NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

LOURENÇO MALHEIRO SERPA DE VASCONCELOS

Bachelor in Computer Science Engineering

IF THIS DO THAT: INTERPRETABLE MACHINE LEARNING MODELS FOR HIGH STAKES DECISION-MAKING

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
December, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

IF THIS DO THAT: INTERPRETABLE MACHINE LEARNING MODELS FOR HIGH STAKES DECISION-MAKING

LOURENÇO MALHEIRO SERPA DE VASCONCELOS

Bachelor in Computer Science Engineering

Adviser: Cláudia Alexandra Magalhães Soares
Assistant Professor, NOVA University Lisbon

Co-advisers: João Alexandre Carvalho Pinheiro Leite
Associate Professor, NOVA University Lisbon
Ricardo João Rodrigues Gonçalves
Assistant Professor, NOVA University Lisbon
Matthias Knorr
Assistant Professor, NOVA University Lisbon

Examination Committee

Chairs: Carmen Pires Morgado
Assistant Professor, NOVA University Lisbon
Ludwig Krippahl
Researcher, NOVA LINCS
Cláudia Alexandra Magalhães Soares
Assistant Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
December, 2022

If this do that: Interpretable machine learning models for high stakes decision-making

Acknowledgements

I would like to thank my adviser, professor Cláudia Soares, and my co-advisers, professors João Leite, Ricardo Gonçalves, and Matthias Knorr, for giving me the chance to work on a subject that I'm very passionate about, Machine Learning, and for all the guidance and suggestions that made this work possible. I would like to extend my thanks to all the professors of the Computer Science Department I had over these past five years for everything they taught me.

I also wish to thank my colleagues João Ferreira and Gonçalo Mateus for all the ideas we exchanged throughout this work. To all friends I made in the university who helped me achieve academic success, and Anna Jarczak for being the best friend during my breaks throughout this work, a friend that I made during my Erasmus in Poland that will remain a friend for life.

Lastly, I have to thank my brother, Tomás Vasconcelos, my mother, Ana Paula Malheiro, and her boyfriend, António Tavares, for all the support throughout my whole academic journey which culminated in this work.

Abstract

In recent years, there has been an increasing trend in the use of black-box machine learning models in high-stakes decision making throughout different domains of society. Even though these models might be very accurate, there is a need to understand the reason behind the decisions, that is, a need to interpret the models. This need can be seen in predicting the waiting time in the Emergency Department of a hospital, where we can not simply trust a model prediction to manage a real health environment without understanding its reasons, thus, making black-box models impracticable in this scenario. Although multiple studies have aimed to explain black-box models, this can give the model even more power and be misleading which is why we should avoid it in high-stakes decisions and use an interpretable model instead.

This thesis demonstrates that interpretable machine learning models can be as good or even better than state of the art black-box models when predicting the waiting time in the Emergency Department of a hospital. Moreover, we also propose four new rule-based methods. To achieve this, we implemented 12 machine learning models, 6 non-interpretable (black-box) methods (ARIMA, SARIMA, Prophet, LSTM, GRU and the Transformer) and 6 rule-based and interpretable methods (RuleFit, SIRUS, REN, RDLL, RDLE and RDLR), 4 of which are proposed in this work (the REN, RDLL, RDLE and RDLR). The results obtained revealed that although some black-box models can achieve very good predictions of the waiting times in the emergency department, in some scenarios, interpretable models can outperform them.

Keywords: Interpretability, Machine Learning, Time Series, Interpretable Machine Learning, Rule-based Algorithms, Emergency Department Waiting Times

Resumo

Nos últimos anos, tem havido uma tendência crescente na utilização de modelos *black-box* de *machine learning* na tomada de decisões de alto risco em diferentes domínios da sociedade. Embora estes modelos possam ser muito precisos, existe a necessidade de compreender a razão por detrás das decisões, isto é, existe uma necessidade de interpretar os modelos. Esta necessidade pode ser vista na previsão do tempo de espera no Departamento de Emergência de um hospital, onde não podemos simplesmente confiar numa previsão de um modelo para gerir um ambiente de saúde real sem compreender a sua razão, tornando assim os modelos *black-box* impraticáveis neste cenário. Apesar de terem sido desenvolvidos vários estudos com o objetivo de explicar os modelos *black-box*, isto pode dar-lhes ainda mais poder e ser enganador. Por estas razões devemos optar por usar um modelo interpretável em vez de modelos *black-box*.

Esta tese demonstra que modelos de aprendizagem automática interpretáveis podem ser tão bons ou até mesmo melhores do que os modelos não interpretáveis considerados *state-of-the-art* ao prever o tempo de espera no departamento de emergência de um hospital. Além disso, esta tese propõe quatro modelos de aprendizagem automática novos baseados em regras. De forma a atingir estes objetivos, implementámos 12 modelos de aprendizegem automática, 6 dos quais não interpretáveis (*black-box*) (ARIMA, SARIMA, Prophet, LSTM, GRU e o Transformer) e 6 modelos interpretáveis baseados em regras (RuleFit, SIRUS, REN, RDLL, RDLE e RDLR), 4 dos quais foram desenvolvidos e propostos nesta dissertação (REN, RDLL, RDLE e RDLR). Os resultados obtidos revelaram que, embora alguns modelos *black-box* consigam fazer previsões muito boas dos tempos de espera no serviço de urgência, em alguns cenários, os modelos interpretáveis são capazes de os superar.

Palavras-chave: Interpretabilidade, Aprendizagem Automática, Time Series, Aprendizagem Automática Interpretável, Algoritmos baseados em regras, Tempo de Espera no Serviço de Emergência

Contents

List of Figures

List of Tables

Acronyms

AIC Akaike Information Criterion 22, 42, 43
ANN Artificial Neural Network 23
ARIMA Autoregressive Integrated Moving Average 21, 22, 23, 41, 42, 43, 44, 45, 63, 64, 65, 66, 76, 83, 84

BIC Bayesian Information Criterion 22, 42, 43
BPTT Back-Propagation Through Time 45

CART Classification and Regression Tree 20
CEC Constant Error Carousel 46
CELOE Class Expression Learning for Ontology Engineering 58

ED Emergency Departments 21, 22, 23, 24, 25

GAM Generalized Additive Model 9, 10, 11
GBM Gradient Boosting Machines 23
GLM Generalized Linear Models 9
GRU Gated Recurrent Unit xiv, 41, 49, 50, 54, 63, 65, 66, 72, 76, 83, 84

ISLE Importance Sampled Learning Ensemble 15

LSTM Long Short-Term Memory 41, 45, 46, 48, 49, 50, 51, 54, 63, 64, 65, 66, 72, 76, 83, 84

MAE Mean Absolute Error xv, xix, xx, xxi, 25, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 78, 79, 80, 84
MAPE Mean Absolute Percentage Error 22, 23, 24
MART Multiple Additive Regression Trees 15

MSE Mean Squared Error 23, 25, 65
MSPE Mean Squared Prediction Error 24

OLS Ordinary Least Square 25

RDLE Interpretable Rules by DL-Learner and ElasticNet Regression xv, xx, 61, 70, 71, 72, 75, 76, 78, 79, 80, 81, 83, 85
RDLL Interpretable Rules by DL-Learner and Lasso Regression xv, xx, 61, 70, 71, 72, 75, 76, 78, 79, 80, 81, 83, 84, 85
RDLR Interpretable Rules by DL-Learner and Ridge Regression xx, 61, 70, 71, 72, 75, 76, 77, 78, 83, 84, 85
REN Rulefit with ElasticNet xv, xx, 69, 70, 72, 75, 76, 77, 78, 79, 80, 81, 83, 84, 85
RNN Recurrent Neural Network 41, 45, 46
RTRL Real-Time Recurrent Learning 45

SARIMA Seasonal Autoregressive Integrated Moving Average 22, 23, 41, 43, 64, 65, 66, 76, 83, 84
SIRUS Stable and Interpretable Rule Set xv, xix, xx, 11, 13, 17, 18, 19, 20, 21, 55, 56, 57, 66, 68, 69, 70, 72, 75, 76, 78, 79, 80, 81, 83, 84, 85
SVM Support Vector Machines 23

1

Introduction

1.1 Motivation

With the widespread use and continuous growth of machine learning, it is becoming a necessity to understand how machine learning models work and make decisions. This continuous growth comes from the increase both in computational power and in the amount of available data which allows machine learning models to extract new knowledge from [1].

Nowadays we have a machine learning model almost everywhere in our lives, let it be in online shopping with recommendation systems or at our houses with a robot vacuuming our floors. We must not let these models become biased as they can easily become discriminative against under-represented groups in the dataset. Since these models are also being used in services like health and judicial courts this can deeply affect the life of a person. To find out if a model is becoming biased or not we need to be able to understand it and understand why it took a certain decision. This will not only allow us to reduce the bias of the model but also improve it as we can understand why it took a certain wrong action and correct it.

Interpretability has never been so important and could help avoid catastrophic consequences since some of these machine learning models are being used in high-stakes decisions. Moreover, we can not blindly trust a machine learning model for high-stakes decisions that affect society. This could cause problems in many important domains like healthcare and criminal justice. There is a wrong belief that creating methods to explain these black-box machine learning models will solve the problem, when in fact, it can make the problem even worse and cause great harm to society [2] as we will see.

Instead of trying to explain "black-box" machine learning models in high-stakes problems, we should be implementing interpretable models instead.

This thesis will test if we can have an interpretable model with comparable accuracy to a black-box model in a particular high-stakes problem. Specifically, this thesis will try to predict the waiting time in the emergency department in the hospital *Hospital Santa Maria* with an interpretable model.

1.2 Why should we use interpretable machine learning

Every day we have more and more machine learning algorithms in our life, some of them making decisions for us with some of those decisions being high stake decisions like deciding how long a person should stay in prison. This is why interpretability in machine learning is becoming extremely important. We can not blindly trust a machine learning algorithm without understanding it, we may be taking a wrong decision only because the algorithm told us without understanding why.

The existing black-box machine learning models are too complicated to understand (which is why they are called black-box models) and are hard to troubleshoot. Moreover, these models sometimes have undesired behaviors and predict the right answer for the wrong reasons, which may lead to poor performance outside the training data. An example of this is shown in the paper [3] where they found that the model was labeling horses just because the images had the same source tag in the left lower corner and not because of the horse in the image as can be seen in figure 1.1. Another example is in criminal justice where individuals may have got more years of extra prison time due to an error in the black-box model inputs which could have been avoided if an interpretable model was used instead since the reason for the prison time would have been understandable [4].

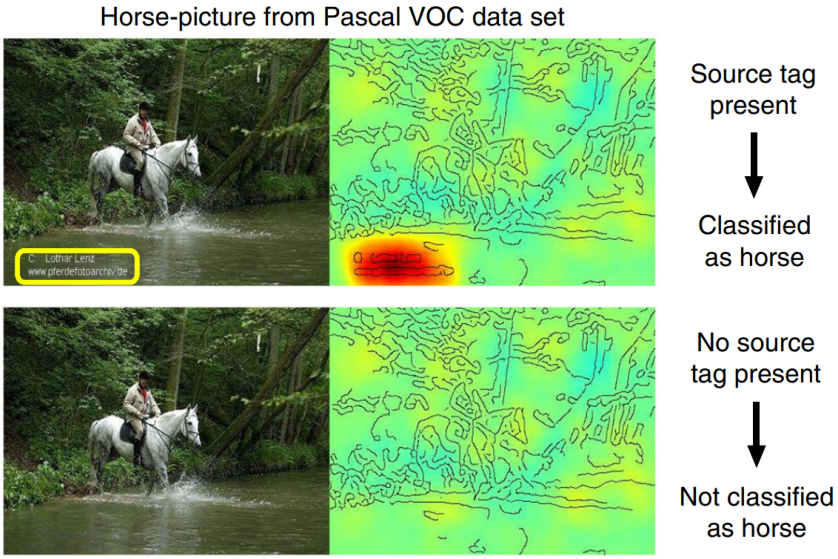


Figure 1.1: A model trained on the PASCAL VOC 2007 dataset identifies a horse on the image by focusing on the source tag presented in the lower left corner. When the tag is removed the model can not classify the figure as a horse anymore [3].

Explaining black-box models raises problems as the process only ensures a high correlation between the prediction of the explanation and the prediction of the black-box, and thus it may fail to represent the causal relation between input feature and prediction; there could be more than a single explanation to the same black-box that looks different and they may not be robust and vary significantly with a small perturbation to input data.

Considering these problems, the explanation of the black-box models may be misleading [5].

Interpretable machine learning models are clearly needed. Explaining black-box models instead of having an interpretable one can make the problem of not understanding the model even worse since it can mislead us.

There is literature where the interpretability of a model is confused with its explainability, which is something we will clearly differentiate here. There is also literature that explains black-box models without considering if there is an interpretable model with the same accuracy.

1.2.1 Interpretability

Interpretability is a domain-specific notion [6], so there is no all-purpose definition.

Mathematically, it is very difficult to define interpretability. Non-mathematically we can think of interpretability as the degree to which a human can understand the cause of a decision [7]. With this definition, we can compare the interpretability of two models by comparing how easy it is to comprehend why the decisions or predictions were made by a machine learning model, the easier it is, the higher its interpretability is. One way of doing this comparison is by comparing how easy it is for a human to comprehend the decisions from the different models [8].

Interpretable machine learning can be seen as the extraction of relevant knowledge from a machine learning model, where this knowledge can come either from the data or learned by the model [9].

1.2.1.1 Importance of Interpretability

Sometimes when we have a predictive model we do not want to know why the prediction was made, instead we only want to know what the prediction is. But in other problems, we may want to know why the prediction was made so we can learn more about the problem, the data, and the reasons why a model might fail. If we have a model being used in a low-risk environment, where it does not have serious consequences (like an advertiser system) we may not need to know why the predictions are made. The need for interpretability comes from problems or tasks where it is not enough to get the prediction, where we also need an explanation about how the model made the prediction that because a correct prediction only solves part of the problem.

One of the reasons that drive the demand for interpretability according to [10] is human curiosity and learning. When we use black-box machine learning models in research the scientific findings remain hidden, since only the model knows how it found them and does not explain its predictions to humans. To make it easier to learn and satisfy human curiosity it is crucial that the predictions or behaviors of machine learning models are interpretable and explainable.

Another reason for this demand for interpretability is the human desire to find meaning in the world and harmonize contradictions or inconsistencies between elements of our knowledge structures. In science, we want to gain knowledge, but sometimes we use black-box models to solve the problems, and only the model gains knowledge. To extract this knowledge from the model we need it to be interpretable.

When machine learning models are used in real-world tasks that require safety measures and testing, like a self-driving car, we need to be 100% sure that the system is error-free otherwise it could have serious consequences. By being able to interpret and understand the model we can think about cases that we would not think about before since we would not know what are the most important learned features for our model.

Machine learning models usually get biased from the training data, and this can make a model racist by discriminating against underrepresented groups. Interpretability allows us to detect bias in machine learning models.

Integrating machine learning models and algorithms into our daily lives requires interpretability to increase social acceptance. People will accept more machines and algorithms that they can understand, and for that, the machines or algorithms need to be interpretable so they can explain their predictions.

By creating a shared meaning of something, an explainer influences the recipient actions, emotions, and beliefs. Explanations are used to manage social interactions. This is also valid for machines, they have to "persuade" us somehow that they can achieve their goal, otherwise we would not use them.

1.2.1.2 Scope of Interpretability

Each step of a machine learning algorithm that trains a model to produce predictions can be evaluated in terms of transparency or interpretability.

The transparency of an algorithm is about understanding how the algorithm works, how the model is learned from the data, and what kind of relationships it can learn, but not for a specific model. Algorithm transparency does not require knowledge about the data or learned model, only about the algorithm used.

If an entire model can be comprehended at once, it can be described as an interpretable model [11]. This kind of model allows us to understand how it makes decisions based on a total view of its features and learned components. To explain the global model output we need everything: knowledge of the algorithm, the trained model, and the data. With these, we can also say which features are more important and what kind of interactions between the features take place.

Global model interpretability on a modular level is when we can easily understand a single weight for example in a Naive Bayes model but can not manage to memorize all the weights. While global model interpretability is usually very hard to get, there is a good chance of understanding at least some models on a modular level. For linear models,

the interpretable parts are the weights but for trees, it would be the splits and leaf node predictions.

When we can analyze a single instance and understand why the model predicted something for a given input, we have local interpretability for a single prediction. The prediction might only depend linearly or monotonically on a few features instead of having a complex dependence on them.

Local interpretability for a group of predictions is when we can explain the model predictions for multiple instances. This can be achieved either with global model interpretation methods (on a modular level) or with explanations of individual instances. With the global model interpretation method, we can take a group of instances and treat them as if the group were the whole dataset and use the global methods with this subset. With the individual explanation method, we use it on each instance and then list or aggregate the entire group.

1.2.2 Interpretable Machine Learning Fundamentals

There are 5 fundamental principles for interpretable machine learning according to [12].

The first principle is "An interpretable machine learning model obeys a domain-specific set of constraints to allow it to be more easily understood by humans. These constraints can differ dramatically depending on the domain."

An interpretable supervised learning setup, where $\{z_i\}_i$ represents the data, and the models are chosen from function class F is:

$$\min_{f \in F} \frac{1}{n} \sum_i \text{Loss(f,}z_i) + C \cdot \text{Interpretability Penalty}(f),$$

subject to Interpretability Constraint(f).

The loss function and the interpretability constraints are chosen to match the domain. These constraints make the resulting model f or its predictions more interpretable and we can use the constant C to trade-off between accuracy and interpretability of the model. We can tune this constant C and sacrifice either accuracy for more interpretability or interpretability for more accuracy either by cross-validation or by the desired tradeoff.

We can use the same equation for unsupervised learning, but in this case, the loss term is replaced with a loss term for the unsupervised problem.

Interpretability differs across domains and there are so many different interpretability metrics that we might choose from a combination of them that are specific to the domain we are working on. We may not be able to define the best definition for interpretability but if our chosen interpretability measure is helpful, we should use it.

The second principle is: "Despite common rhetoric, interpretable models do not necessarily create or enable trust - they could also enable distrust. They simply allow users to decide where to trust them. In other words, they permit a decision of trust, rather than trust itself."

