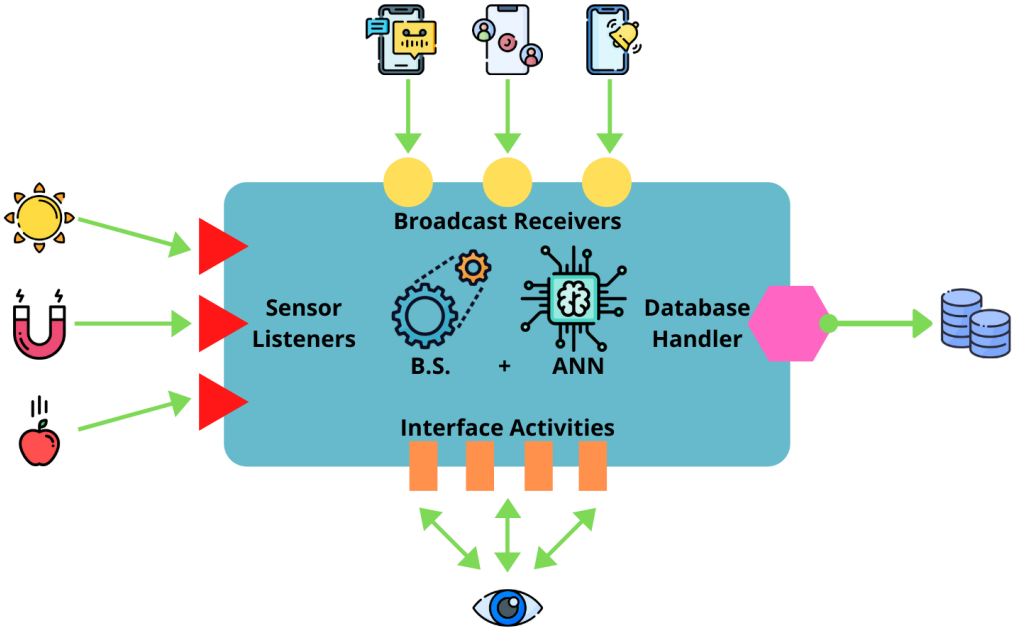


Figure 5.17: *TaeDIUM* Architecture.

Figure 5.18 shows a notification triggered when the *ANN* identified boredom in the person interacting with the smartphone after the evaluation period.

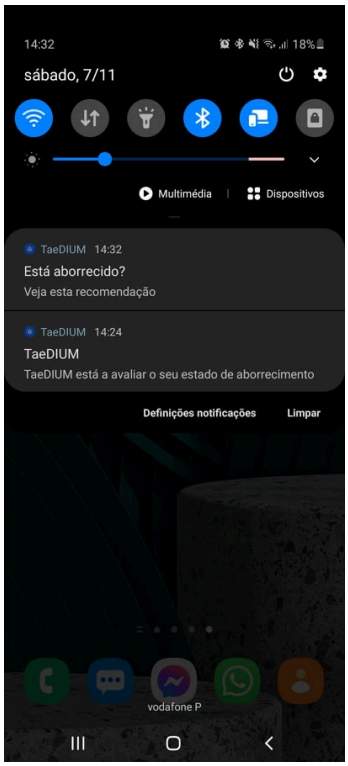


Figure 5.18: The suggestion of the *Taedium* of content at a moment of boredom.

6. CONCLUSIONS AND FUTURE WORK

The most important conclusion to be drawn from this research is that it is possible to predict the boredom status of a person through the respective smartphone, using an *ML* approach. A concern that has always been present in this study was the modelling of many models, present in various learning paradigms. Not assuming whether this problem would be suitable for *SL* or *UL* models has reduced the possibility of the existence of a better method to achieve this goal.

Several aspects could have a different approach to increase the quality of the *dataset*. More participants would lead to an even richer diversity of data than the one that is present in the used *dataset*. However, around 70 participants are already a good adhesion given the sensitivity of the society of today to the existence of applications that collect such sensitive data. The data collection period was from 1st November to 29th February, and to increase the number of *entries* of the *dataset*, the time interval would have to last longer. Still, this period represents 1/4 of the time window for this research. There is an interesting detail regarding this time interval and lays in the distribution charts of the "Day of the Month" *features* present in Appendix A. If the reader notices in the 25th day of the month, there are few responses with a boredom status level higher than 50. As there were only three days on the 25th day of the month in the *dataset*, and one of them is Christmas day, this means that fortunately, people on Christmas day do not feel so bored. The possibility of reopening another data collection period during the quarantine has been considered. This same hypothesis cancelled because the feeling of boredom could surge by the lockdown situation itself, and thus cause a certain level of *bias* in the data.

Both the first and the second application developed in this study, have an evaluation time that lasts 5 minutes. This amount of time seems to be an appropriate time interval for the case study. However, it would be better to execute a process that could give some scientific support to elect the duration of this evaluation time. The representation of the notifications could be different, such as the creation of a column for each of the most used mobile applications today. Later, during the *feature* engineering phase, these *features* could also be categorized by social networks, chatting, among others. The main reason for this approach is that some models could take advantage of knowing that there is no notification from *Facebook* during the evaluation period, or that there are more than 50 messages to read on *WhatsApp*. Android provides means to collect data about other physical sensors on a mobile phone, such as the Linear Acceleration sensor and the Rotation Vector. Although, the *SDK* version of *BoredomApp* would have to increase and thus decrease the number of smartphone models that the application would be operational.