When using a black-box model, we make decisions about trusting the model or not with much less information, we do not have the knowledge about the reasoning process of the model and we also do not know if it will generalize outside of the training data. In [13] the authors study the ethical implementation of AI to select embryos in In Vitro Fertilization and they state that clinicians should explain the basis of how embryos are selected for transfer, whether it is clinical or AI-assisted. This is not possible if the clinicians do not know how the AI model chose that specific embryo, just like the patient is trusting the clinician, the clinician is trusting the AI model not knowing how he chose it. While interpretable AI is an enhancement of human decision-making, black-box AI is a replacement for it.

The third principle states that it is important to not assume that to improve interpretability one needs to sacrifice accuracy. Interpretability often improves accuracy and not the opposite. "Interpretability versus accuracy is, in general, a false dichotomy in machine learning" [12].

Interpretability and complexity have been traditionally associated, specifically interpretability and sparsity. Usually, sparsity is one component of interpretability, and there is almost always a tradeoff between accuracy with sparsity but there is no evidence of a tradeoff between accuracy with interpretability. In fact, as stated in [14] Interpretability might even improve accuracy, as it permits an understanding of when the model might be incorrect.

In [15] the authors show that it is possible to build sparse rule lists (decision trees) that have an accuracy comparable to COMPAS, which is a black-box model used to predict criminal recidivism because no one outside of its designers knows its secret formula and yet it is used widely across the United States. This is an example where a black-box model is used unnecessarily since an interpretable model could be used instead with comparable accuracy.

For problems where we have "raw" data, such as images, sound files, or text, where each pixel, bit, or word is not useful on its own, neural networks currently have an advantage over other approaches as shown in [16]. In [17] the authors show that even neural networks can benefit from being interpretable. In their experiments, they compare the interpretability, average relative location deviations for multi-category classification, and classification accuracies based on different datasets between a normal neural network and an interpretable neural network. The results they got show that interpretable neural networks not only are more interpretable but also have much better location stability and still have better accuracy than ordinary convolutional neural networks.

These are some clear examples where having an interpretable model instead of a black-box model would not result in a loss of accuracy and could even improve it.

The fourth principle is: "As part of the full data science process, one should expect both the performance metric and interpretability metric to be iteratively refined."

It is useful to create many interpretable models that satisfy the known constraints and have domain experts choose between them. Since the definition of interpretability

may vary depending on our problem, the reason why the experts choose one model over another helps us to refine the definition of interpretability in our problem.

The fifth principle is that for high-stakes decisions, interpretable models should be used if possible, rather than "explained" black-box models.

As stated previously, explaining black-boxes can be misleading and generally they do not serve their intended purpose. These problems have arisen with the assessment of fairness and variable importance, and in [18] the authors introduce a method that modifies an existing model and downgrades feature importance of key sensitive features across six explanation methods and unseen test points across four datasets while keeping a similar accuracy. They demonstrate that many popular used explanation methods are not able to indicate if a model is fair or not reliably. It raises concerns about relying on explanation methods to measure or enforce standards of fairness. Black-box models are also more difficult to troubleshoot and if the explanation model is not correct, it can be difficult to tell if it is the black-box model that is wrong or if it is right and the explanation model is wrong. Another issue with explaining black-box models is that the explanations themselves could contain significant uncertainty that undermines users trust in the predictions and raises concern about the robustness of the models as stated in [19].

Generally, black-box models do not have better accuracy than a well-designed interpretable model[12], and explanations that seem reasonable can lead to a lack of interest in finding an interpretable model with the same accuracy as the black-box model.

Explaining a black-box model is often used as an excuse to use it instead of trying to find an interpretable model instead. Not only that, but it also gives even more authority to the black-box model that could be wrong [14].

1.2.3 Problems with explaining black-box machine learning models

A black-box model can be either a model that is too complicated for any human to understand or a model that is proprietary. Deep learning models, for example, usually are black boxes of the first type due to their high recursivity. An explanation of a black-box model is a separate model that is supposed to replicate most of the black-box model behavior [2].

Explanations of black-box models are not faithful to what the original model computes, otherwise, the explanation model would equal the original model and we would not need the black-box model in the first place, only this explanation model which would be an interpretable model. This leads to the danger mentioned in [20], any explanation model for a black-box can be inaccurate in parts of the feature space.

An inaccurate explanation model limits trust in the explanation, and by consequence, in the black-box model that it is trying to explain. Even if an explainable model has a 95% agreement with the black-box model, it will be wrong 5% of the time and we do not know when it is correct or incorrect and therefore cannot trust the explanations. Consequently,

Figure 1.2: Saliency does not explain anything, it only shows us where the network is looking at. We have no idea why this image is labeled as either a dog or a musical instrument when considering only saliency. The explanations look almost the same for both classes. Credit: Chaofen Chen, Duke University [2].

we cannot trust the original black-box. If we cannot trust for certain in our explanation model, we cannot trust the black-box model that it is trying to explain.

The explanations for complex black-box models hide the fact that these are difficult to use in practice. An example of this is shown in [4], mentioned before, where a typographical error leads to extra time in jail. It is easier to make an error in a model with 130 hand-typed inputs than in one involving just 5 hand-typed inputs.

Even if the explanation model is correct in its approximation of the black-box model and the original black-box is correct in its prediction, it is still possible that the explanation does not explain everything. In [2] the author gives us an example from image processing where saliency maps are used. These maps are often considered explanatory and can be useful to know what parts of the images are being used, but it does not explain how these parts of the images are being used. Knowing where the model is looking within the image does not explain to the user what it is doing with those pixels as illustrated in figure 1.2. The saliency maps for multiple classes could be essentially the same, and this happens in figure 1.2, where the saliency maps give us almost the same explanation as to why the image might contain a Siberian husky and why it might contain a transverse flute.

An unfortunate trend is to show explanations only for the observations that were correctly labeled when demonstrating the explanation method. This can be misleading and establish a false sense of confidence both in the explanation method and consequently in the black-box model.

Troubleshooting a black-box model may be flawed and we do not know it because it is difficult to troubleshoot. Having an explanation model of it may not help and it makes the problem even worse because now we must troubleshoot two models instead of just one.

1.2.4 Sparse and Logical Models

In problems where we have tabular data where the features are meaningful, sparsity is often used as a measure of interpretability of the model. Sparsity is an important measure since humans can only handle 7±2 cognitive entities at once [21]. Sparsity in machine learning makes it easier to troubleshoot the model, check for typographical errors, since there will be less to check, and reason about counterfactuals. Sparsity is seldom the only consideration for interpretability, but by designing a sparse model, it often can handle additional constraints. By optimizing for sparsity, we can establish a baseline for how sparse a model could be with a particular level of accuracy

Despite all these advantages, it is not always a good idea to have more sparsity. As stated in [22], humans are mentally opposed to too simplistic representations of complex relations. An example where having a very sparse model is not a good idea is in medicine where a single symptom rarely allows for good predictive power.

Logical models are models that consist of logical statements. They involve "if-then", "or" and "and" clauses and are one of the most popular algorithms for interpretable machine learning. This is because they provide a human-understandable reason for each of their predictions. These models are often a very good choice to model categorical data with potentially complicated interaction terms and for multiclass problems. They are also robust when data have outliers and can easily handle missing data. Logical models can also be highly nonlinear and even classes of sparse nonlinear models can be very powerful.

In Figure 1.3 we can visualize three different logical models, on the left of the figure we have a decision tree, on the top right a decision list, and on the lower right, we have a decision set. Decision trees are predictive models that have the structure of a tree where each leaf node makes a prediction and to get to the leafs each brand node tests a condition. Just like rule lists, decision lists are composed of if-then-else statements where the rules are tried in order and the first satisfied rule makes the prediction. Unlike decision trees, rules in a decision list may have multiple conditions in each split. Decision sets are made of an unordered collection of rules that are a conjunction of conditions and a positive prediction is made if at least one of the rules is satisfied.

1.2.5 Generalized Additive Models

Generalized Additive Model (GAM) present a flexible extension of Generalized Linear Models (GLM), allowing arbitrary functions to model the influence of each feature on a response. The set of GAMs includes the set of additive models, which includes the set of linear models which includes scoring systems. This hierarchy can be seen in figure 1.4.

The standard form of a GAM can be seen in the following equation

$$g(E[y]) = \beta_0 + f_1(x_{\cdot 1}) + ... + f_p(x_{\cdot p})$$

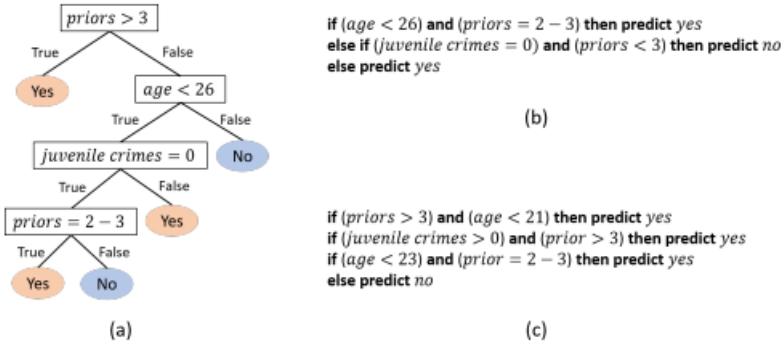


Figure 1.3: Predicting which individuals are arrested within two years of release by a decision tree (a), a decision list (b), and a decision set (c). The dataset used is the ProPublica recidivism dataset [23] [12].

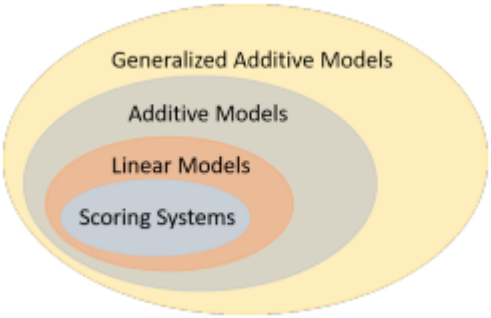


Figure 1.4: Hierarchical relationships between GAMs, additive models, linear models, and scoring systems [12].

where $g(\cdot)$ is a link function, $E[y]$ is the expectation of y, $x_{\cdot i}$ indicates the ith feature and f_j's are univariate component functions that are possibly nonlinear. The expression describes an additive model (such as a regression model) if $g(\cdot)$ is the identity. The expression describes a generalized additive model that can be used for classification if $g(\cdot)$ is the logistic function. This standard form of GAMs is interpretable because the model is constrained to be a linear combination of univariate component functions. We can even plot each component function individually and see the contribution of a single feature to the prediction. The GAM becomes a linear model if the features are all binary and the resulting visualization are just step functions. If the GAM has bivariate component functions, that is, if f_j depends on two variables, according to [24] we can visualize these two-dimensional interactions using a heatmap and understand the pairwise interactions.

Contrary to decision trees, which are capable to handle complex interactions of categorical variables, GAMs usually do not handle more than a few interaction terms.

The f_j component function can have many forms. If we consider that they take the form of the weighted sum of indicator functions, we can represent them as:

$$f_j(x_{\cdot j}) = \sum_{thresholds j'} c_{j,j'} \mathbb{1}[x_{\cdot j} > 0_{j'}] \tag{1.1}$$

GAM becomes a scoring system if the weights on the indicator functions are integers and only a small set of weights are nonzero. In (1.1), all the indicators are forced to aim in one direction ($\mathbb{1}[x_{\cdot j} > 0_{j'}]$), if to this we add the constraint that all coefficients have to be nonnegative, then the function will be monotonic, this is, is entirely nonincreasing or nondecreasing.

We can fit GAMs in many different ways, but the traditional way is to use back fitting. In [25] different procedures for fitting GAMs are compared and the results show that boosting performed slightly better than all the other procedures. In [26] they found that using size-limited bagged trees on gradient boosting generally achieved better performance.

GAMs simplicity comes from their sparsity in the number of component functions and smoothness of these functions. When we have prior knowledge (for example if risk increases with age) we can improve the interpretability of the GAM by imposing monotonicity of the component functions.

1.3 Objectives

With this dissertation we aim to introduce four novel rule-based machine learning models, three of which use the DL-Learner software, which uses a logic reasoner to create and choose the best and shortest rules. Moreover, we compare our new models with two other rule-based machine learning algorithms (RuleFit and SIRUS) and test if interpretable machine learning models can have comparable accuracies to black box machine learning models in a specific high-stakes scenario, which is the prediction of the waiting time in the Emergency Department in the hospital "Santa Maria".

In the next chapter, we will start by studying the current state-of-the-art rule-based models, which are the RuleFit and SIRUS, and how other studies have tried to predict the affluence and waiting time in Emergency Departments of different hospitals across the world.

2

State of the art

2.1 Interpretable Machine Learning Models

One way of having an interpretable machine learning algorithm is to build a rule-based tree learning algorithm. In this chapter we will study two rule-based tree methods (RuleFit and SIRUS) that are intrinsically interpretable, that is, the interpretability in these models is achieved by restricting their complexity.

2.1.1 RuleFit

Besides being interpretable, rule-based tree learning algorithms are shown to produce predictive accuracies comparable to the best methods in regression and classification problems [27]. These models are constructed as linear combinations of simple rules derived from the data where each rule has a simple form and therefore is easy to understand as is its influence on predictions.

RuleFit makes use of one of the most powerful learning methods which are the learning ensembles. By using rules in an ensemble learning RuleFit can be described as a rule-based ensemble that takes the following form

$$\mathrm{F(x)} = a_0 + \sum_{m=0}^{M} a_m f_m(\mathrm{x})$$

where f represents the different elements of the ensemble, a the coefficient of each ensemble, M the size of the ensemble used, and F(x) the prediction of the ensemble.

Each ensemble method has its base learners, $\{f_m(x)\}_1^M$, a method to derive them from the data, and a method to obtain the linear combination parameters. One approach to get the linear combination parameters with a given set of base learners is to use the regularized linear regression Lasso on the training data $\{x_i, y_i\}_1^N$. The Lasso regression solution can be described as follows

$$\{\hat{a}_m\}_0^M \in \underset{\{a_m\}_0^M}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, a_o + \sum_{m=1}^{M} a_m f_m(x_i)\right) + \lambda \cdot \sum_{m=1}^{M} \mid a_m \mid \tag{2.1}$$

where L is the loss function, which may be any loss function chosen according to the problem, and λ is the regularization parameter. The Lasso regression induces sparsity, meaning that many coefficients will be set to zero and the corresponding rules will not be included.

Each rule in a rule-based ensemble can be written as (r_m)

$$r_m(\mathrm{x}) = \prod_{j=1}^{n} I(x_j \in s_{jm})$$

where I is the indicator of the truth of each argument, that is, $I(x) = 0$ if the argument is false and $I(x) = 1$ if the argument is true. As a result, each base learner will have a value of 1 or 0. It takes the value 1 if the conditions are all true otherwise it takes the value 0.

To be more interpretable the ensemble should have simple rules, each defined by a small number of variables. For example, the rule $r_y(\mathrm{x})$ shown below is defined by three variables and a value of 1 means that all the conditions are met, meaning that the person has an age between 18 and 34, is single, or lives with another person but is not married and is renting a house. Since every number multiplied by 0 results in 0, the outcome of the multiplication will be 0 if atleast one of the requirements is not satisfied. On the other hand, a value of 1 in this example increases the odds of frequenting bars and nightclubs.

$$r_y(\mathrm{x}) = \begin{cases} I(18 \leq \text{age} \leq 34) \\ \cdot I(\text{marital status} \in \{\text{single, living together-not married}\}) \\ \cdot I(\text{householder status} = \text{rent}) \end{cases}$$

The rules in a rule ensemble method can be generated by a tree algorithm. This allows us to take advantage of the existing fast algorithms to produce decision trees and use these decision trees as base learners for our rule ensemble where each node of each tree produces a rule.

The size of the trees is proportional to the complexity of the rules that they allow to produce, larger trees can produce more complex rules since each rule may have more variables (factors) defining it. The more complex a rule is the more interactions among the variables it will be able to capture. High order interaction effects require larger trees, but the larger the tree the harder the rules are to understand, making them less interpretable. Besides the loss of interpretability, having ensembles that have a large fraction of high order interaction rules will make them worse to capture low order interaction effects, since smaller trees are better for these latter targets. A strategy found that solves this problem is producing an ensemble of trees of varying sizes, this way we can capture both high and low order interaction effects.

2.1.1.1 Comparison between rule-based and tree-based ensembles

An important measure in every learning method is accuracy, which is the percentage of correct predictions for the test data. In the paper, [27] the authors compared the

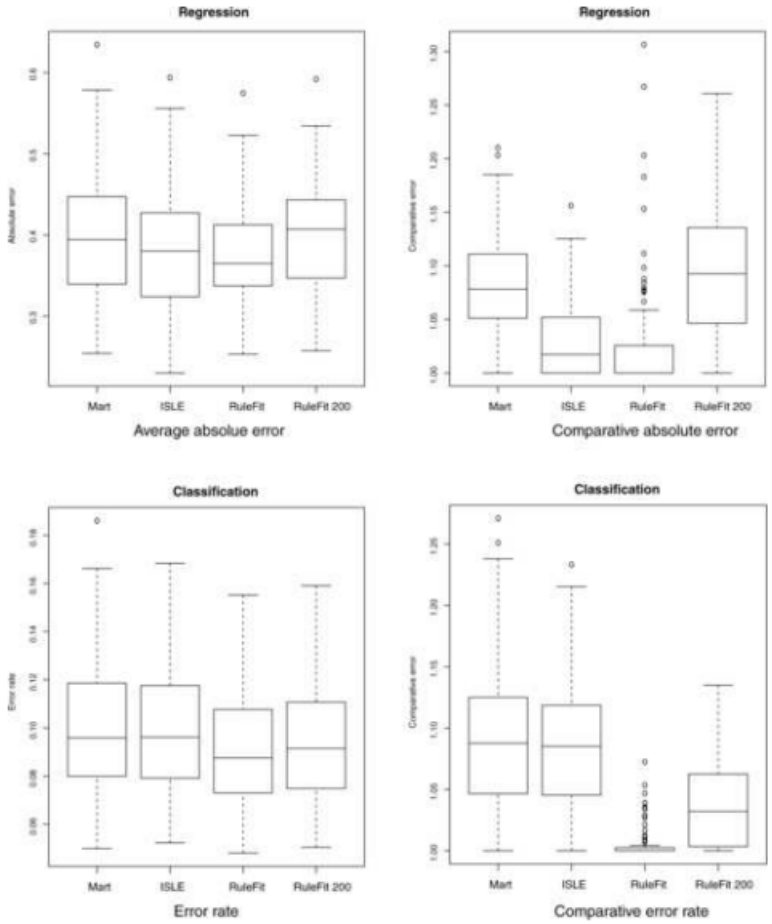


Figure 2.1: Inaccuracy comparisons between tree ensemble methods (Mart, ISLE) and rule based ensembles (RuleFit, RuleFit 200) [27].

performance of four based ensemble methods, two rule-based ensemble methods, and the two best tree-based ensemble methods over the 100 datasets used. The two tree-based ensemble methods were Multiple Additive Regression Trees (MART), which is a tree boosting method, and Importance Sampled Learning Ensemble (ISLE). The two rule-based ensemble methods were two different RuleFit methods, one that extracted the ten rules associated with each of the trees in the first 500 trees (resulting in 5000 rules) and one that extracted the ten rules associated with each of the trees but only in the first 200 trees (resulting in 2000 rules). By comparing the average absolute error, comparative absolute error, error rate and, comparative error rate, RuleFit (the one using the first 500 trees) was almost always the best in every dataset tested, followed by the RuleFit 200 in the classification problems. This comparison can be seen on figure 2.1.