The *Genetic Algorithm* is a process that in this study tried to find a maximum value in the space of parameters configuration possibilities to develop *ML* models. In the field of mathematics, it is always a concern that methods like these find local maximums instead of the global one. One way to avoid that the optimization algorithm would stop at a local maximum would be to insert random *Chromosomes* in each generation. So these new individuals could find other spaces that could get better results and then reproduce themselves by passing their genetic information on to subsequent

generations. This solution would bring an increase in the consumption of computational resources and consequently, an increase in the execution time of the model optimization process. Then it would lead to a need for using more powerful and efficient technology. The model that obtained the largest *F1-Score* was the *ANN* model implemented by the *Tensorflow* library. Two parameters are necessary for the construction of the model that are not present in the *Chromosome* composition, which is the number of Epochs and the Batch Size. The reason for this decision is because these two parameters have some sensitivity, as shown in Section 4.2.4.1, and it would be harder to take into account during the operation of the *Genetic Algorithm*.

Besides the *F1-Scores* presented in this dissertation, the *Accuracy* values of all models are also known. However, *F1-Score* is a learning metric that better indicates the performance of a model concerning false positive and false negative predictions. The application triggers a notification to entertain a person when the person is not looking for stimulation, has a more reduced use, and this is the main reason for the preference of *F1-Score* to *Accuracy*. One conclusion to be drawn from the results of the *AdaBoost* and *RF* meta-models is that only the optimization provided by the *Genetic Algorithm* is sufficient, since the base estimators, when optimized by this operator, obtained better results than the meta-models that use them.

This research has also allowed the development of a scientific article in which it reports some of the most relevant results presented here. This article was presented at the 9th *European Starting AI Researchers' Symposium (STAIRS)* in the *European Conference on Artificial Intelligence (ECAI)* held remotely from Santiago de Compostela, Spain.

For future work, as already mentioned in Section 5.3, where possible, train the neural network incorporated in *TaeDIUM* using data relating only to the mobile phone in which it is incorporated, and thus make the model more personalized to the person using the application.

REFERÊNCIAS

Alfaro, E., García, N., Gámez, M., e Elizondo, D. (2008). Bankruptcy forecasting: An empirical comparison of adaboost and neural networks. *Decision Support Systems*, 45(1):110–122.

Allen, L. K., Mills, C., Jacovina, M. E., Crossley, S., D'mello, S., e McNamara, D. S. (2016). Investigating boredom and engagement during writing using multiple sources of information: the essay, the writer, and keystrokes. In *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge*, pages 114–123.

Almeida, A. M. d., Castel-Branco, M. M., e Falcao, A. (2002). Linear regression for calibration lines revisited: weighting schemes for bioanalytical methods. *Journal of Chromatography B*, 774(2):215–222.

Ayodele, T. O. (2010). Types of machine learning algorithms. In *New advances in machine learning*. IntechOpen.

Bergler, E. (1945). On the disease-entity boredom ("alysosis") and its psychopathology. *Psychiatric Quarterly*, 19(1):38–51.

Bixler, R. e D'Mello, S. (2013). Detecting boredom and engagement during writing with keystroke analysis, task appraisals, and stable traits. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 225–234. ACM.

Bogomolov, A., Lepri, B., Ferron, M., Pianesi, F., e Pentland, A. S. (2014). Daily stress recognition from mobile phone data, weather conditions and individual traits. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 477–486. ACM.

Bogomolov, A., Lepri, B., e Pianesi, F. (2013). Happiness recognition from mobile phone data. In *2013 International Conference on Social Computing*, pages 790–795. IEEE.

Bowd, C., Weinreb, R. N., Balasubramanian, M., Lee, I., Jang, G., Yousefi, S., Zangwill, L. M., Medeiros, F. A., Girkin, C. A., Liebmann, J. M., et al. (2014). Glaucomatous patterns in frequency doubling technology (fdt) perimetry data identified by unsupervised machine learning classifiers. *PLoS One*, 9(1).

Brownlee, J. (2019). How to use learning curves to diagnose machine learning model performance.

Burbidge, R. e Buxton, B. (2001). An introduction to support vector machines for data mining. *Keynote papers, young OR12*, pages 3–15.

Chaudhari, P. e Agarwal, H. (2017). Progressive review towards deep learning techniques. In *Proceedings of the International Conference on Data Engineering and Communication Technology*, pages 151–158. Springer.

Close, A. G. e Kukar-Kinney, M. (2010). Beyond buying: Motivations behind consumers' online shopping cart use. *Journal of Business Research*, 63(9-10):986–992.

Dai, Q. (2013). A competitive ensemble pruning approach based on cross-validation technique. *Knowledge-Based Systems*, 37:394–414.

Eastwood, J. D., Frischen, A., Fenske, M. J., e Smilek, D. (2012). The unengaged mind: Defining boredom in terms of attention. *Perspectives on Psychological Science*, 7(5):482–495.

El Naqa, I. e Murphy, M. J. (2015). What is machine learning? In *Machine Learning in Radiation Oncology*, pages 3–11. Springer.

Gao, Y. e Gao, F. (2010). Edited adaboost by weighted knn. *Neurocomputing*, 73(16-18):3079–3088.

Guo, Q., Agichtein, E., Clarke, C. L., e Ashkan, A. (2009). In the mood to click? towards inferring receptiveness to search advertising. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 319–324. IEEE.

Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516.

Hartigan, J. A. e Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108.

Hassan, N. e Akamatsu, N. (2004). A new approach for contrast enhancement using sigmoid function. *Int. Arab J. Inf. Technol.*, 1:221–225.

Ikotun Abiodun, M., Lawal Olawale, N., e Adelokun Adebowale, P. (2011). The effectiveness of genetic algorithm in solving simultaneous equations. *International journal of computer applications*, 975:8887.

Iscen, A., Tolias, G., Avrithis, Y., e Chum, O. (2019). Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5070–5079.

Johnson, G. e Rajamani, R. (2020). Smartphone localization inside a moving car for prevention of distracted driving. *Vehicle system dynamics*, 58(2):290–306.

Junker, M., Hoch, R., e Dengel, A. (1999). On the evaluation of document analysis components by recall, precision, and accuracy. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*, pages 713–716. IEEE.

Karasuyama, M. e Mamitsuka, H. (2013). Manifold-based similarity adaptation for label propagation. In *Advances in neural information processing systems*, pages 1547–1555.

Katariya, K. e Aluvalu, R. (2014). Agglomerative clustering in web usage mining: a survey. *International Journal of Computer Applications*, 89(8).

Khan, R., Hanbury, A., e Stoettinger, J. (2010). Skin detection: A random forest approach. In *2010 IEEE International Conference on Image Processing*, pages 4613–4616. IEEE.

Kim, J., Seo, J., e Laine, T. H. (2018). Detecting boredom from eye gaze and eeg. *Biomedical Signal Processing and Control*, 46:302–313.

Kulkarni, V. Y. e Sinha, P. K. (2012). Pruning of random forest classifiers: A survey and future directions. In *2012 International Conference on Data Science & Engineering (ICDSE)*, pages 64–68. IEEE.

LiKamWa, R., Liu, Y., Lane, N. D., e Zhong, L. (2013). Moodscope: Building a mood sensor from smartphone usage patterns. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 389–402. ACM.

Liu, D., Li, T., e Liang, D. (2014). Incorporating logistic regression to decision-theoretic rough sets for classifications. *International Journal of Approximate Reasoning*, 55(1):197–210.

Mikulas, W. L. e Vodanovich, S. J. (1993). The essence of boredom. *The Psychological Record*, 43(1):3.

Miller, G. (2012). The smartphone psychology manifesto. *Perspectives on psychological science*, 7(3):221–237.

Moro, S., Laureano, R., e Cortez, P. (2011). Using data mining for bank direct marketing: An application of the crisp-dm methodology. *Proceedings of the European Simulation and Modelling Conference*.

Müllner, D. (2011). Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*.

Myles, A. J., Feudale, R. N., Liu, Y., Woody, N. A., e Brown, S. D. (2004). An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 18(6):275–285.

Navaz, K., Athinarayanan, S., Sameena, S., e Kavitha, R. (2018). Distributed load balancing algorithm for wireless sensor network. *ICTACT Journal on Communication Technology*, 9(4).

Oulasvirta, A., Rattenbury, T., Ma, L., e Raita, E. (2012). Habits make smartphone use more pervasive. *Personal and Ubiquitous Computing*, 16(1):105–114.

Parker, J. (2001). Rank and response combination from confusion matrix data. *Information fusion*, 2(2):113–120.

Picard, R. W., Vyzas, E., e Healey, J. (2001). Toward machine emotional intelligence: Analysis of affective physiological state. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, pages 1175–1191.

Pielot, M., Dingler, T., Pedro, J. S., e Oliver, N. (2015). When attention is not scarce-detecting boredom from mobile phone usage. In *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*, pages 825–836. ACM.

Pimenta, A., Carneiro, D., Neves, J., e Novais, P. (2016). A neural network to classify fatigue from human-computer interaction. *Neurocomputing.*, 172:413–426.

Plarre, K., Raij, A., Hossain, S. M., Ali, A. A., Nakajima, M., Al'Absi, M., Ertin, E., Kamarck, T., Kumar, S., Scott, M., et al. (2011). Continuous inference of psychological stress from sensory measurements collected in the natural environment. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 97–108. IEEE.

Price, C., Maor, R., e Shachaf, H. (2018). Using smartphones for monitoring atmospheric tides. *Journal of Atmospheric and Solar-Terrestrial Physics*, 174:1–4.

Rish, I. et al. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46.

Rochac, J. F. R., Liang, L., Zhang, N., e Oladunni, T. (2019). A gaussian data augmentation technique on highly dimensional, limited labeled data for multiclass classification using deep learning. In *2019 Tenth International Conference on Intelligent Control and Information Processing (ICICIP)*, pages 145–151. IEEE.

Sano, A. e Picard, R. W. (2013). Stress recognition using wearable sensors and mobile phones. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pages 671–676. IEEE.

Seo, J., Laine, T. H., e Sohn, K.-A. (2019a). An exploration of machine learning methods for robust boredom classification using eeg and gsr data. *Sensors*, 19(20):4561.

Seo, J., Laine, T. H., e Sohn, K.-A. (2019b). Machine learning approaches for boredom classification using eeg. *Journal of Ambient Intelligence and Humanized Computing*, 10(10):3831–3846.

Simplilearn (2019). Neural network in 5 minutes | what is a neural network? | how neural networks work | simplilearn.

Sitanggang, I., Yaakob, R., Mustapha, N., e Ainuddin, A. (2013). Predictive models for hotspots occurrence using decision tree algorithms and logistic regression. *Journal of applied sciences*, 13(2):252–261.

SmartVision (2015). About crisp-dm.

Torous, J., Larsen, M. E., Depp, C., Cosco, T. D., Barnett, I., Nock, M. K., e Firth, J. (2018). Smartphones, sensors, and machine learning to advance real-time prediction and interventions for suicide prevention: a review of current progress and next steps. *Current psychiatry reports*, 20(7):51.

Van der Aalst, W. M., Rubin, V., Verbeek, H., van Dongen, B. F., Kindler, E., e Günther, C. W. (2010). Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87.

Wang, Y.-m., Chang, J.-x., e Huang, Q. (2010). Simulation with rbf neural network model for reservoir operation rules. *Water resources management*, 24(11):2597–2610.

Xu, Q., Xiong, Y., Dai, H., Kumari, K. M., Xu, Q., Ou, H.-Y., e Wei, D.-Q. (2017). Pdc-sgb: Prediction of effective drug combinations using a stochastic gradient boosting algorithm. *Journal of theoretical biology*, 417:1–7.

Yegnanarayana, B. (2009). *Artificial neural networks*. PHI Learning Pvt. Ltd.

Yeniay, Ö. e GÖKTAŞ, A. (2002). A comparison of partial least squares regression with other prediction methods. *Hacettepe Journal of Mathematics and Statistics*, 31:99–111.

Zhang, Z., Zhao, Y., Canes, A., Steinberg, D., Lyashevska, O., et al. (2019). Predictive analytics with gradient boosting in clinical medicine. *Annals of translational medicine*, 7(7).

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., e Schölkopf, B. (2004). Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328.

Zhu, X. e Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130.

A. APPENDIX

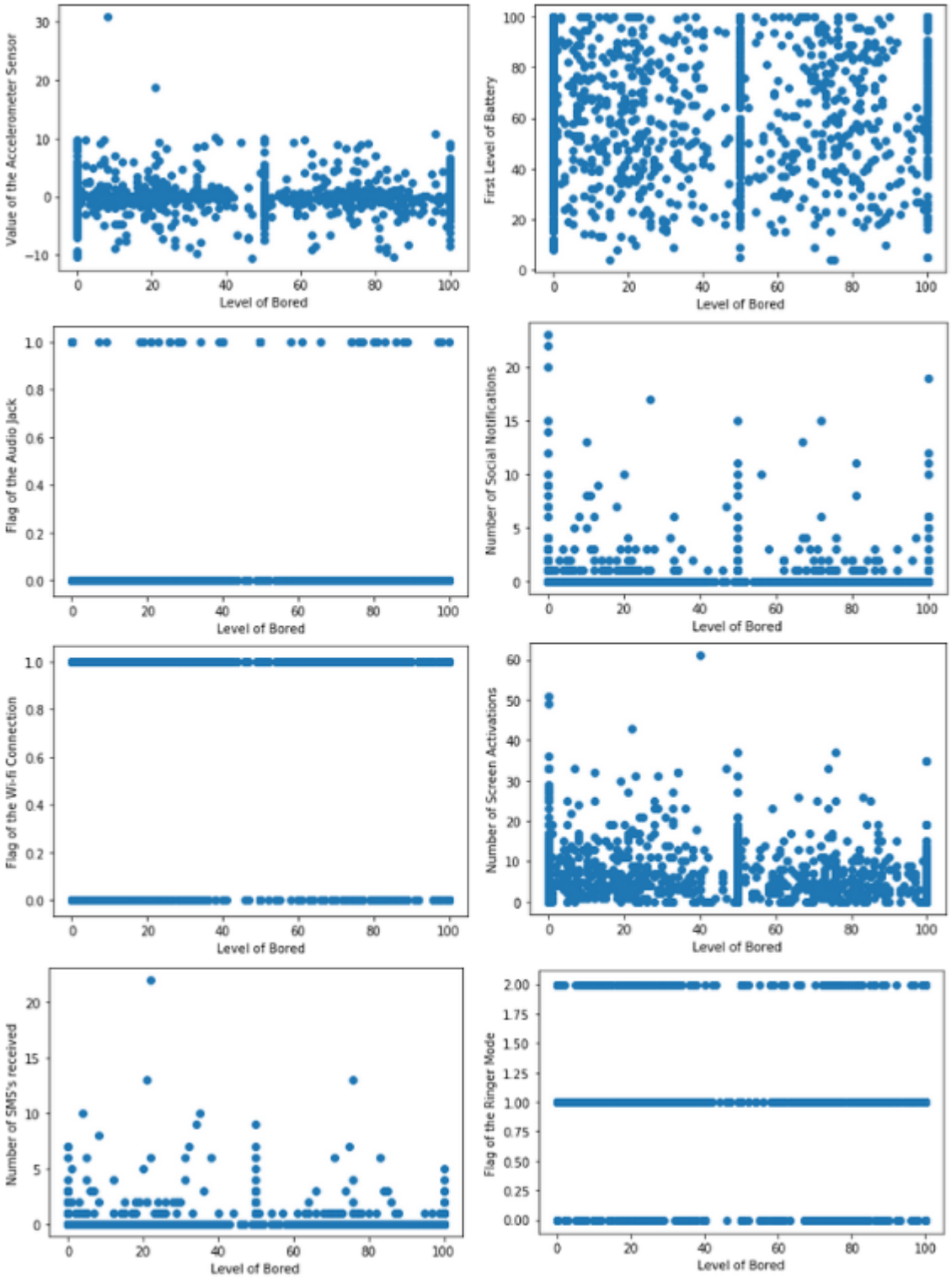


Figure A.1: Distribution Charts of the variables of the *dataset* concerning the target label.