These results suggest that the rule-based approach to ensemble learning produces comparable predictions to those based on decision trees.

2.1.1.2 Approximating Linear basis functions

Linear basis functions are among the most difficult ones for rule and tree-based ensembles to approximate. These functions usually have a substantial number of coefficients with relatively large absolute values and such targets can require a large number of rules for accurate approximation, especially if the training sample is not large and/or the signal-to-noise ratio is small.

A solution for this problem is to include the original variables $\{x_j\}_1^n$ as additional basis functions to complement the rule ensemble, this way the linear behavior is directly included in the model.

With these additions, the predictive model can be written as

$$F(x) = \hat{a}_0 + \sum_{k=1}^{K} \hat{a}_k r_k(x) + \sum_{j=1}^{n} \hat{b}_j l_j(x_j)$$

where $\{a_m\}_0^M$ correspond to the parameters specifying the particular linear combination, K is the number of rules, $r_k(x)$ are the binary features for the rules generated by the tree ensemble, n is the number of original variables and $l_j(x_j)$ are the features representing the original variables.

2.1.1.3 Relevance of rules

The rules generated represent easily understandable functions of the input variables, just like the linear functions. Besides that, the predictors have varying coefficients values depending on their estimated predictive relevance, so we can easily know which are the most important predictors. The relevance of a rule, I_k is given by

$$I_k = | \hat{a}_k | \cdot \sqrt{s_k(1 - s_k)}$$

where s_k is the rule support. For the linear predictors, the corresponding relevance is

$$I_j = | \hat{b}_j | \cdot \text{std} (l_j(x_j))$$

where std ($l_j(x_j)$) is the standard deviation of $l_j(x_j)$ over the data. These importances are global, meaning that they reflect the average influence of each predictor over the distribution of all joint input variables. To measure the influence of the rules at a specific point x, this is, the local influence, we can calculate

$$I_k(\text{x}) = | \hat{a}_k | \cdot | r_k(x) - s_k |$$

and for linear terms the corresponding is

$$I_j = | \hat{b}_j | \cdot | l_j(x_j) - \bar{l}_j |$$

where $\bar{l}_j$ is the mean of $l_j(x_j)$ over the training data. These quantities measure the absolute change in the prediction F(x) when the corresponding predictor is removed from the predictive equation.

Another important measure is the relative importance or relevance of the input variable J_l(x) to the predictive model. The most relevant input variables are those that preferentially define the most influential predictors appearing in the model. This measure can be calculated as

$$J_l\ (\mathrm{x}) = I_l\ (\mathrm{x}) + \sum_{x_l \in r_k} I_k(x)/m_k$$

where I_l(x) is the importance of the linear predictor involving x_l, I_k is the importance of the rule predictor and m_k is the total number of input variables that define the rule.

2.1.2 SIRUS

Stable and Interpretable Rule Set (SIRUS) [28] is an example of another algorithm that is based on a random forest and takes the form of a short and simple list of rules.

State-of-the-art learning algorithms are often black-boxes because of the high number of operations involved in their prediction process. This lack of interpretability may be highly restrictive for applications with critical decisions at stake. On the other hand, algorithms with a simple structure are known for their instability. This makes the conclusions of the data analysis unreliable and is a strong limitation.

Unlike RuleFit, SIRUS is not sensitive to data perturbations and does not produce a long, complex, and unstable list of rules. Therefore SIRUS can improve stability, which is described as the number of rules that are kept the same after a slight perturbation in the training set, and simplicity which refers to the number of rules kept by each model while still maintaining comparable accuracy.

Decision trees can model non-linear patterns while having a simple structure. They are therefore presented as interpretable. However, the structure of trees is very sensitive to small data perturbation. Rule algorithms are another type of nonlinear method with a simple structure, defined as a collection of elementary rules. An elementary rule is a set of constraints on input variables, this forms a hyperrectangle in the input space on which the associated prediction is constant. The problem with these algorithms is that even though they have a high predictivity and simplicity, they share the same limitations as decision trees, they are unstable.

Its general principle is that since each node of each tree of a random forest can be turned into an elementary rule, the idea is to extract the most frequent rules from the tree ensemble(the random forest). The most frequent rules represent robust patterns in the data and are linearly combined to form predictions.

2.1.2.1 SIRUS Algorithm

The objective of the SIRUS algorithm is to estimate the regression function with a small and stable set of rules.

The SIRUS algorithm can be divided into 4 main stages.

The first stage of SIRUS is to generate rules. SIRUS creates a random forest with a large number of trees based on the available sample D_n (n is the number of observations). A critical feature of this approach to guarantee the stability of the forest structure is to restrict node splits. After obtaining the forest, it is broken down into a large collection of rules. Since each node of each tree of the final ensemble defines a hyperrectangle in the input space, then, each one of them can be turned into an elementary regression rule by defining a constant estimate whose value solely depends on whether the query points are inside the hyperrectangle or not. The path P, which represents the order of splits to reach a node from the trees root, is used to represent a node in a tree.

The second stage of SIRUS is where the rules are selected from the random forest, since not all rules are revelant. Despite the randomization while construction the forest, there are redundant rules. The chosen rules will be those with a high frequency of appearance, since these represent strong and robust patterns in the data. The occurrence frequency is denoted by $\hat{P}_{M,n}(P)$ for each possible path $P \in \prod$ where $\prod$ represents the finite list of all possible paths. Then a threshold $p_0 \in (0,1)$ is used to select the relevant rules, that is

$$\hat{P}_{M,n,p0} = \{\, P \in \prod : {}_{\hat{p}M,n,}(P) > p_0 \,\}$$

The threshold p_0 is a tuning parameter whose optimal values select rules made of one or two splits. Greater sensitivity to data perturbation is related with rules that have more splits, therefore are also associated with lower values of $P_{M,n}(P)$. In conclusion, the selected rules will be those which have fewer splits, which are the rules less sensitive to data perturbations and therefore have better stability.

The third stage of SIRUS is the rule set post-treatment. The set of distinct paths $\hat{P}_{M,n,p_0}$ is dependent on the path extraction mechanism. The linearly dependent rules, this is, the rules that have overlapping hyperrectangles are filtered in this stage. Let r_a be the rule induce by the path $P \in \hat{P}_{M,n,p_0}$ SIRUS filters the rules with the following criteria: if r_a is a linear combination of rules associated with paths with a higher frequency of appearance, then P is removed from $\hat{P}_{M,n,p_0}$.

The fourth and last stage is rule aggregation. After the previous stage, we obtain a small set of regression rules. Each rule $\hat{g}_{n,P}$ associated with a path P is a piecewise constant estimate: if a query point falls inside the hyperrectangle defined by the rule, the rule returns the average of the Y_i's for the training points X_i's that belong to that hyperrectangle. In case the point falls outside the hyperrectangle defined by the rule, the rule returns the average of the Y_i's for training points outside of the hyperrectangle. After this, a non-negative weight is associated with each of the selected rules to combine them into a single estimate of $m(x)$. These weights are defined by the ridge regression

solution and constrained to be non-negative, where each predictor is a rule $\hat{g}_{n,P}$ for $P \in \hat{P}_{M,n,p_0}$. The aggregated estimate $\hat{m}_{M,n,p_0}(\mathrm{x})$ of $m(\mathrm{x})$ computed in this stage can be written in the form

$$\hat{m}_{M,n,p_0}(\mathrm{x}) = \hat{\beta}_0 + \sum_{P \in \hat{P}_{M,n,p_0}} \hat{\beta}_{n,P} \hat{g}_{n,P}(\mathrm{x})$$

where $\hat{\beta}_0$ and $\hat{\beta}_{n,P}$ are the solutions of the ridge regression problem that can be obtained from

$$(\hat{\beta}_{n,P}, \hat{\beta}_0) = \underset{\beta \geqq 0, \beta_0}{\operatorname{argmin}} \frac{1}{n} \parallel Y - \beta_0 1_n - T_{n,p_0}\beta \parallel_2^2 + \lambda \parallel \beta \parallel_2^2 \tag{2.2}$$

where $Y = (Y_1, ..., Y_n)^T$, $T_{n,p0}$ is the matrix whose rows are the rules values $\hat{g}_{n,P}(X_i)$ for $i \in \{1, ..., n\}$, $1_n = (1, ..., 1)^T$ is the n-vector with all components equal to 1 and λ is a positive parameter tuned by cross-validation that controls the penalization severity.

2.1.2.2 Comparing interpretable models

Just like in SIRUS, in RuleFit the rules are also extracted from a tree ensemble, the difference is that RuleFit uses the regularizer Lasso to aggregate them. The problem with using this regularizer is that Lasso is highly unstable when features are highly correlated and the rules are highly correlated by construction. This makes RuleFit unstable and is one of the advantages of SIRUS over RuleFit. SIRUS manages to stay stable by using the parameter p_0 to control sparsity and the ridge regression to enable a stable aggregation of the rules.

When comparing different interpretable algorithms we can use three different measures: their simplicity, stability, and predictivity. The simplicity of a model is the number of operations involved in the prediction mechanism, in a rule-based algorithm like SIRUS, the simplicity is given by the number of rules. For the stability, let $\hat{P}'_{M,n,p0}$ be the list of rules output by SIRUS on an independent sample D'_n. To measure the stability of the model we can use the Dice-Sorensen index and calculate the proportion of rules shared by $\hat{P}_{M,n,p0}$ and $\hat{P}'_{M,n,p0}$ with the following formula:

$$\hat{S}_{M,n,p0} = \frac{2|\hat{P}_{M,n,p0} \cap \hat{P}'_{M,n,p0}|}{|\hat{P}_{M,n,p0}| + |\hat{P}'_{M,n,p0}|}$$

Usually, we do not have access to an additional sample D'_n so we use 10-fold cross-validation instead to simulate data perturbation. The stability is given by the average proportion of rules shared by two models of two distinct folds of the cross-validation. This will give us a value of 1 if the same list of rules is selected over the 10 folds and a value of 0 if all rules are distinct between any 2 folds.

To calculate the predictivity of the model in regression problems we use 10-fold cross-validation and get the proportion of unexplained variance.

In the paper, [28] the authors compare SIRUS simplicity, stability, and predictivity with its two main competitors RuleFit and Node harvest over 8 different datasets. To have a baseline for predictive accuracy they also ran Random Forest and CART. The results of their experience can be seen in the tables 2.1, 2.2 and 2.3 where SIRUS stability is considerably better with much smaller rule lists than any of the other algorithms, therefore with better simplicity, while still maintaining a comparable predictivity to Node harvest and only slightly worse than RuleFit.

Dataset	CART	RuleFit	Node harvest	SIRUS	SIRUS sparse
Ozone	15	21	46	**11**	10
Mpg	15	40	43	**10**	10
Prostate	11	14	41	**9**	12
Housing	15	54	40	**6**	61
Diabetes	**12**	25	42	**12**	15
Machine	**8**	44	42	9	7
Abalone	20	58	35	**8**	13
Bones	17	5	13	**1**	1

Table 2.1: Mean model size over a 10-fold cross-validation for various public datasets. Minimum size and maximum stability are in bold ("SIRUS sparse" put aside). [28]

Dataset	RuleFit	Node harvest	SIRUS	SIRUS sparse
Ozone	0.22	0.3	**0.62**	0.63
Mpg	0.25	0.43	**0.77**	0.76
Prostate	0.32	0.23	**0.58**	0.59
Housing	0.19	0.40	**0.82**	0.82
Diabetes	0.18	0.39	**0.69**	0.65
Machine	0.23	0.29	**0.86**	0.84
Abalone	0.31	0.38	**0.75**	0.74
Bones	0.59	0.52	**0.96**	0.78

Table 2.2: Mean stability over a 10-fold cross-validation for various public datasets. Minimum size and maximum stability are in bold ("SIRUS sparse" put aside). [28]

The difference between algorithms that use a loss function and tree algorithms like SIRUS in learning is that instead of trying to minimize a loss function, SIRUS tries to choose splits that maximize the CART-splitting criterion. Starting from the root SIRUS tries to choose the best split in each node to reach another given node.

The set of all possible paths is defined as $\prod$. The probability $p^*(P)$ that a given path P belongs to a theoretical randomized tree in SIRUS is

$$p^*(P) = \mathbb{P}(P \in T^*(\theta))$$

where $T^*(\theta)$ is the list of all paths contained in the theoretical tree built with randomness θ in which splits are chosen to maximize the theoretical CART-splitting criterion. The theoretical set of selected paths is $P^*_{p0} = \{P \in \prod : p^*(P) > p0\}$, where $p0 \in [0, 1]$.

Dataset	Random Forest	CART	RuleFit	Node harvest	SIRUS	SIRUS sparse	SIRUS 50 rules
Ozone	0.25	0.36	**0.27**	0.31	0.32	0.32	**0.26**
Mpg	0.13	0.20	**0.15**	0.20	0.20	0.20	**0.15**
Prostate	0.48	0.60	**0.53**	**0.52**	**0.55**	0.51	**0.54**
Housing	0.13	0.28	**0.16**	0.24	0.30	0.31	0.20
Diabetes	0.55	0.67	**0.55**	**0.58**	**0.56**	0.56	**0.55**
Machine	0.13	0.39	**0.26**	**0.29**	**0.29**	0.32	**0.27**
Abalone	0.44	0.56	**0.46**	0.61	0.66	0.64	0.64
Bones	0.67	0.67	**0.70**	**0.70**	**0.73**	0.77	**0.73**

Table 2.3: Proportion of unexplained variance estimated over 10-fold cross-validation for various public datasets. For rule algorithms only, i.e., RuleFit, Node harvest, and SIRUS, minimum values are displayed in bold, as well as values within 10% of the minimum for each dataset ("SIRUS sparse" put aside). [28]

2.2 Related Work

Currently there have been multiple papers and studies developed exploring different black-box methods to predict the Emergency Departments (ED) waiting time and affluence in hospitals. We will see some of those studies in this chapter. We decided to divide these studies in two different categories, those which tried to predict the waiting time and those which tried predicting the affluence in the ED.

2.2.1 Predicting Affluence in Emergency Departments

We start by looking at some studies that tried to predict the number of people visiting the ED, either daily or monthly, in different hospitals around the world. We think it is important to look at these studies since the methods used to predict the number of people visiting an ED may also be investigated to predict the time the patients have to wait.

Yan Sun et al. [29] performed a study trying to predict the daily attendance at an emergency department to aid resource planning. They used time series models to predict daily attendance at the emergency department of a hospital in Singapore. The patients were divided into 3 categories, P1, P2, and P3 according to the patient acuity category scale, where P1 is the most severe category. They also included other data for the study like the public holidays, and weather factors (temperature, air quality, and relative humidity). The results showed that P1 did not have any weekly or yearly periodicity and the only relevant feature was the air quality. For P2 and for the model with all the data, this is, P1, P2 and P3 combined, they found a weekly periodicity and a correlation with the public holidays. Lastly, they found that the P3 attendances were significantly correlated with the air quality, the public holidays, the day of the month of the year, and the day of the week. Concerning the performance of the Autoregressive Integrated Moving Average (ARIMA) models, the best model was the one regarding all the acuity levels combined,

which achieved a Mean Absolute Percentage Error (MAPE) of 4.8%. On the other hand, the worst model was the one forecasting the P1 attendance, which had a MAPE of 16.8%. The P2 and P3 models had a MAPE of 6.7% and 8.6% respectively. This study showed that the daily attendance at the emergency department of this hospital was not predicted by weather conditions. Although this may be because Singapore is a tropical city, with hot and humid weather that does not have big variations throughout the year.

A study made in a medical centre in Southern Taiwan, made by Juang et al. [30], analysed the time series ARIMA to forecast the monthly visits of the following year. The statistical tests showed that six ARIMA models were candidate models to be the best, but the model ARIMA (0,0,1) had the minimum Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). This model resulted in a MAPE of 8.91%. Nevertheless, the authors stated that there were some possible influencing factors not incorporated, like epidemiological information, atmosphere changes and political issues, which could be medical policy, resource allocation or healthcare funding strategies.

With the goal of exploring and evaluating the use of multiple statistical forecasting methods when predicting the daily affluence on the ED, Jones et al. [31], compared the accuracy of multiple statistical forecasting methods with the accuracy of a previously proposed forecasting method at three different hospitals ED from the United States. The different methods evaluated were the Seasonal Autoregressive Integrated Moving Average (SARIMA), time series regression, exponential smoothing, and an artificial neural network models. The results revealed that all the models implemented had improved, even if slightly, the baseline model performance. The best method in the first facility was the SARIMA model, with a MAPE value of 13.89%, which represented an improvement of 0.51% compared to the baseline (14.40%). For the second and third facilities, the best performance was achieved by the time series regression model including climatic variables, with MAPE values of 8.91% and 8.52% respectively. These represented an improvement of 0.50% in the second facility and 0.61% in the third facility when compared to the baseline values which were 9.41% and 9.13% respectively. Furthermore, it was possible to conclude that introducing climatic variables in the time series regression model improved its performance on all three facilities when compared to the time series regression model without the climatic variables, thus, showing a correlation between them and the emergency department affluence.