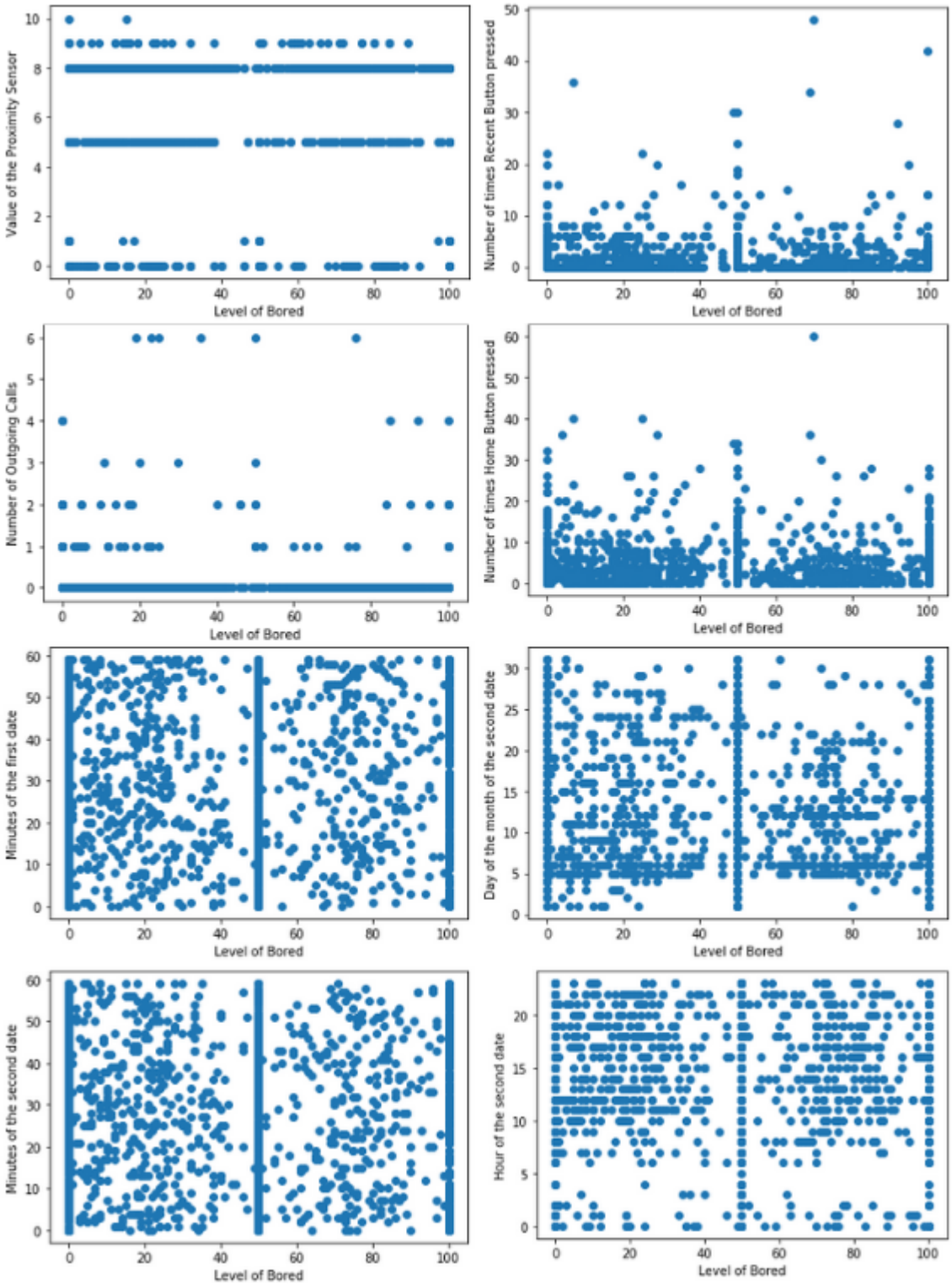


Figure A.2: Distribution Charts of the variables of the *dataset* concerning the target label.