On [32] Jin et al. evaluated three different time series models to predict the daily affluence in the Emergency Department (ED) of a Korean hospital. The first model was the ARIMA, the second and third models were two different SARIMA models, where one of them was univariate and the other multivariate. To evaluate how good the models fit, they used the AIC and BIC, which we will see in Section 4.2.1, while the forecast accuracy was measured using MAPE. The results showed that the best accuracy was achieved by the multivariate SARIMA, with the lowest MAPE ($\approx 7.4\%$). Moreover, they concluded that from the multiple variables in the multivariate SARIMA model, the most relevant were Chuseok, the average temperature, precipitation and season, and only those could

be selected as explanatory for this model. On the other hand, the worst model was the ARIMA model with a MAPE of 11.2%, while the univariate SARIMA model had a MAPE of $\approx 8.5\%$.

The last study we analyzed regarding affluence in the emergency department was made by Carvalho et al. [33]. They study the assessment of forecasting models to predict the patients arrival at the emergency department of a hospital in Portugal, the Braga Hospital. The models used in this study were different models of the ARIMA, built with data regarding the emergency department arrivals from 2012 to 2013 and tested in data from 2014. The results showed that the best ARIMA models to predict the patients arriving at the emergency department were the ARIMA (1,1,1) and the ARIMA (1,0,1) with a MAPE (mean absolute percentage error) of 5.92% followed by ARIMA (0,0,1) and ARIMA (1,0,0) with a MAPE of 8.28%. All these ARIMA models were able to outperform both the moving average and the exponential smoothing, which achieved a MAPE of 9.82% and 10.23% respectively.

2.2.2 Predicting Waiting Time in Emergency Departments

Predicting the waiting time in ED is one of the focus of this thesis, and thus, we decided to look at what have been made so far and what were the results obtained when trying to make this prediction.

One of these studies was made by Yong-Hong Kuo et al. in [34] where they present multiple machine learning models to predict the real-time and personalized waiting time in an emergency department. They found that machine learning models were more effective than linear regression models to predict the waiting time. Moreover, they say that the knowledge of the emergency department system may improve the performance of the prediction model. The authors believe that introducing this concept of systems thinking, this is, the usage of knowledge and principles of how the system works, can enhance the performance of the algorithm. The models tested in this study were the Linear Regression model, used as baseline model, feedforward artificial neural networks, Support Vector Machines (SVM) and Gradient Boosting Machines (GBM). They tested these models on a dataset collected from the Emergency Department of the Prince of Wales Hospital in Hong Kong. The results showed that the three machine learning models (SVM, GBM and ANN) were not significantly better than the baseline model (linear regression) when using the dataset without the system knowledge. However, when they included the new features that represent the system knowledge, the machine learning models performance had a significantly improvement. While the linear regression model reduced its Mean Squared Error (MSE) about fifteen percent, the three machine learning models achieved around twenty percent less MSE than the the linear regression model with these new features. The Gradient Boosting model had the least MSE (around 4.383) with these new features, which was around 5.598 before introducing the system knowledge, thus, representing an improvement of almost 50% percent with the new features. These results clearly show

us that introducing system knowledge in the machine learning models clearly improved the models performance, and thus, showing us the importance to do an interdisciplinary research, joining data scientists that develop algorithms with domain experts of the field that can share their knowledge and provide feedback. Although machine learning models were able to perform better in the predictions, we must not forget that these are "black-box" models, and thus, do not provide an explanation on why a prediction was made, which is a critical problem in the healthcare environment.

Sun et al. [35] tried to use quantile regression to predict the waiting time in real-time in the emergency department of a hospital in Singapore. In the dataset of this hospital, the patients were categorized into 3 different categories, acuity category 1,2, and 3. Patients with acuity category 1 are the most critical and have the highest priority, followed by category 2 and then 3. It is important to note that the patients of category 3 are cared for by a different team of physicians, which may affect the waiting time. They obtained a median absolute prediction error of 11.9 minutes for the acuity category 2 and 15.7 minutes for the acuity category 3. These were good results for the authors that showed that using scant clinical information extracted from the existing emergency department system resulted in obtaining good accuracies for the waiting time with the quantile regression.

Another article, made by Pak et al. [36], studied the prediction of waiting time for treatment of patients in the emergency department (ED) of a hospital in Australia. In this study, they implemented five different algorithms: ordinary least squares, Ridge regression, LASSO regression, quantile regression, and Random Forest. They compared the prediction of these models with the rolling average and the median estimators, which have limited accuracy. In order to implement the Machine Learning algorithms, they used a large set of variables, including queueing and service flow variables, which they wanted to prove that improve the prediction of the waiting time. They measured the results using the Mean Squared Prediction Error (MSPE) and MAPE and observed that all five estimators outperformed significantly the rolling average. In fact, the LASSO regression, which obtained the best results, reduced the MSPE by 21%. Moreover, the quantile regression reduced the number of patients with underpredicted waiting times by 42%. With these results, they obtained clear evidence that these estimators have better accuracy in predicting the waiting time in ED than the rolling average. Furthermore, they also showed that using variables carefully constructed that represent a complex emergency department environment, like the patient queue development, the service rates, the characteristics of patients, and the diurnal fluctuations, is critical for the models to produce better waiting time forecasts. Lastly, their results also showed that the predictive accuracy improved when the emergency department operates at full capacity.

A Q-Lasso model, which combines statistical learning and fluid model estimators, was proposed by Ang et al. in [37] to predict the waiting time in ED. They tested this model in four hospitals and compared its accuracy with the rolling average accuracy. The results showed that in all four hospitals, Q-Lasso had a better accuracy than the rolling average,

even though it still had a large error. It improved the rolling average accuracy by 33.3% in the first hospital, 25.11% in the second hospital, by 12.0% in the third hospital and by 13.2% in the fourth hospital. The higher improvements in the first two hospitals, are due the fact that these exhibit a higher variance in the waiting times. This higher variance, gave the opportunity for Q-Lasso to explain the variation of the waiting time through predictor variables that were able to capture conditions and events in the emergency department (ED), which the rolling average does not. Moreover, they showed that Q-Lasso is capable of tracing the diurnal fluctuation of the waiting time much better than the rolling average.

A study conducted towards a real-time prediction of waiting times in ED [38] analyzed different machine learning techniques. This study used data from two Italian hospitals and evaluated the predictive ability of five different models: LASSO, Random Forest, Support Vector Regression, Artificial Neural Networks, and the Ensemble Method, which ensemble all the previous methods, assigning a weight to each one of them. They also introduced queue-based variables that captured the current state of the emergency department. To identify the technique that provided the best prediction and best real-time estimations of the waiting times they used two forecasting error measures, the MSE and the MAE. In this study, they also compared the different techniques in terms of computational time. In terms of accuracy, the results showed that the Ensemble Method outperformed all other techniques since it achieved the lowest for both metrics. For the first dataset tested in this study, the Ensemble Method improved the accuracy of Ordinary Least Square (OLS) between 23 and 28%, while for the second dataset it improved between 14 and 17%. LASSO was the worst technique of the machine learning techniques implemented since it only reduced the error from OLS up to 6% on the first dataset and 0.6% on the second dataset. The Random Forest and Support Vector Regression performed very similarly in both datasets. Even though the Ensemble Method obtained the best results in terms of accuracy, it was the technique that took the longest computational time while LASSO was the fastest model running. Despite taking a longer time to run, which was expected since it combines the other four techniques, these results show that the Ensemble Method was the most accurate predictor of waiting times, significantly outperforming all the others.

3

Exploratory Data Analysis

In this chapter, we will be exploring and analyzing the data that will be used in this thesis. The dataset used in this work has information from the Portuguese National Health Service (Serviço Nacional de Saúde(SNS)) taken from their website. In this dataset, we have information about the waiting time and number of people waiting in the Emergency Department of 4 hospitals in Portugal: Santa Maria, São Francisco Xavier, Hospital Dona Estefânia, and São José. These are the 4 main hospitals in Lisbon. The data we will be analyzing corresponds to the time period from November 15, 2017, until September 24, 2019.

3.1 Overall Statistics

Each record in the dataset has information regarding the number of people waiting and the average waiting time in the last 2 hours in each Emergency Stage level, which goes from 1 to 5 and corresponds to the Manchester Triage color (where 1 corresponds to blue, 2 to green, 3 to yellow, 4 to orange and 5 to red). A sample from the dataset can be seen in the figure 3.1, where we can see the fields of the record, this is, the variables that we are going to use. We have 2 numerical and 6 categorical variables which are described in the table 3.1 with their respective meaning.

Tables 3.2 and 3.3 show us some summary statistics on our data regarding each Emergency Stage.

In the table 3.2 we can observe that the emergency stages 1, 2, and 3 have their average value under the maximum value that the Portuguese national health service tries to keep for the hospitals analyzed in this dataset (which is 240, 120 and 60 minutes respectively) but the Emergency Stage 4 has it mean waiting time over that maximum value which is 10 minutes. The Emergency Stage 5 maximum value is 0 but we have 4 registries with 1 minute in the 291 registries that we have from this Emergency Stage. Even though we have 4 registries with this value, they are all within the same 2 hours interval in the same

	Acquisition_Time	Hospital	Urgency_Type	Service	Emergency_Stage	Waiting_Time	People_Waiting	H_Name
0	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Medicina Interna	4	17	0	S Jose
1	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Medicina Interna	3	34	0	S Jose
2	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Cirurgia Geral	3	6	0	S Jose
3	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Oftalmologia	3	5	0	S Jose
4	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Medicina Interna	2	309	8	S Jose
5	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Ortopedia	2	4	0	S Jose
6	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Otorrinolaringologia	2	4	0	S Jose
7	11/15/2017 0:01	211	Urgncia Polivalente	Espera: Pequena Cirurgia	2	86	0	S Jose
8	11/15/2017 0:02	218	Urgncia Central	Cirurgia	4	16	0	Santa Maria
9	11/15/2017 0:02	218	Urgncia Central	Medicina	4	12	0	Santa Maria
10	11/15/2017 0:02	218	Urgncia Central	Cirurgia	3	26	1	Santa Maria
11	11/15/2017 0:02	218	Urgncia Central	Medicina	3	33	1	Santa Maria
12	11/15/2017 0:02	218	Urgncia Central	Cirurgia	2	30	1	Santa Maria
13	11/15/2017 0:02	218	Urgncia Central	Medicina	2	144	2	Santa Maria
14	11/15/2017 0:02	218	Urgncia Central	Medicina	1	270	0	Santa Maria
15	11/15/2017 0:03	216	HSFX - Urgncia Geral	Aguarda Balcao (Cirurgia)	3	30	0	SFX
16	11/15/2017 0:03	216	HSFX - Urgncia Geral	Aguarda Atendimento Geral	2	85	1	SFX
17	11/15/2017 0:03	216	HSFX - Urgncia Geral	Aguarda Pequena Cirurgia Geral	2	93	0	SFX
18	11/15/2017 0:03	216	HSFX - Urgncia Geral	Aguarda Balcao (Medico)	2	40	0	SFX
19	11/15/2017 0:03	216	HSFX - Urgncia Geral	Aguarda Cirurgia Plastica	2	68	0	SFX
20	11/15/2017 0:11	211	Urgncia Polivalente	Espera: Medicina Interna	4	13	0	S Jose

Figure 3.1: Sample from the dataset.

Variable name in the record	**Description**
Acquisition_Time	Timestamp of the observation
Hospital	Number of the hospital
Urgency_Type	Type of urgency
Service	Type of service
Emergency_Stage	Level of emergency, which corresponds to the Manchester Triage (1 is blue, 2 is green, 3 is yellow, 4 is orange, 5 is red)
Waiting_Time	Average waiting time for the past two hours
People_Waiting	Number of people waiting at the time
H_Name	Name of the hospital

Table 3.1: Variables in each record of the dataset and their corresponding description.

hospital, so we can conclude that it only happened once in the time period studied and it is uncommon that someone in this emergency stage needs to wait at all.

	Emergency Stage 1	**Emergency Stage 2**	**Emergency Stage 3**	**Emergency Stage 4**	**Emergency Stage 5**
Mean	123.47	55.81	38.02	16.49	0.01
Standard deviation	126.53	62.38	41.04	20.99	0.12
Minimum	0	0	0	0	0
Maximum	1046	716	910	518	1

Table 3.2: Statistics regarding the waiting time (in minutes) in each emergency stage.

Table 3.3 shows us that stages 2 and 3 have a higher mean number of people waiting than stage 1. Despite that, from table 3.2 we can see that they have a lower waiting time.

This is because they are more severe cases that can not wait as long and need priority over stage 1 cases.

	Emergency Stage 1	Emergency Stage 2	Emergency Stage 3	Emergency Stage 4	Emergency Stage 5
Mean	0.9	3.60	2.31	0.39	0.08
Standard deviation	1.33	5.80	3.92	0.797	0.26
Minimum	0	0	0	0	0
Maximum	13	55	41	10	1

Table 3.3: Statistics regarding the number of people waiting in each emergency stage.

In Figure 3.2 we have the number of observations of each Emergency Stage in our dataset. We can see that Emergency Stage 5 is so low (291) compared to the other emergency stages that we can not even see its bar. We can also clearly see that most entries in this dataset are from Emergencies Stages 2 and 3.

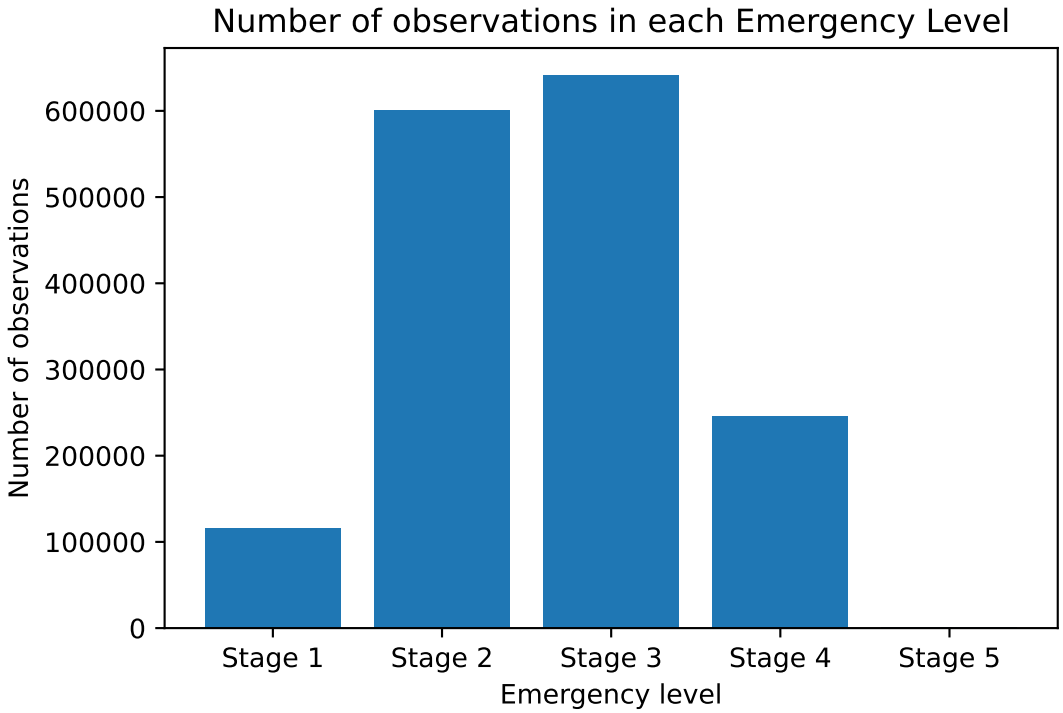


Figure 3.2: Total number of entries per year in the different emergency levels.

The distributions of the continuous variables of our dataset, this is, the waiting time and the number of people waiting, can be seen in Figures 3.3 and 3.4 respectively. These plots seem to have approximately Power Law distributions, which means that there are many observations with a low waiting time and a low number of people waiting and rare observations with a high waiting time and number of people waiting.

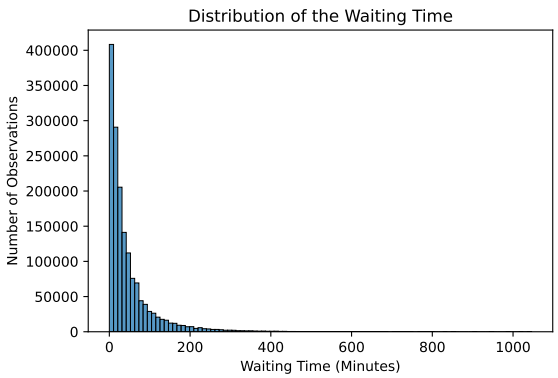


Figure 3.3: Distribution of the waiting time in the dataset.

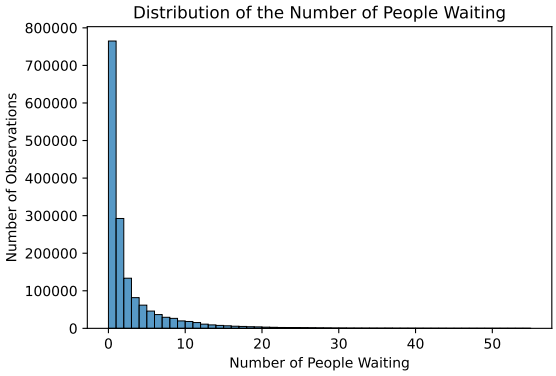


Figure 3.4: Distribution of the number of of people waiting.

In Figure 3.5 we can see the relation between the waiting time and the Emergency Level. It is clear that the higher the Emergency Level, the lower the waiting times tend to be, which is something expected since the higher the Emergency Level the more severe a case is and can not wait as long. On the other hand, if we analyze the people waiting in each of the Emergency Levels, in Figure 3.6, we can see that the same does not happen. Emergency level 2 seems to be the one with more people waiting followed by Emergency Level 3, but since they are more severe cases than Emergency Level 1 cases, they have priority and are taken care of first, thus, they do not have waiting times as long as Emergency Level 1.

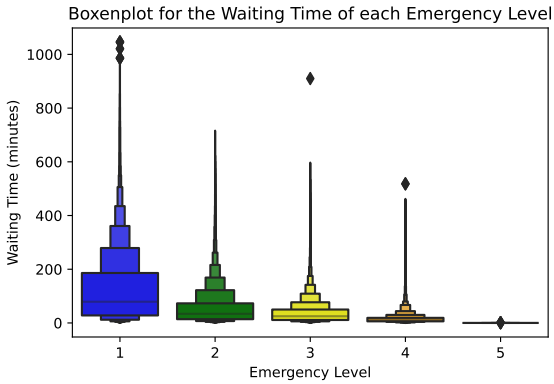


Figure 3.5: Relation between the waiting time and the Emergency Level.