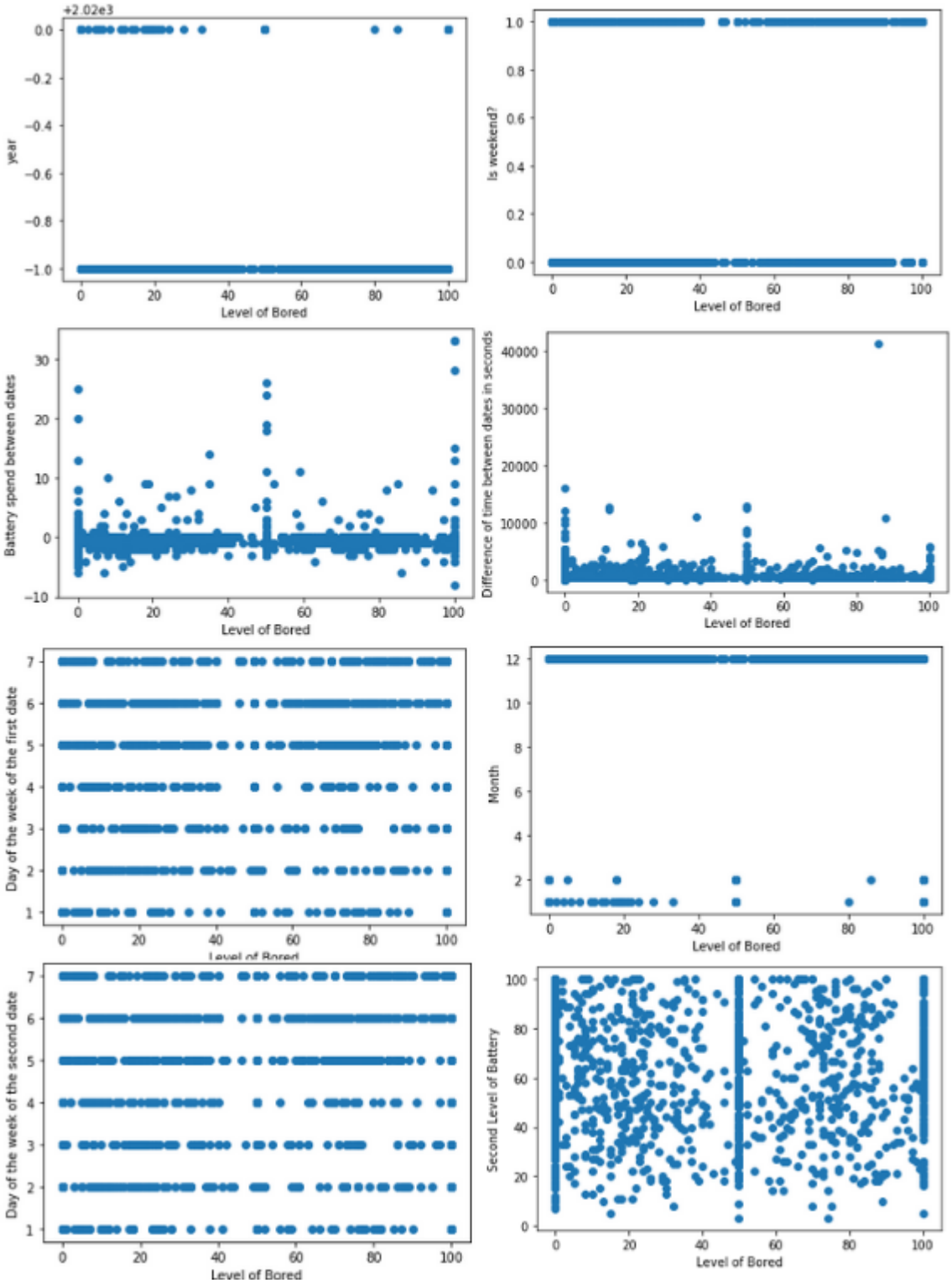


Figure A.3: Distribution Charts of the variables of the *dataset* concerning the target label.

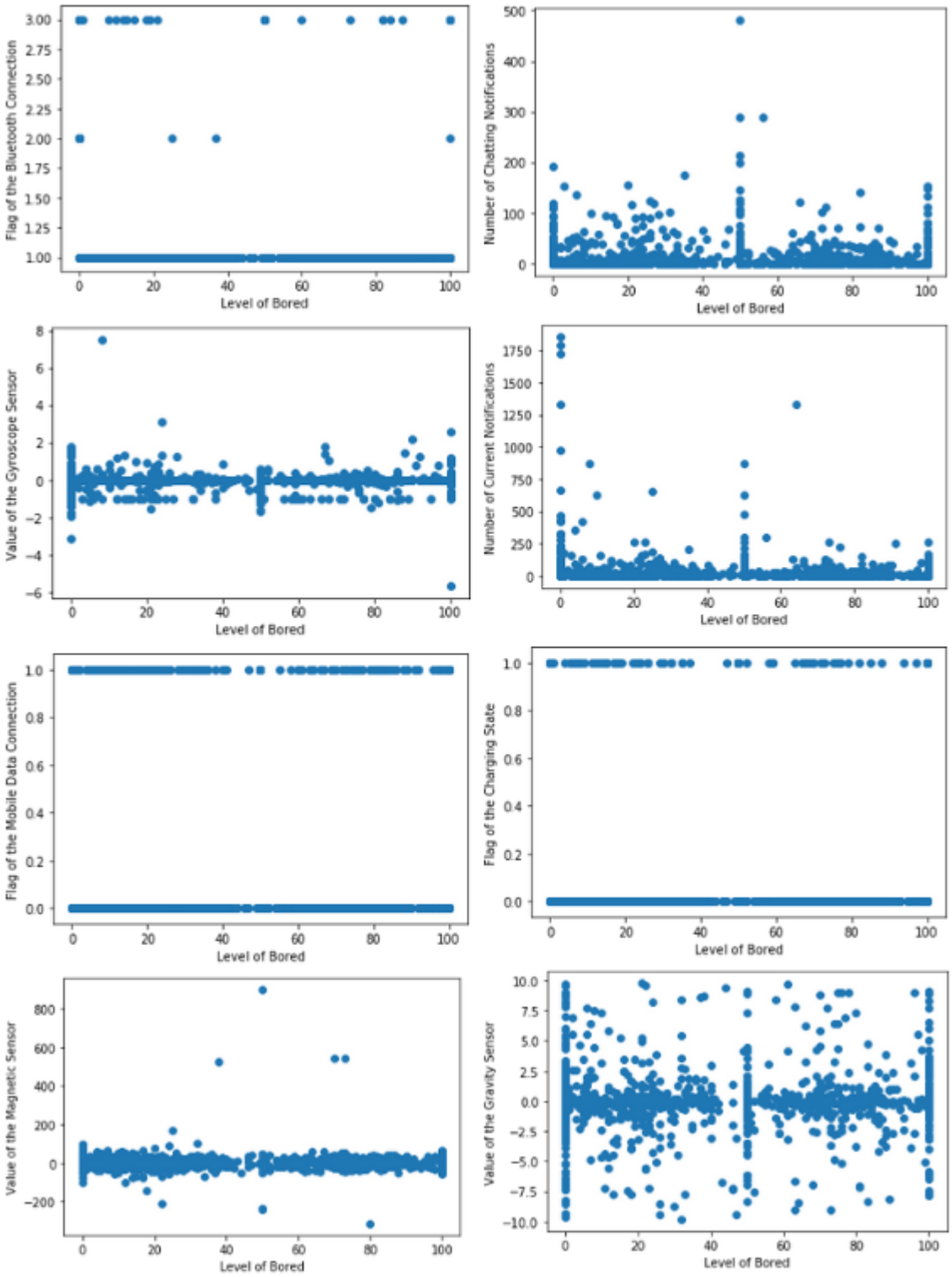


Figure A.4: Distribution Charts of the variables of the *dataset* concerning the target label.

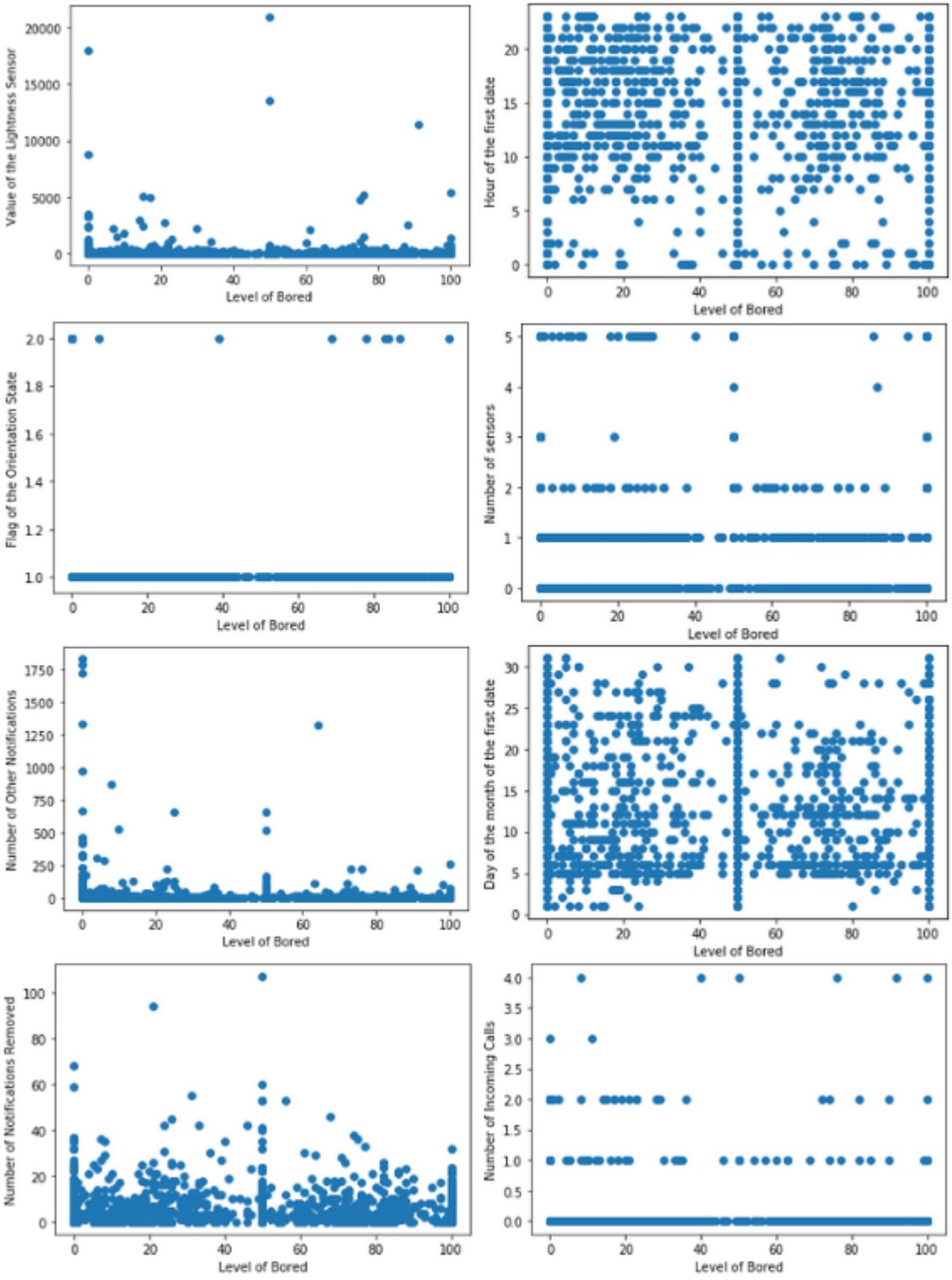


Figure A.5: Distribution Charts of the variables of the *dataset* concerning the target label.