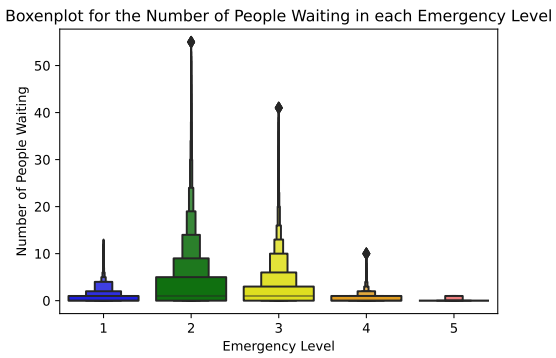


Figure 3.6: Relation between the number of people waiting and the Emergency Level.

3.2 Seasonalities

Next, we are going to study how these waiting times and the number of people waiting changed throughout the years and how they are related to the time of the year, month, week, and day.

We can see in Figure 3.7 that throughout the years the higher the emergency stage of a patient, the less time the mean of the waiting time was. This relation makes sense since the higher the emergency stage, the more urgent the case is and can not wait as long as the other emergency stages. We can visualize the values from table 3.2 in this Figure (3.7). Even though in the table they refer to the entire dataset, these values stayed almost the same throughout these 3 years.

While the average waiting time in the different Emergency Stages seemed to be almost constant throughout the years, the average number of people has not. In Figure 3.8 we can see that Emergency Level 4 has been having on average slightly more people waiting each year. We can also see that in 2018 more people were waiting on average on Emergency Levels 1 and 5 than in 2017 and 2019. On the other hand, in 2018, fewer people were waiting on average on Emergency Level 3.

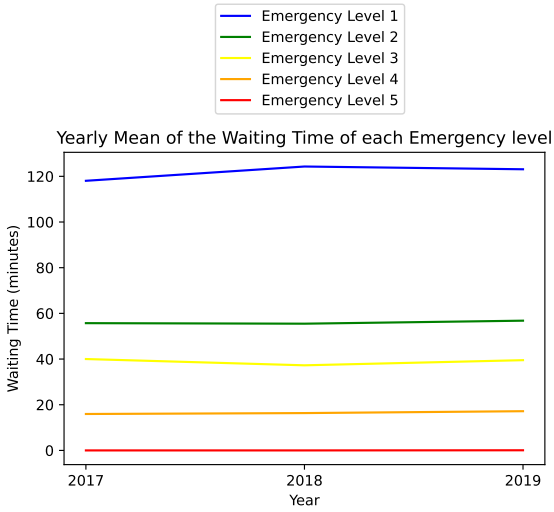


Figure 3.7: Average waiting time in minutes throughout the years in each Emergency Stage.

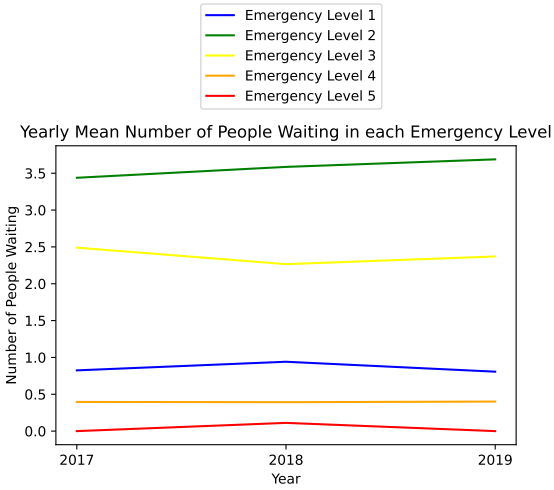


Figure 3.8: Average number of people waiting throughout the years in each Emergency Stage.

In Figure 3.9 we can observe the monthly behavior of the average waiting time in the different Emergency Levels. This value seems to be higher in the winter months (from November to February) in the Emergency levels 1, 2, and 3, and also increases at the end of the summer (August and September), but have their peak in the winter. Emergency level 5 has an average of 0 minutes waiting every month and never changes while Emergency level 4 has a spike in June, unlike the other emergency levels which seem to have a lower waiting time this month.

When studying the monthly number of people waiting in Figure 3.10 we can notice that the months with more people waiting are the winter months for Emergency Levels 2,3 and 4. Emergency Level 1 seems to have a huge pike in June and stays high in the summer contrary to every other Emergency Level. Emergency level 5 has a peak in August but its value is low all over the year and even tho the number of people waiting might be

higher than 0, the people in this Emergency level never have to wait any time, as seen in Figure 3.9.

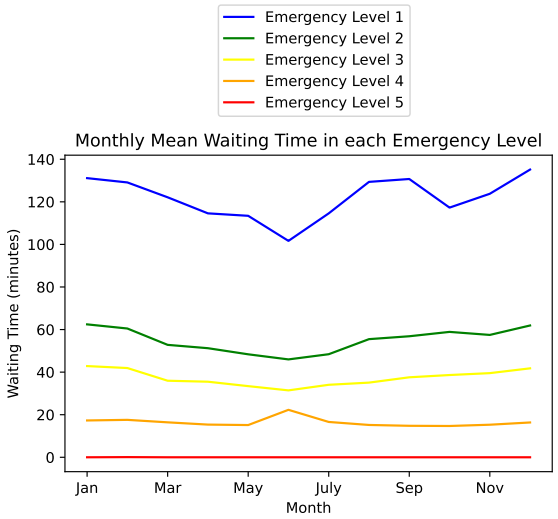


Figure 3.9: Relation between the mean waiting time with the months of the year.

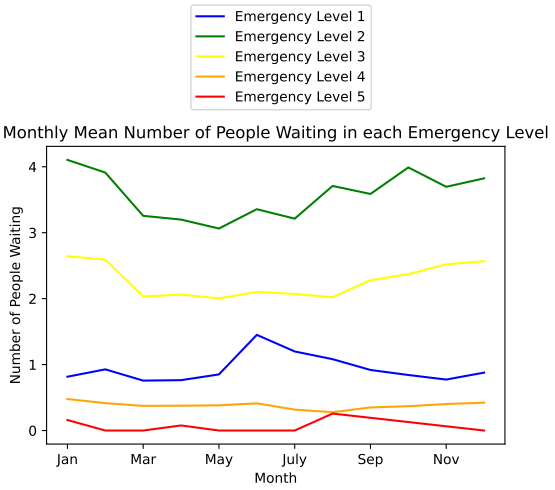


Figure 3.10: Relation between the mean number of people waiting with the months of the year.

In Figure 3.11 we can see the relation between the weekday and the waiting time in each of the Emergency Levels. Tuesday is the day of the week with a higher average waiting time for Emergency Levels 1,2 and 3. That value seems to slowly decrease during the rest of the week until Sunday for Emergency Levels 2 and 3 while the Emergency Level 1 waiting time decreases a lot on Friday but increases during the weekend. The average waiting time for Emergency Levels 4 and 5 seems to be the same every day of the week.

If we look at the relation between the weekday and the average number of people waiting in Figure 3.12 we notice that this number is high at the beginning of the week (Monday) and slowly goes down during the week for the Emergency Levels 1, 2 and 3. The average number of people waiting for the Emergency Level 4 is the same every day of the week. For Emergency Level 5, this number seems to be different from zero on Fridays and Sundays, but even when people are waiting on this Emergency Level, they do not have to wait any time as we can see in Figure 3.11.

In Figure 3.13 we can see the relation between the average waiting time and the day of the month. In that figure, we can observe that there seem to be some days where people have to wait more on Emergency Level 1. Most of these days seem to be at the beginning and end of the month, but also on day 12. When analyzing the line of Emergency Level 2 we see that it also slightly increases in the last days of the month. For Emergency levels 3,4 and 5 the waiting time seems to be almost the same every day of the month.

When analyzing the relation between the average number of people waiting and the day of the month in Figure 3.14 we notice again that in the beginning and right before the

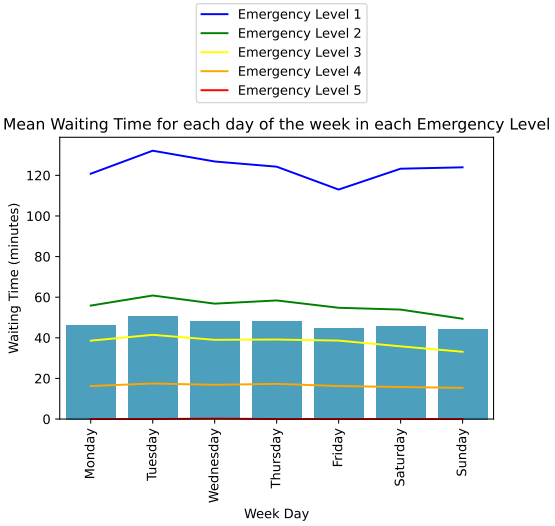


Figure 3.11: Relation between the mean waiting time with the day of the week.

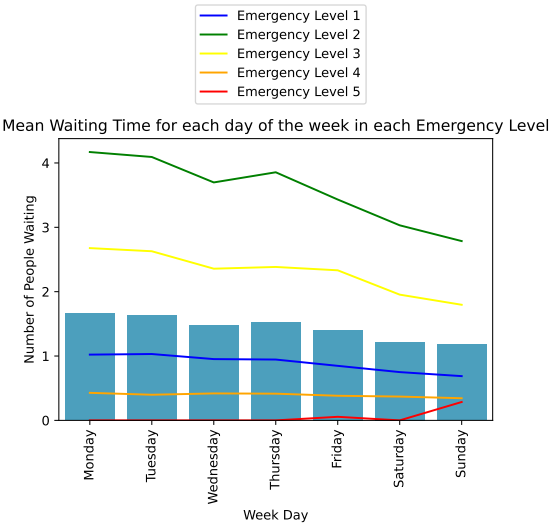


Figure 3.12: Relation between the mean number of people waiting with the day of the week.

Figure 3.13: Relation between the average waiting time with the day of the month.

end of the month we have more people waiting in the Emergency Levels 2 and 3. On the last days of the month (30 and 31), we have a decrease in the average of people waiting in all Emergency Levels except Emergency Level 5. The mean of Emergency level 5 on day 14 of the month can not be completely trusted since we only have 1 entry on the dataset for the 14th day of a month with Emergency Level 5 in the 18 months of data. That means since that single entry has the value of 1 on the number of people waiting, the average of that day will also be 1.

Regarding the daily values, we can see in Figure 3.15 that the average waiting time

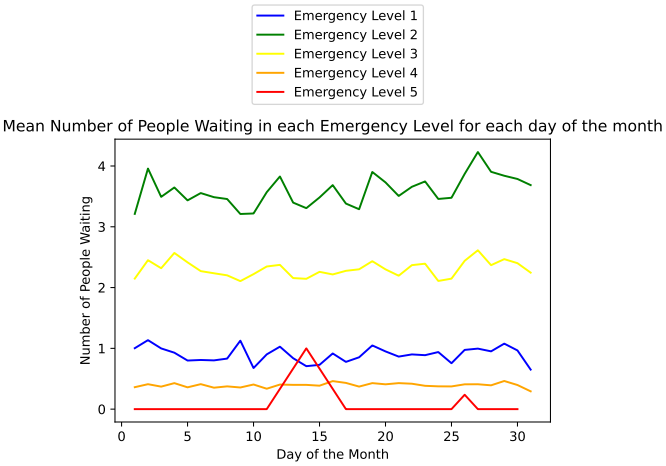


Figure 3.14: Relation between the average number of people waiting with the day of the month.

in Emergency Levels 1,2,3, and 4 move together and have their peak at around 3-4h and their lowest value around 11h. The average waiting time in the Emergency Level 5 seems to be 0 during the whole day. The daily behavior of the average number of people waiting in each of the Emergency Levels seems to be very different from the average waiting time. While the average waiting time seems to be higher in the early hours of the day, at these hours the number of people waiting seems to be the lowest of the day, having its minimum at around 6h, as can be seen in figure 3.16. The maximum of these values for Emergency levels 1,2,3 and 4 seem to be around 14h to 20h while for Emergency Level 5 it looks at around 13h to 14h.

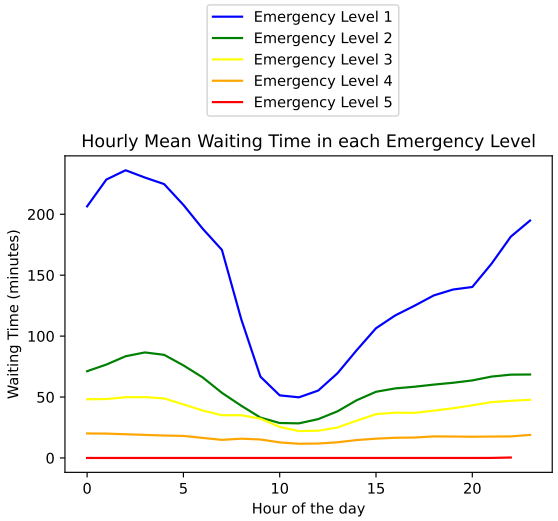


Figure 3.15: Daily behavior of the waiting time in each Emergency Level.

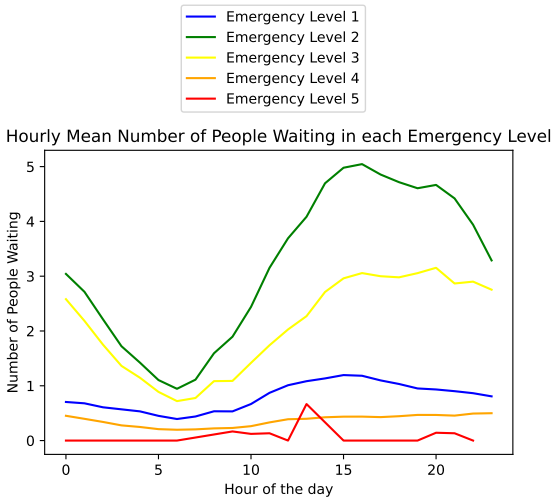


Figure 3.16: Daily behavior of the number of people waiting in each Emergency Level.

In Figures 3.17 and 3.18 we analyze the hospitals individually. We can observe that

both the waiting time and the number of people waiting seem to change together in every hospital according to the hour of the day. Regarding the Waiting Time they all seem to have their highest value at night and during the early hours of the day except for the children's hospital "Estefania " which also has one of its highest waiting time values in the afternoon after 14h. Analyzing the number of people waiting we can see that they all have their lowest during the early hours of the day and start having their highest values after 10, reaching their peak around lunch hour (13h) and in the hospital "S Jose" at dinner time (around 20h).

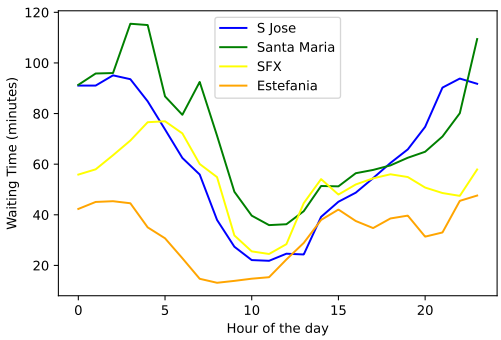


Figure 3.17: Daily behavior of the waiting time in each Hospital.

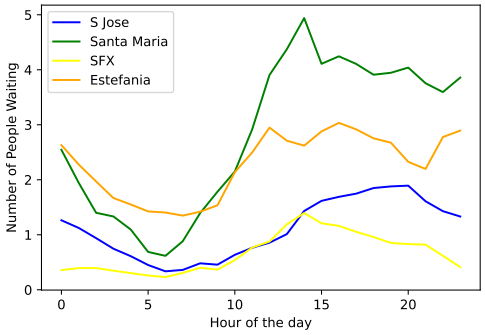


Figure 3.18: Daily behavior of the number of people waiting in each Hospital Level.

3.3 Hospital Analysis

When we analyzed the Hospitals individually, in figure 3.19, we noticed that of the 4 hospitals in the dataset, the hospital "Santa Maria" has the highest mean waiting time (69.60 minutes), followed by "São Francisco de Xavier" (42.36 minutes) and "São José" (42.29 minutes). On top of that, the hospital "São José" seems to have the longest maximum waiting time (1046 minutes) and the greatest amount of outliers.

Concerning the number of people waiting in each of the hospitals, in Figure 3.20, we can see that the hospital "Santa Maria" has both the highest mean (4.15) and maximum (55) of people waiting. On the other hand, the hospital "São Francisco de Xavier" has the lowest mean (1.27). While the hospital "Estefânia" has the second highest mean of people waiting (3.19), as we saw in Figure 3.19, it has the lowest mean waiting time (31.12) by over 10 minutes. These values could indicate that this hospital makes the most efficient management of the patients, as it is the second hospital with more people waiting and at the same time, the hospital with the minimum waiting time.

To deepen our analysis of the waiting time in each of the hospitals, we studied how the waiting time changed according to the type of service in every hospital. In Figure 3.21 we can clearly see that the hospital "Santa Maria" has always the highest average waiting

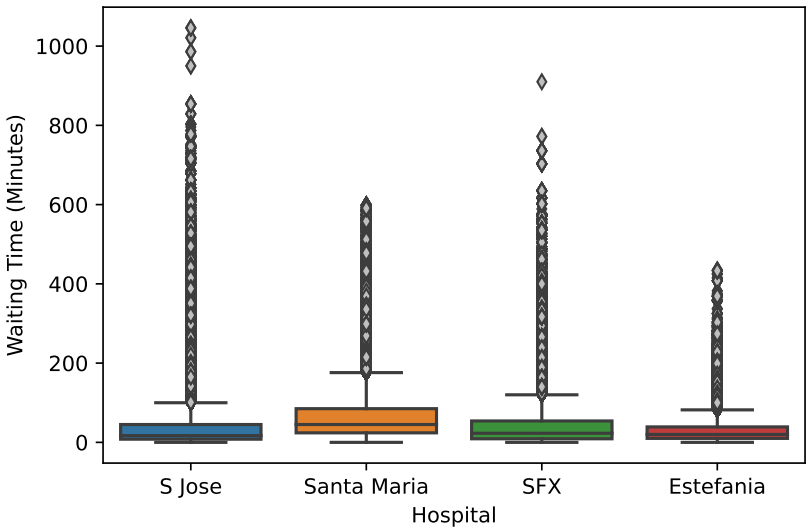


Figure 3.19: Relation between the waiting time and the Hospital.