B. APPENDIX

Model	RMSE	Parameter	Value
Decision Tree	1.944	Criterion	MAE
		Splitter	Best
		Maximum Depth	4
k-Nearest Neighbor	2.729	Number of Neighbors	30
		Leaf Size	27
		Weights	Uniform
		Algorithm	Ball Tree
		P	1
Support Vector Machine	2.697	Kernel	Polynomial
		Degree	6
		C	0.008
		Gamma	0.171

Table B.1: Results from the prediction of the null values present in the Gravity sensor *feature*.

Model	RMSE	Parameter	Value
Decision Tree	0.409	Criterion	MAE
		Splitter	Best
		Maximum Depth	1
k-Nearest Neighbor	0.420	Number of Neighbors	30
		Leaf Size	8
		Weights	Distance
		Algorithm	Ball Tree
		P	2
Support Vector Machine	0.410	Kernel	RBF
		Degree	6 (*)
		C	0.021
		Gamma	0.012

Table B.2: Results from the prediction of the null values present in the Gyroscope *feature*. The values marked with (*) are ignored by the Kernel.

Model	RMSE	Parameter	Value
Decision Tree	888.115	Criterion	MAE
		Splitter	Random
		Maximum Depth	2
k-Nearest Neighbor	897.433	Number of Neighbors	27
		Leaf Size	50
		Weights	Uniform
		Algorithm	Ball Tree
		P	1
Support Vector Machine	883.324	Kernel	Polynomial
		Degree	4
		C	1.072
		Gamma	0.475

Table B.3: Results from the prediction of the null values present in the Luminosity sensor *feature*.

Model	RMSE	Parameter	Value
Decision Tree	40.208	Criterion	MAE
		Splitter	Best
		Maximum Depth	2
k-Nearest Neighbor	42.385	Number of Neighbors	30
		Leaf Size	13
		Weights	Distance
		Algorithm	Brute
		P	1
Support Vector Machine	40.606	Kernel	Polynomial
		Degree	9
		C	3.054
		Gamma	0.002

Table B.4: Results from the prediction of the null values present in the Magnetic sensor *feature*.

Model	RMSE	Parameter	Value
Decision Tree	12.053	Criterion	MSE
		Splitter	Best
		Maximum Depth	2
k-Nearest Neighbor	15.912	Number of Neighbors	29
		Leaf Size	29
		Weights	Distance
		Algorithm	KD Tree
		P	1
Support Vector Machine	14.454	Kernel	Polynomial
		Degree	6
		C	0.187
		Gamma	0.002

Table B.5: Results from the prediction of the null values present in the Pressure sensor *feature*.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	187		
Maximum Depth	2		
ETA or Learning Rate	0.039	0.605	0.382
Gamma	0.040		
Minimum Child Weight	4		
Column Sample by Tree	0.150		

Table B.6: Results of the first iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	110		
Maximum Depth	4		
ETA or Learning Rate	0.083	0.608	0.477
Gamma	0.020		
Minimum Child Weight	6		
Column Sample by Tree	0.250		

Table B.7: Results of the second iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	1434		
Maximum Depth	23		
ETA or Learning Rate	0.100	0.565	0.517
Gamma	0.080		
Minimum Child Weight	6		
Column Sample by Tree	0.250		

Table B.8: Results of the third iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	955		
Maximum Depth	22		
ETA or Learning Rate	0.013	0.581	0.627
Gamma	0.080		
Minimum Child Weight	8		
Column Sample by Tree	0.300		

Table B.9: Results of the fourth iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	1757		
Maximum Depth	7		
ETA or Learning Rate	0.083	0.601	0.602
Gamma	0.040		
Minimum Child Weight	4		
Column Sample by Tree	0.250		

Table B.10: Results of the fifth iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	766		
Maximum Depth	18		
ETA or Learning Rate	0.083	0.585	0.607
Gamma	0.080		
Minimum Child Weight	8		
Column Sample by Tree	0.300		

Table B.11: Results of the sixth iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	769		
Maximum Depth	30		
ETA or Learning Rate	0.100	0.590	0.583
Gamma	0.020		
Minimum Child Weight	8		
Column Sample by Tree	0.300		

Table B.12: Results of the seventh iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	1064		
Maximum Depth	41		
ETA or Learning Rate	0.057	0.587	0.696
Gamma	0.020		
Minimum Child Weight	8		
Column Sample by Tree	0.250		

Table B.13: Results of the eighth iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	1662		
Maximum Depth	1		
ETA or Learning Rate	0.006	0.609	0.641
Gamma	0.020		
Minimum Child Weight	8		
Column Sample by Tree	0.200		

Table B.14: Results of the ninth iteration of *GB* optimization.

Parameter	Value	Internal Score	External Score
Number of boosting rounds	1049		
Maximum Depth	13		
ETA or Learning Rate	0.083	0.595	0.742
Gamma	0.020		
Minimum Child Weight	8		
Column Sample by Tree	0.200		

Table B.15: Results of the tenth iteration of *GB* optimization