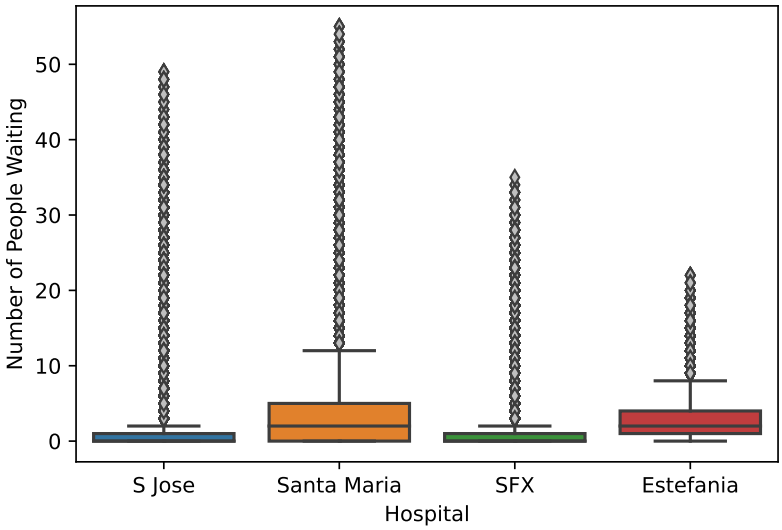


Figure 3.20: Relation between the number of people waiting and the Hospital.

time both in "Cirurgia" and "Medicina". On the other hand, the hospital "Estefania" has the lowest average waiting time in both of these services. Moreover, we notice that the hospitals that have the lowest and highest average waiting time in one type of service, also have it on the other. Despite that fact, the hospital "São José" has almost the same waiting time in the service "Cirurgia" as "Estefania", which is the lowest of the 4 hospitals and has the second highest average waiting time in "Medicina". This shows us that the waiting time in these two services is not necessarily correlated.

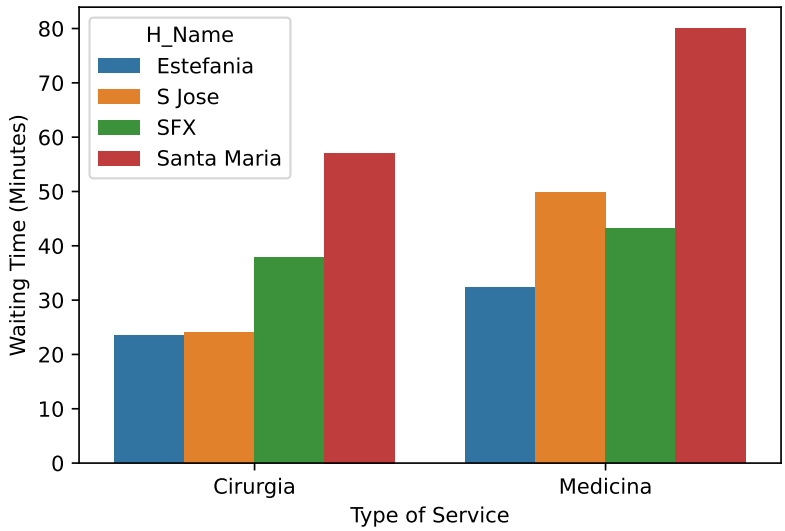


Figure 3.21: Relation between the waiting time and the service in each Hospital.

The same study was made for the average number of people waiting. As can be seen in Figure 3.22, the hospital "Santa Maria" is not only the hospital with the highest average waiting time in both services but also the highest average number of people waiting in both of them, followed by "Estefania". The hospital "Estefania"surprised us here in a positive way, because even tho this hospital has the second highest average number of people waiting in both services, it has the lowest average waiting time as we saw in figure 3.21. On the other hand, the hospital "Sao Francisco Xavier (SFX)" even though it has the lowest average number of people waiting in both services, it has the second highest average waiting time in "Cirurgia" and the third highest in "Medicina", which indicates that they might have more complex situations to manage there.

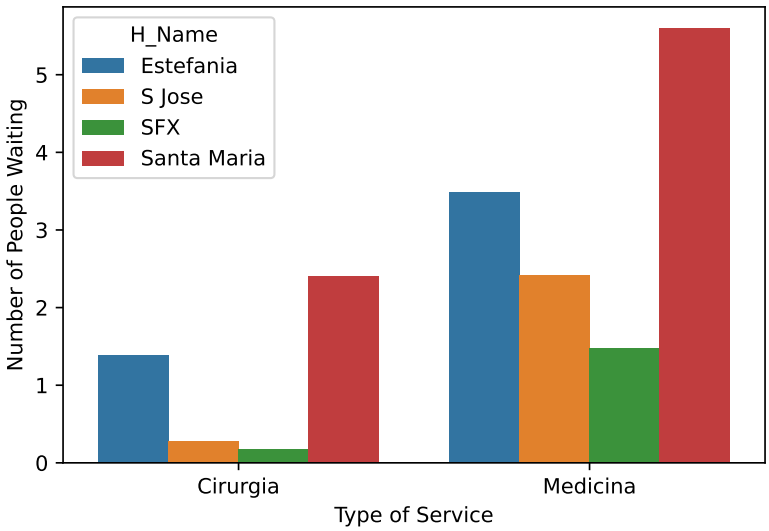


Figure 3.22: Relation between the number of people waiting and the service in each Hospital.

4

Methodology

4.1 Dataset Pre-Processing

In Chapter 3 we analyzed the whole dataset used in this thesis, however, we only use part of it, specifically the rows regarding the hospital "Santa Maria". From the 395.858 rows about the Santa Maria hospital, we removed 158 where the Service was not either "Cirurgia" or "Medicina" and did not make sense in our data. This left us with data describing the waiting times of the hospital "Santa Maria" from November 15, 2017, until April 29, 2019.

We did not have to remove outliers since those which were very high, like 1046 minutes, observed in chapter 3, were not found in the data regarding this hospital ("Santa Maria"). Nevertheless, we had some very high values, like 599 minutes, but those are data points that we must take into consideration since they can happen again. A sample from the dataset after extracting only the data regarding the hospital "Santa Maria" can be seen in Figure 4.1.

	Acquisition_Time	Hospital	Urgency_Type	Service	Emergency_Stage	Waiting_Time	People_Waiting	H_Name
2	11/15/2017 0:02:22	218	Urgncia Central	Cirurgia	4	16	0	Santa Maria
3	11/15/2017 0:02:22	218	Urgncia Central	Medicina	4	12	0	Santa Maria
4	11/15/2017 0:02:22	218	Urgncia Central	Cirurgia	3	26	1	Santa Maria
5	11/15/2017 0:02:22	218	Urgncia Central	Medicina	3	33	1	Santa Maria
6	11/15/2017 0:02:22	218	Urgncia Central	Cirurgia	2	30	1	Santa Maria
7	11/15/2017 0:02:22	218	Urgncia Central	Medicina	2	144	2	Santa Maria
8	11/15/2017 0:02:22	218	Urgncia Central	Medicina	1	270	0	Santa Maria
9	11/15/2017 0:12:36	218	Urgncia Central	Cirurgia	4	16	0	Santa Maria
10	11/15/2017 0:12:36	218	Urgncia Central	Medicina	4	12	0	Santa Maria
11	11/15/2017 0:12:36	218	Urgncia Central	Cirurgia	3	26	1	Santa Maria
12	11/15/2017 0:12:36	218	Urgncia Central	Medicina	3	33	1	Santa Maria
13	11/15/2017 0:12:36	218	Urgncia Central	Cirurgia	2	30	1	Santa Maria
14	11/15/2017 0:12:36	218	Urgncia Central	Medicina	2	144	2	Santa Maria
15	11/15/2017 0:12:36	218	Urgncia Central	Medicina	1	270	0	Santa Maria

Figure 4.1: Sample from the dataset regarding only the hospital "Santa Maria".

This dataset has information about the different emergency stages in the emergency department updated every 10 minutes, with a granularity that goes down to the seconds.

Predicting the waiting time down to the second would not allow us to get accurate predictions, so, it was opted to group the data by day, as done in previous studies (seen in Chapter 2.2).

Since we had the information regarding the different emergency stages, besides grouping the data by day, we decided to split it according to the emergency stage, to get a better and more specific prediction. Even though the original dataset had 5 different emergency stages, as there are 5 different emergency stages in the Manchester Triage, we only included stages 1,2,3, and 4, since emergency stage 5 did not contain enough observations to allow for a satisfactory and reliable prediction. This resulted in the creation of 4 different datasets, 1 for each emergency stage, describing the waiting times of 531 days (from November 15, 2017, until April 29, 2019).

Having the different types of service in each emergency stage made us wonder if that information would allow our models to have even better accuracy, so, besides the above datasets created, we created another 4 datasets (1 for each emergency stage from 1 to 4) that also had the data grouped by day, but with the difference that for each day now we had 2 rows, one regarding the service "Cirurgia" and another regarding the service "Medicina". These datasets, with the information about the service in each emergency stage, were only used to compare the interpretable models since the time series models did not allow for multiple observations in each time step.

Moreover, we derived some attributes from the "Acquisition_Time" column and added some columns to the dataset. These columns were the weekday, the day of the month, the month of the year, and the season (Autumn, Winter, Spring, and Summer). Furthermore, the missing entries in all the datasets were imputed with the mean value of the respective column. This was necessary since there are models that do not accept empty rows and need information regarding every day (this is, they can not have missing days in a time interval). Since we wanted to have the same data for all models, we used this dataset even in the models that could handle missing entries. The non-numerical variables, like the values from the "Season" column, were coded into numerical variables (for example, the values from the season column were coded into 1,2,3 and 4) on the models that could not process non-numerical variables.

The mean, standard deviation, minimum and maximum for each emergency stage in the first 4 datasets can be seen in table 4.1, while the mean, standard deviation, minimum and maximum for each emergency stage of the second group of datasets, where the services were kept apart, can be seen in table 4.2.

	Emergency Stage 1	**Emergency Stage 2**	**Emergency Stage 3**	**Emergency Stage 4**
Mean	126.41	81.26	62.75	25.90
Standard deviation	52.71	26.90	20.28	10.61
Minimum	38.22	33.59	22.72	8.91
Maximum	357.39	203.73	181.84	76.14

Table 4.1: Statistics regarding the first group of datasets.

	Emergency Stage 1	Emergency Stage 2	Emergency Stage 3	Emergency Stage 4
Mean	126.41	81.26	62.75	25.90
Standard deviation	76.29	39.65	29.48	18.66
Minimum	9.56	21.56	14.0	1.0
Maximum	506	279.83	241.35	134.67

Table 4.2: Statistics regarding the second group of datasets.

Lastly, note that all the models implemented in the following sections will try to predict the Emergency Department waiting time and will be compared using the mean absolute error (MAE). Moreover, since the variation of the waiting time values is higher in the second group of datasets (as can be seen in 4.1 and 4.2), we expect to have greater MAE values in the predictions made in this group of datasets than in the first group of datasets where the services were grouped and the variation is lower.

4.2 Baseline Non Interpretable models

As stated in [39], finding an accurate and simple model is almost always harder than finding an accurate but complex model. In their work, they define the Rashomon set, which is the set of almost equally accurate models, and if that set is large, it contains many accurate models where one of which may be the simple model we are looking for. To define the Rashomon set for future interpretable models, first, we implemented state-of-the-art non-interpretable models, which allow us to define a baseline accuracy, that our interpretable alternatives will try to match. The chosen models to have a baseline accuracy for our set were: ARIMA, SARIMA, Prophet, two Recurrent Neural Network (RNN) models (Long Short-Term Memory (LSTM) and GRU), and Transformer.

All the non interpretable models were implemented using the Python package "*darts*".

4.2.1 ARIMA

The first non-interpretable model we decided to implement was the Autoregressive Integrated Moving Average (ARIMA). This model can be described as a linear regression model that makes its predictions based only on the past values of the target variable. This model can be divided into 3 components, the Autoregressive (AR), the Integrated (I), and the Moving Average (MA) component [40].

The first component of this model, the Autoregressive, predicts the target value based on its own previous values, this is, it uses lagged values of the target as the X variables. This component can be seen in the equation (4.1), where Y is the prediction and is simply a linear function of its past (lagged) n values (n being the number of values we want to consider, a parameter we choose). Each of its past values is multiplied by a regression beta (B_0, B_1,B_2,B_n) that we fit when training the model.

$$Y_{\text{forward 1}} = B_0 + B_1 Y + B_2 Y\_lag_1 + B_3 Y\_lag_2 + ... + B_n Y\_lag_{n-1} \tag{4.1}$$

The second component of the ARIMA is the Integrated component, which replaces the data values with the difference between the data values and the previous values. This can be seen in the equation (4.2), which is the same as the equation (4.1), but instead of having the value Y, we have the difference between Y, and the previous value, Y_lag_1. The Integrated component is also controlled by a parameter we can choose by deciding the order of differentiation, i.e., the number of times the data have had past values subtracted.

$$Y_{\text{forward }1} - Y = B_0 + B_1(Y - Y\_lag_1) + B_2(Y\_lag_1 - Y\_lag_2) + ...B_n(Y\_lag_{n-1} - Y\_lag_n) \quad (4.2)$$

The third and last component of this model is the Moving Average (MA) which can be described with the equation (4.3). The only difference from this equation to the AR equation ((4.1)) is that instead of using the past values we are using the error (E) of the past values. Just like in the AR equation, we decide how many values (n) we want to consider and can choose that parameter.

$$Y = B_0 + B_1 E\_lag_1 + B_2 E\_lag_2 + ... + B_n E\_lag_n \quad (4.3)$$

To find the best value for each of the parameters, this is, the best value for the parameter of the Autoregressive component (p), the parameter of the Integrated component (d) and the parameter of the Moving Average component (q) and consequently find the best model, we investigated the Akaike Information Criterion (AIC) [41] and the Bayesian Information Criterion (BIC) values to compare the different models.

The AIC is an estimator of prediction error founded on information theory. This criterion estimates the relative quality of statistical models based on the relative amount of information that a given model loses to represent the process that generated the data. Consequently, the higher the AIC value, the higher the amount of information the model loses, thus meaning, that the best model will be the one with the lowest AIC value since it is the model with the lowest loss of information. Moreover, since AIC deals with the trade-off between how well the model fits and its simplicity, it is expected that the models with lower AIC scores will balance better the ability to fit the data set while preventing over-fitting. The Akaike Information Criterion (AIC) can be defined by the equation (4.4), where k is the number of parameters of the model and $\hat{L}$ is the maximum value of the likelihood function for the model.

$$AIC = -2\log(\hat{L}) + 2k \quad (4.4)$$

BIC is another criterion we can use to find the best model. Just like the Akaike Information Criterion, the lower the BIC value, the better the model, as it is also based on the likelihood function and also penalizes models with more parameters like AIC. The main and only difference between BIC and AIC is that the second term, the penalty term, in BIC depends on the sample size [42] and will consequently be larger. BIC can

be defined by Equation 4.5, where k is the number of parameters of the model, n is the number of data points and $\hat{L}$ is the maximized value of the likelihood function.

$$BIC = -2\log(\hat{L}) + k\log(n) \tag{4.5}$$

After carrying out a grid search and comparing the AIC and BIC values of the different models, we chose the model with the lowest AIC value for each of the datasets. The parameters for the best model for each of the datasets can be seen in Table 4.3. The results show us that the four datasets share the same best parameter on the Integrated component, 1, but the parameter for the Moving Average is different in all of them.

Emergency Level	1	2	3	4
(p,d,q)	(1,1,2)	(4,1,7)	(1,1,9)	(1,1,3)

Table 4.3: Parameters for the best ARIMA model for each dataset.

4.2.2 SARIMA

SARIMA stands for Seasonal-ARIMA (Seasonal Autoregressive Integrated Moving Average). This model is an extension of the ARIMA and is formed by adding seasonal terms to the ARIMA models [43]. These terms are the (P, D, Q)$_s$, where P and Q are similar to p and q in the ARIMA model, but now refer to the seasonal orders, this is, P stands for the seasonal autoregressive order and Q stands for seasonal moving average order. Regarding D, it represents the seasonal difference order. Lastly, s represents the length of the season.

Combining these new seasonal terms with the terms of the ARIMA, the SARIMA model is defined as (p, d, q)(P, D, Q)$_s$.

Similarly to what we did to find the best ARIMA model, we performed a grid search where we compared the AIC and BIC values of different models, where we changed the p, d, q, P, D, Q, and s values, and chose the one with the lowest AIC value for each of the datasets. The results can be seen in the table 4.4 and show us that all the SARIMA models share one parameter, the length of the season, with a value of 4.

Emergency Level	1	2	3	4
(p,d,q)(P,D,Q)$_s$	$(1,1,2)(0,1,1)_4$	$(1,1,1)(1,1,2)_4$	$(3,0,3)(0,1,2)_4$	$(1,0,2)(0,1,1)_4$

Table 4.4: Parameters for the best SARIMA model for each dataset.

4.2.3 Prophet

Prophet [44] is an open-source software available both in Python and R to generate time series models. It was designed to have intuitive parameters that can be fine-tuned without knowing about the details of the underlying model. Prophet overcomes some drawbacks of ARIMA, such as the need of having regularly spaced measurements and the need of

inserting missing values in the data. Moreover, ARIMA fails to capture any seasonality and if there is a trend change near the cutoff period its forecasts are susceptible to large trend errors. Prophet can be described as the sum of three main components, the trend function, $g(t)$, which models the non-periodic changes in the value of time series, and the seasonality function, $s(t)$, which represent cyclic changes and the holidays' function, $h(t)$, that represent the effects of holidays which occur on potentially irregular schedules, plus an error term, ϵ_t, that represent idiosyncratic changes. This formula can be seen in the equation 4.6.

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{4.6}$$

Prophet has two different models implemented to capture the overall trend of the data, a saturation growth model, and a piece-wise linear model. The first one, the saturation growth model, is derived from the logistic growth equation where the carrying capacity is replaced by a function of time, and the growth rate and offset change their values at each changepoint. This model should be used when there is a limit in the time series and can be seen in the equation 4.7, where k is the base rate, δ is a vector of rate adjustments, where δ_j is the change in rate that occurs at time s_j, m is the offset, γ is a vector of rate adjustments for the offset, where γ_j is the change that occurs at time s_j and $a(t)$ is a vector that can be represented as

$$a_j(t) = \begin{cases} 1, \text{if } t > s_j, \\ 0, \text{ otherwise} \end{cases}$$

$$g(t) = \frac{C(t)}{1 + exp(-(k + a(t)^T \delta)(t - (m + a(t)^T \gamma))} \tag{4.7}$$

The second growth model implemented in Prophet is the piece-wise linear model and is used when there is no saturation growth. This trend equation can be seen in 4.8.

$$g(t) = (k + a(t)^T \delta)t + (m + a(t)^T \gamma) \tag{4.8}$$

The changepoints s_j mentioned above are moments of time where there is a change that impacts the behavior of the data. These moments can either be manually defined or automatically specified given a set of candidates.

Prophets seasonality function is a Fourier Series as a function of time, thus, it is a sum of successive sines and cosines that can approximate the seasonal (cyclic) values of the data. This seasonality can either be additive or multiplicative. In case of the additive seasonality, the effect of seasonality is added to the trend while in the case of multiplicative seasonality it is multiplied, which may make it either grow or decrease.

The last component of Prophet is the holidays' function which allows us to provide a list of dates that may affect the predictions so it can adjust the forecast. Prophet will then use these dates to add or subtract value from the forecast from both the growth and seasonality functions based on the past influence of these dates on the values.

Finding the best parameters to get the best Prophet model was easier than finding the best ARIMA model. The first thing that we chose was to use the linear growth function since we do not have a limit for the time a person can wait in the emergency department of a hospital. To choose which seasonalities we should use, we tried different models using different combinations of the daily, weekly, and monthly seasonalities and compared the different results. The best results were achieved using the daily and weekly seasonalities. While trying the different seasonalities we also tried using both the additive and multiplicative seasonality. Once again, we chose the one that achieved the best results which was the additive seasonality. Lastly, we inserted the Portuguese holidays which improved the results of the model.

4.2.4 RNNs

Recurrent Neural Networks (RNNs) [45] are an important part of neural networks research, which have become more popular due to their potential to solve problems that involve time sequences of events or ordered data like words in a sentence. These networks can use their connections to store a representation of the recent input events in form of activations, which can be described as having a "memory", more specifically a short-term memory since it only stores the representation of recent events. Recurrent Neural Networks view the input as a sequence of events and when making a prediction, they consider the information from past entries. The input of the network can be seen as a sequence of vectors where each layer of the network will receive part of this input. Besides the part of the input that each layer gets, it also receives some information from the previous layer. This information from the previous layer encodes aspects of the sequence received so far, which can be very useful for problems like time series analysis or natural language processing.

Recurrent Neural Networks have the problem of having a short memory, this is, they have trouble relating events that are too spread apart in time. Current algorithms like Back-Propagation Through Time (BPTT) or Real-Time Recurrent Learning (RTRL), either take too long to compute the activations over long time lags or simply do not work at all. This happens because of the way error signals flow backward in time, they either blow up, which leads to oscillating weights, or vanish, which makes the network take too long to learn long time lags.

4.2.4.1 LSTM

To overcome the short-memory problem and not lose relevant information from events further away in the past while keeping the short time lag capabilities, Sepp Hochreiter and Jürgen Schmidhuber [46] proposed a novel recurrent network architecture, the "Long Short-Term Memory" (LSTM). This network introduced two additional features to the RNN architecture, an input gate unit and an output gate unit. Current Long Short-Term Memory networks also have a third gate, the "forget gate" [47], which prevents the blowing

or vanishing error problem. When Sepp Hochreiter and Jürgen Schmidhuber introduced the LSTM, instead of this third gate, they introduced the Constant Error Carousel (CEC) concept. It was the central feature of LSTM, which as we will see further is the same as having the forget gate with the value 1.0. The CEC is obtained when the activation function used is the identity function and the recurrent connection has a constant weight of 1.

Initially, Sepp Hochreiter and Jürgen Schmidhuber tried using only the CEC before adding any other feature to the RNN, but it did not work well except in simple problems that involved local input/output representations and non-repeating input patterns. To solve this problem, Sepp Hochreiter and Jürgen Schmidhuber [46] extended the CEC and added additional features to allow constant error flow even through special and self-connected units without the problems of the naive approach. The first feature introduced was a multiplicative input gate unit that protects the memory content stored in a specific unit, j from perturbations made by irrelevant inputs. In the same way, they introduced a multiplicative output gate that protected other units from perturbations of memory contents that are currently irrelevant but are stored in j.

The result of these additions is a more complex unit called memory cell. The j-th memory cell is defined as c_j. The center of each memory cell around which the cell is built is a linear central unit with a fixed self-connection, the CEC. Along with the input that the memory cell already received, net_{c_j}, now it also gets input from a multiplicative output gate, out_j, the "output gate", and from a multiplicative input gate, in_j, the "input gate". If we denote the activation at time t of the input gate as $y^{in_j}(t)$ and the activation at time t of the output gate as $y^{out_j}(t)$ we have:

$$y^{out_j} = f_{out_j}(net_{out_j}(t)) \tag{4.9}$$

and

$$y^{in_j} = f_{in_j}(net_{in_j}(t)). \tag{4.10}$$

where

$$net_{out_j}(t) = \sum_u w_{out_{j_u}} y^u(t-1) \tag{4.11}$$

and

$$net_{in_j}(t) = \sum_u w_{in_{j_u}} y^u(t-1). \tag{4.12}$$

The indices u present in the summations can stand for different types of inputs like input units, gate units, memory cells, or conventional hidden units. All these input units might communicate useful information regarding the current state of the network. For example, the input gate may use the inputs from other memory cells to determine where or not they should store (access) certain information in the memory cell.

These gate units allow the memory cell c_j to avoid input and output weight conflicts. To avoid the input weight conflicts, in_j controls the error flow from the input connections, $w_{c_j i}$, thus, it can decide when to keep or ignore information in memory cell c_j and to circumvent output weight conflicts, out_j controls the error flow from unit j's output connections, by deciding when to prevent other units from being perturbed by c_j. This architecture of the memory cell created by Sepp Hochreiter and Jürgen Schmidhuber [46] can be seen in Figure 4.2.

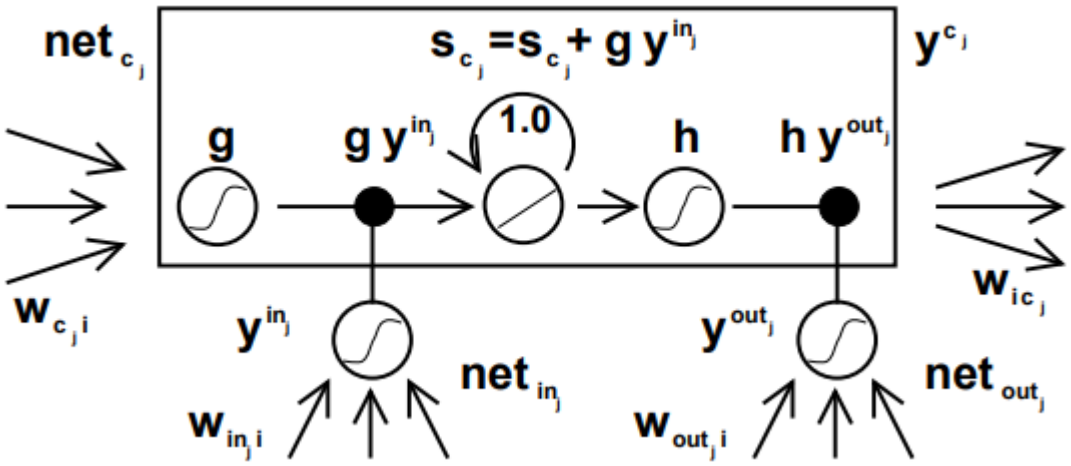


Figure 4.2: Architecture of memory cell c_j (the box) and its gate units in_j, out_j. The self-recurrent connection (with a weight of 1.0) indicates feedback with a delay of one time step. It builds the basis of the "constant error carousel" (CEC). The gate units open and close access to CEC [46].

The memory cell c_j allows the information to flow through the sequence chain, so it can be seen as the memory of the network. Since it can carry relevant information from the beginning of the sequence throughout the processing of the sequence, it can reduce the effects of the short-term memory that affect Recurrent Neural Networks. Nevertheless, we must be aware that the cell may receive irrelevant information that it should remove, and this is where the gates step in, they regulate which information is kept or removed throughout the sequence. The gates are neural networks that decide which information is allowed to get in and out of the memory cell, thus, during training, they learn what information is relevant to retain in the memory cell and what information is not relevant and thus should be removed. Every gate contains a *sigmoid* activation, which squishes the values between 0 and 1. This helps to update or forget data because if the value is 0, any number multiplied by 0 will be 0 and thus will disappear (be forgotten). On the other hand, any number multiplied by 1 will remain the same value as before, thus, the value will be kept in memory.

To update the memory cell state, the input gate receives the previous hidden state and the current input of the sequence and passes them into the *sigmoid* function. This will decide if the values are important or not and thus, if they will be updated or not, by

squishing the values between 0 and 1, where 0 means it is not important and 1 means it is important. The hidden state and the current input are also fed into a *tanh* function, which ensures that the values will stay between -1 and 1, thus, regulating the network by avoiding some values to explode and becoming so high that others would seem insignificant. The result from the *tanh* function will then be multiplied by the *sigmoid* output that decides which information is important to keep. The result of this multiplication will be added to the result of the forget gate [48], which was introduced by Gers et al. [47]. This new architecture can be seen in Figure 4.3.

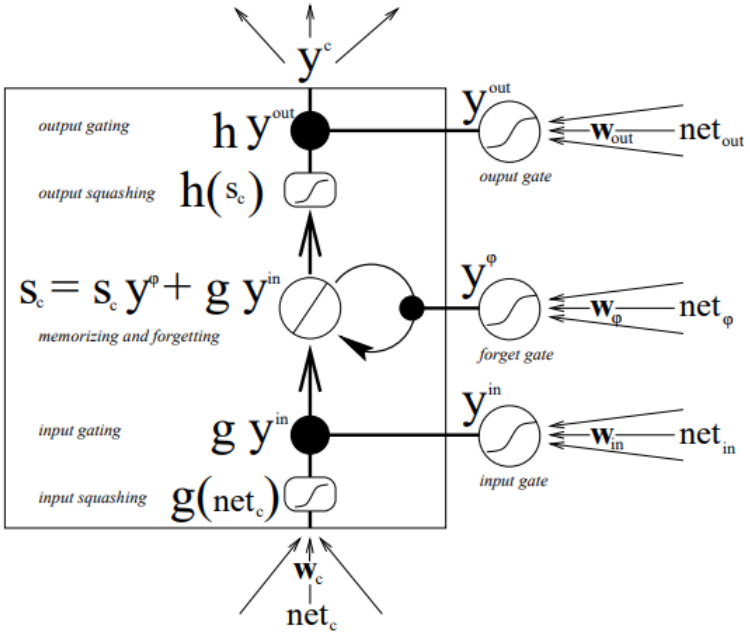


Figure 4.3: Memory block with only one cell for the extended LSTM. A multiplicative forget gate can reset the cells inner state s_c [47].

The forget gate, which Gers et al. [47] introduced, decides what information is kept and what information is removed. The standard LSTM tended to grow linearly during the presentation of a time series, the nonlinear characteristics of the sequence processing were made only by the squashing functions and the input and output gates. The forget gates solve this problem by resetting the memory blocks when the contents are out of date and hence useless. These resets do not mean to immediately reset to zero, but instead, gradually resetting, corresponding to slowly fading cell states. To make this happen, the forget gate receives the previous hidden state of the current input and passes them through the *sigmoid* function. Just like in the input gate, the values will be comprised between 0 and 1 where the closer to 0 the more it will forget, and the closer to 1 the more information it will keep. If we designate the activation at time t of the forget gate as $y^{\varphi_j}(t)$, similarly to the input and output gates activations, we have

$$y^{\varphi_j} = f_{\varphi_j}(net_{\varphi_j}(t)), \tag{4.13}$$

where

$$net_{\varphi_j}(t) = \sum_u w_{\varphi_{ju}} y^u(t-1). \tag{4.14}$$

Lastly, there is the output gate which will decide what should be the next hidden state. This gate starts by receiving the previous hidden state and the current input and passes them into a *sigmoid*. After that, the updated memory cell (which was already updated by combining the outputs from the input and forget gates) is passed to a *tanh* function. The result values from the *sigmoid* and *tanh* functions are multiplied to decide the final hidden state output, this is, what information it should carry. The new memory cell state and new hidden state are then carried over to the next time step.

The main limitations of LSTM consist of the fact that it requires additional units, the gates, in each memory cell (compared to standard recurrent networks) and they are prone to overfitting. On the other hand, the main advantages of LSTM consist in the error backpropagation, which allows them to bridge very distant time lags. Moreover, they have a good ability to handle noise and work well over a large range of parameters, such as learning rate, input gate bias, and output gate bias.

To find the best LSTM model for each of the datasets, we did a grid search for each one of them where we experimented with different parameters, trying different values for the learning rate, the size of features maps in the hidden state, dropout value (the fraction of neurons that are dropped in all except the last layer) and different loss functions.

4.2.4.2 GRU

The Gated Recurrent Unit (GRU) was a new architecture proposed by [49] to make the recurrent units able to adaptively capture dependencies of different time scales.

Just like the LSTM, the GRU also has gating units that regulate the flow of information inside the unit. The main difference between LSTM and GRU is that in this new architecture a single gating unit controls both the forgetting factor and the decision to update the state unit [50], thus, having only two gates, a reset gate, and an update gate. Moreover, the GRU uses the hidden state to transfer the information inside the unit without using a separate memory cell [51].

The activation h_t^j at time t of the GRU is a linear interpolation between the previous activation h_{t-1}^j and the candidate activation $\tilde{h}_t^j$:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{4.15}$$

where z_t^j, the update gate, decides how much the unit updates its activation and is computed by:

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j, \tag{4.16}$$

where x_t is the current input, W_z is the network unit weight, h_{t-1} holds the information of the previous $t-1$ units and U_z is its own weight. This update gate acts similar to the forget and input gates in the LSTM and determines how much information from the past should be kept and what new information to add.

The candidate activation $\tilde{h}_t^j$ is computed by:

$$\tilde{h}_t^j = \tanh(W x_t + U(r_t \odot h_{t-1}))^j, \tag{4.17}$$

where $\odot$ represents an element-wise multiplication and r_t the reset gate, which will decide how much of the previous information to forget. If r_t^j is close to 0, the reset gate will make the unit behave like it is reading the first input of an input sequence, making it ignore the previously computed state.

The reset gate r_t^j is calculated using a formula similar to the one used for the update gate:

$$r_t^j = \sigma(W_r x_t + U_r h_{t-1})^j \tag{4.18}$$

A single Gated Recurrent Unit is illustrated in Figure 4.4.

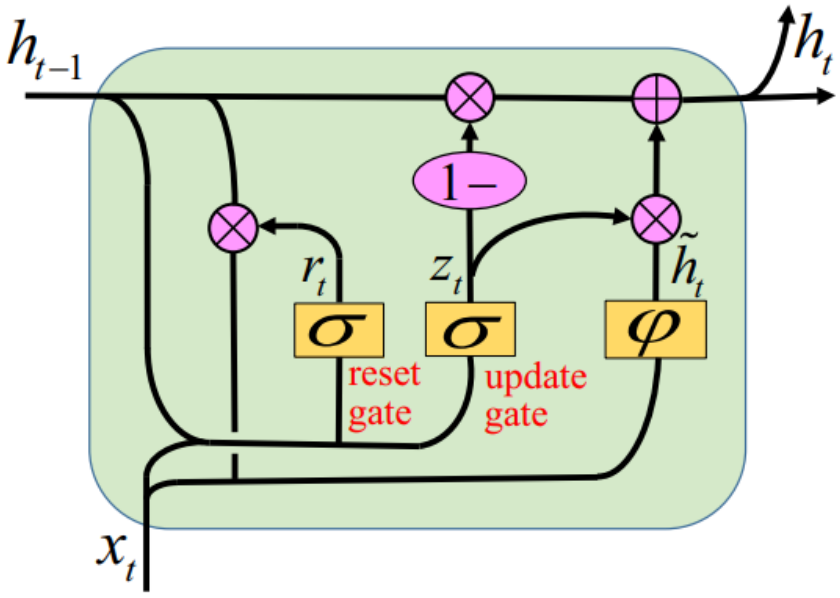


Figure 4.4: A diagram of a GRU block. The reset gate r_t controls whether ignoring the previous hidden state h_{t-1} or not. The update gate z_t decides whether the hidden state h_t is to be updated with a new hidden state $\tilde{h}_t$ [52].

To find the best GRU model for each of the datasets, we did a grid search just like we did for LSTM, where we tried different parameters in each of the models. The different

parameters tested here were similar to the ones tested in the LSTM, this is, we tested different learning rates, the number of recurrent layers, the number of features in the hidden state, the dropout value, and the loss function.

4.2.5 Transformer

The Transformer is a neural network model that uses an attention mechanism to identify global dependencies between input and output, thus, avoiding recurrence [53]. Attention can be described as a function that maps a query and a set of key-value pairs to an output, where the query, values, keys, and output are all vectors. The output results from a weighted sum of the values, where the weight given to each value is the result of a compatibility function of the query with the corresponding key. Moreover, this model uses an architecture where an encoder maps a sequence of symbol representations (input) to a sequence of continuous representations. Later, a decoder generates an output of one element at a time that result in a sequence of symbols. At every step, the model consumes the previously generated symbol as additional input to generate the next one.

This architecture uses stacked self-attention and point-wise, fully connected layers, both for the encoder and the decoder. The model architecture can be seen in the figure 4.5, where on the left we have the encoder and on the right the decoder.

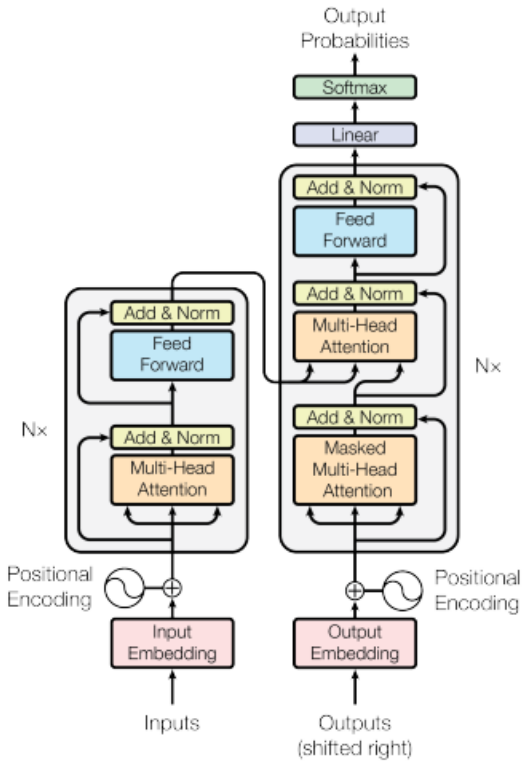


Figure 4.5: Architecture of the Transformer model [53].

The encoder is simply a stack of multiple identical layers. Each layer has two sub-layers, where the first is a multi-head self-attention mechanism, while the second is simply a position-wise fully connected feed-forward network. Around each sub-layer, they employ a residual connection followed by layer normalization. This means that the output of each sub-layer is the normalization of the sum of x and Sublayer(x), where Sublayer(x) is the function implemented by the sub-layer itself. All sub-layers in the model produce an output of the same dimension to facilitate these residual connections.

Just like the encoder, the decoder is composed of a stack of identical layers, but instead of having two sub-layers, it adds a third sub-layer, which will perform multi-head attention over the output of the encoder stack. Like in the encoder, the decoder also employs residual connections around each sub-layer that are followed by a layer normalization. To guarantee that the predictions for the position i can depend only on previously known outputs, the sub-layers in the decoder stack are modified to prevent positions from attending to subsequent positions and the output embeddings are offset by one position.

The authors of this model use a particular attention, which they call "Scaled Dot-Product Attention", which can be seen in Figure 4.6. The input in this attention consists of queries and keys that have a dimension d_k and values that have dimensions d_v. To obtain the weights on the values, they compute the dot products of the query with all the keys and divide each by $\sqrt{d_k}$ before applying a softmax function and getting the final weights. This function can be seen in 4.19, where Q represents a matrix of the queries packed together, K represents the matrix of the keys packed together and V represents the matrix of the values packed together.

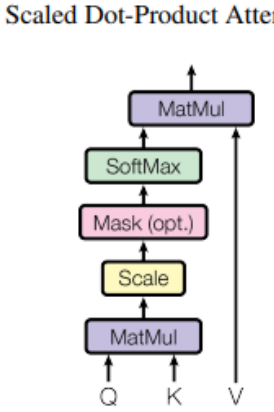


Figure 4.6: Scaled Dot-Product Attention [53].

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^t}{\sqrt{d_k}})V \tag{4.19}$$

To allow the model to jointly attend to information from different representation subspaces at different positions, the authors used multi-head attention. The multi-head attention instead of performing a single attention function with d_m-dimensional keys,

values, and queries, linearly projects the queries, keys, and values with different, learned linear projections h times to d_k, d_k and d_v dimensions, respectively. Thus, the computation of the attention function can be performed in parallel on each of these projected versions of the queries, keys, and values, resulting in d_v-dimensional output values. The model then concatenates them and projects them to produce the final values as can be seen in Figure 4.7.

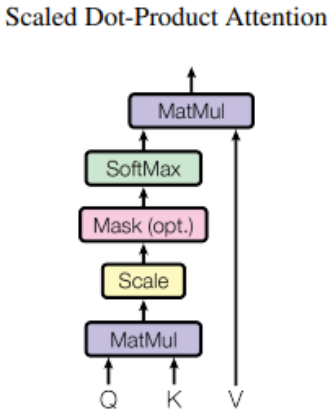


Figure 4.7: Multi-head attention consists of several attention layers running in parallel [53].

The Multi-head attention function can be seen in equation 4.20 where W_i^Q, W_i^K, W_i^V are parameter matrices of the projections and $W^O \in \mathbb{R}^{hd_v \times d_m}$.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_h)W^O \tag{4.20}$$

Where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{4.21}$$

The multi-head attention is used in three different ways in the Transformer. The first one is by the encoder, which contains self-attention layers, where all of the keys, values, and queries come from the previous layer in the encoder. Thus, each position in the encoder can attend to all positions in the previous layer. The second way is in the decoder, that just like in the encoder, has self-attention layers so each position in the decoder can attend to all positions in the decoder up to that position (including it). To preserve the auto-regressive property, they had to prevent leftward information flow in the decoder. In order to do it, they implemented scaled dot-product attention by masking out, this is, setting to -∞ every value in the input of the softmax which corresponds to an illegal connection. The third and last way the multi-head attention is used is in the "encoder-decoder attention" layer, where the queries come from the decoder layer and the memory keys and values come from the output of the encoder. This way, every position in the decoder can attend all positions in the input sequence.

Besides the attention sub-layers, each layer, both in the encoder and in the decoder, contain a fully connected feed-forward network applied to each position individually and exactly in the same way. This consists of two linear transformations and a ReLU activation. The linear transformations are the same for every position but with different parameters for each layer.

They used learned embeddings to convert the input and output tokens to vectors of dimension d_{model}. They also use a linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. The Transformer shares the same weight matrix between the two embedding layers and the pre-softmax linear transformation. Moreover, in the embedding layers, those weights are multiplied by $\sqrt{d_{model}}$. In order to find the best Transformer model for each dataset, we did a grid search just like we did for LSTM and GRU where we tried different parameters in each one of them. The different parameters tested in the different Transformer models were the number of heads in the multi-head attention sub-layer, the number of encoder layers in the encoder, the number of decoder layers in the decoder, and the dimension of the feedforward network models.

4.3 Interpretable models

4.3.1 RuleFit

As we saw in Chapter 2.1.1, Rulefit uses a powerful learning method which is the learning ensembles to make an interpretable algorithm. Rulefit uses this method by generating a regression where the variables are rules created by a tree ensemble algorithm. This tree ensemble algorithm can be any of our choice [27].

RuleFit was implemented using two different methods. The first one was using the Python package "*rulefit*" which utilizes Gradient Boosting as tree ensemble algorithm. The second method was a manual implementation of RuleFit. In order to manually implement RuleFit, the tree algorithm, the extraction of rules, and the LASSO regression were individually implemented and then combined, resulting in RuleFit. Since the "*rulefit*" package utilizes Gradient Boosting, we decided to follow the same approach and also use Gradient Boosting as the tree ensemble algorithm.

The main goal of Gradient Boosting is to convert weak learners into strong learners by training each new model concerning the error of the whole ensemble learnt so far, by training it on a modified version of the original dataset. Gradient Boosting is an algorithm that consecutively fits new models to achieve more accurate estimates of the response variables. The main idea is to build new models that are maximally correlated with the negative gradient of the loss function, associated with the whole ensemble [54]. We used the squared error loss, whose equation can be seen in (4.22), as loss function, which results in a consecutive error-fitting learning procedure, although, the loss function applied in a Gradient Boosting algorithm can be arbitrary.

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2. \tag{4.22}$$

The algorithm starts by training a decision tree where each observation is assigned an equal weight. After training the first decision tree, the algorithm evaluates it and increases the weights of the observations that are harder to classify and lowers the weights for the ones that are easier to classify. The second decision tree is grown on this weighted data, where the goal is to improve upon the predictions of the first tree. Now that the model has two decision trees, it computes the classification error from this 2-tree ensemble and builds a third tree to predict the revised residuals. This process is repeated in an additive manner for a specified number of iterations. The subsequent trees help to classify observations that are not well classified by previous trees. The final predictions are weighted sums of the predictions made by the previous models.

Whenever the algorithm grows a new tree, the algorithm updates [55], thus is, fits the new tree to the ensemble, using the formula (4.23) where $\{R_{jm}\}_1^J$ are the disjoint regions created by each terminal node (leaf) of the tree at the m-th step, b_{jm} are the coefficients and ρ_m is a value chosen to minimize a specified loss function.

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x) \tag{4.23}$$

The value ρ_m is calculated using the equation 4.24, where $h(x_i)$ is a decision tree.

$$\rho_m = \underset{\rho}{\text{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i)) \tag{4.24}$$

The decision tree at the m-th step, $h_m(x)$ can be written as

$$h_m(x) = \sum_{j=1}^{J} b_{jm} 1(x \in R_j). \tag{4.25}$$

In the first implementation of the RuleFit, this is, the implementation using the Python package "*rulefit*", no tuning was necessary. For the manual implementation, we tuned the Gradient Boosting on the learning rate, maximum number of trees, and the maximum depth of each tree.

4.3.2 SIRUS

The Stable and Interpretable Rule Set (SIRUS) is an algorithm that follows some principles from RuleFit as seen in Chapter 2.1.2. Just like RuleFit, it extracts rules from trees and uses them in a regression to make predictions. The principal differences between RuleFit and SIRUS are that instead of using a Gradient Boosting algorithm to generate the trees SIRUS uses a Random Forest as the tree ensemble algorithm and instead of using the

LASSO regression it uses the Ridge regression solution, which allows it to have better stability according to the authors.

SIRUS was implemented using the R package "*sirus*". The parameter p_0 that represents the threshold to select the relevant rules was tuned through cross-validation by using the function "*sirus.cv*". Moreover, we performed a study to see how changing it would affect both the accuracy and sparsity of the model to see if we could find a better accuracy with a few more rules than the obtained through cross-validation, but still maintain that number low as we will see further. The number of trees created by SIRUS was automatically defined, as the package creates the necessary number of trees until it achieves a stability of 95%. Lastly, the maximum depth of the trees was left at 2, which is the default.

4.4 Proposed New Interpretable models

Besides the existing methods studied above, we propose four new Interpretable models in this thesis.

4.4.1 REN: RuleFit with ElasticNet

The first method we propose follows the RuleFit architecture, this is, it starts with a tree ensemble algorithm, followed by rule extraction from the generated trees, and finishes with a regression model. We use the same tree ensemble algorithm as in RuleFit, the Gradient Boosting, the difference from this model to the RuleFit is the fact that instead of using the LASSO regression, we use the ElasticNet regression. With this change, we expect to overcome some limitations of the LASSO regression. First, we expect to improve the performance when there is a large dimensional data with few examples [56], and secondly, it should provide a strong feature correlation since when there are multiple features correlated with one another, Lasso picks one of them at random without caring about which one it chooses while ElasticNet is likely to pick both.

The ElasticNet combines the penalty from LASSO, which is the $L1$ penalty with the penalty from Ridge regression, which is, the $L2$ penalty. Using ElasticNet the coefficients for the rules are obtained using the formula (4.26), where y is the observed value, X represents all the features, this is, both the original features and the rules extracted, λ_1 represent the regularization parameter for the LASSO regularizer, and λ_2 represents the regularization parameter for the Ridge regularizer.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|\beta\|_1 \tag{4.26}$$

To implement this model we started by using the tree algorithm and the rule extraction methods we developed when implementing the different parts of RuleFit manually. After that, we implemented the ElasticNet regression and fed it the extracted rules along with the data.

4.4.2 Interpretable Rules by DL-Learner

Our three other proposed new Interpretable methods start with a tree ensemble algorithm just like RuleFit and SIRUS, followed by a rule selection made by the DL-Learner software, and finish with a regression model that assigns a weight to each rule so we can ensemble them. The tree ensemble algorithm we chose was the Gradient Boosting just like in RuleFit (4.3.1). We start by extracting 2000 rules from the trees created by Gradient Boosting. Following this extraction we clean the redundant rules, thus, if there is any rule that can be simplified, we simplify it. This can be seen in the following example:

The rule

$$\text{Month} <= 7.5 \ \& \ \text{Day} > 13.5 \ \& \ \text{Day} > 12.5$$

Can be simplified to the rule

$$\text{Month} <= 7.5 \ \& \ \text{Day} > 13.5$$

And maintain the exact same meaning, since if the number of the day is bigger than 13, it will logically be bigger than 12.

After cleaning the non-significant conditions of each rule, we start to prepare the data for DL-Learner.

The DL-Learner is a software that provides a based machine learning tool to solve supervised learning problems and helps to construct knowledge and learn about the data [57]. It is biased toward short and human-readable definitions, which is exactly our goal in this thesis, to obtain the shortest and most human-readable rules we can. The DL-Learner application is written in Java.

DL-Learner uses a component-based model which has four types of components that can be seen in figure 4.8. The four types of components are the knowledge source, which integrates the background knowledge, the reasoning service, which provides connections to an existing reasoner, the learning problem, which specifies the problem type to be solved by an algorithm and a learning algorithm, which provides methods to solve one or more specified learning problem types. Each of these types has various implemented components. These different components can have their own configuration options, which can be used to change the parameters of a component.

We take advantage of the DL-Learner software to help us understand which are the most important and relevant rules from the 2000 rules extracted from the Gradient Boosting Trees.

In order to do it, we create two files that DL-Learner needs to run, which are the configuration and the background knowledge files. The configuration file is where we define the type of problem, the algorithm that we want to use, the reasoner component, and the name of the knowledge source.

The learning problem we are using is the Positive and Negative Examples, where the goal is to find a class expression (a rule in our case) such that all (or many) positive

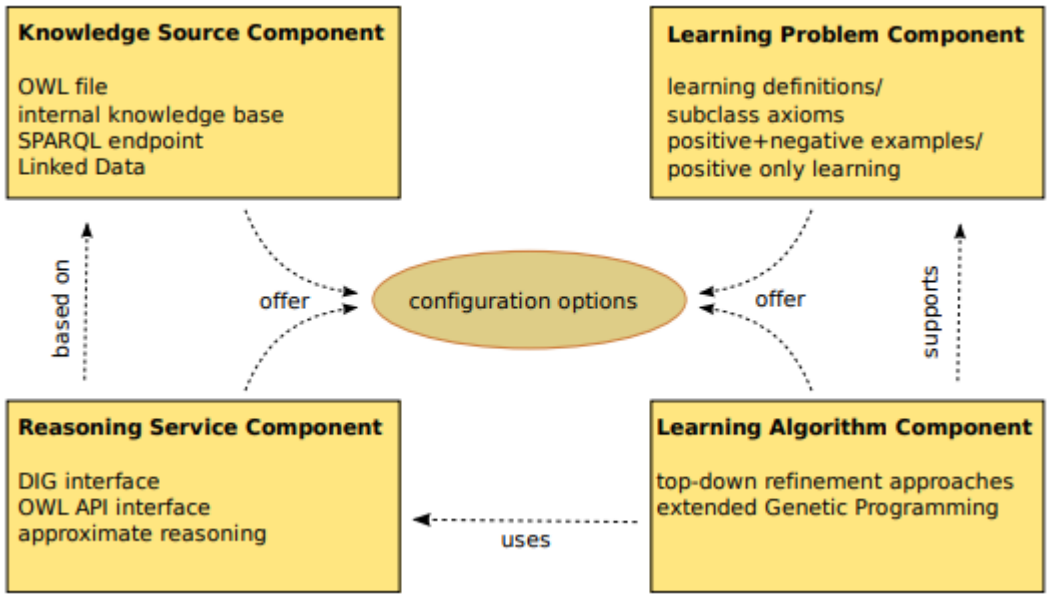


Figure 4.8: The DL-Learner architecture is based on four component types. Each of these component types can have its configuration options. A component manager can be used to create, combine and configure components [57].

examples are instances of that class, this is, respect the rule, and none (or few) negative examples are instances of that class, this is, do not respect the rule. The goal is to find a class expression (a rule) that generalizes to unseen individuals and is readable. Since we are using the Positive and Negative Examples, in the configuration file that we give to DL-Learner, besides defining the different components, we also identify which are the positive and which are the negative examples by giving the id of each instance.

The algorithm we use is the Class Expression Learning for Ontology Engineering (CELOE), which is the best class learning algorithm available within DL-Learner [57]. This algorithm follows the "generate and test" approach, which means that learning is seen as a search process where it generates several class expressions and tests them against a background knowledge base [58]. Each class expression is then evaluated using a heuristic which measures how well a given class expression fits the learning problem and how it is used to guide the search in the learning problem. Furthermore, CELOE guarantees that the returned class expressions, this is, the rules, are minimal in the sense that we cannot remove any part of them (any condition) without getting a different expression.

The reasoner component we use is the Closed World Reasoner, which tests whether an individual is an instance of a class or not.

The knowledge source used is a file we create where we have each instance (each data point) class expressions, which correspond to all the conditions from the 2000 rules generated by the Gradient Boosting Trees that the datapoint respects. An example of part of this file can be seen in image 4.9, where "sample" indicates the sample id and "af", "ai", "ak", "an" and "ao" represent conditions of rules, for example "af" represents "Day > 6.5".

Changing the condition into a combination of letters was necessary since DL-Learner did not accept the knowledge base if there were numbers, spaces, or operators like ">=".

```
sample(sample_6).
af(sample_6).
ai(sample_6).
ak(sample_6).
an(sample_6).
ao(sample_6).
```

Figure 4.9: Sample of part of a knowledge base file.

In order to create the file that will be used as knowledge source, we first create a new dataset. To create that dataset, we start by creating a new column named "target" in our original dataset. This column will have values between 1 and 10 (this interval could be another interval chosen). For each target value, the DL-Learner software will give us the best rules to separate the positive from the negative examples, thus, it will try to find the best rule that identifies data points with that target value.

After adding the column "target" to our dataset, we shrink our values from the column "Waiting_Time", which is the column that has the values we are trying to predict, to have values between 1 and 10, and fill the column "target" with those values. After having the column "target" filled, we add each condition of our rules as a column to our dataset. For example the rule "Month <= 7.5 & Day > 13.5" would give birth to two new columns in the dataset, the column "Month <=7.5" and the column "Day > 13.5". The value in each row is either 1 or 0, depending if the data point satisfies that condition (and thus has the value 1) or not (and thus has the value 0). To finish the dataset that helps us create the knowledge source file for DL-Learner, we drop the initial columns from the dataset, thus, leaving the dataset only with the target, the index, and the conditions columns.

With the new dataset created, for each target value, we create 10 configuration and 10 knowledge base files instead of 1 so DL-Learner has different combinations of positive and negative examples, thus, different data points that have a target value of 1 (positive examples) and data points that have a target value of 0 (negative examples), to learn from different combinations. These files are necessary so DL-Learner can learn from positive and negative examples, thus, learning which conditions are most relevant to identify data points that have that target value. DL-Learner then returns the top 10 rules (either a single condition or a combination of conditions) it found to separate positive and negative examples of this target value, this is, the best rules to identify data points that have that target value. Since DL-Learner returns us the top 10 rules for each of the 10 targets, we get 100 rules, 10 for each target value, but we only extract the one that has the best accuracy and shortest number of conditions for each target value, this is, if there is more than 1 rule with the best accuracy, we only extract the one that has the shortest number of conditions, which leave us with 1 rule for each target value.

After running the DL-Learner for the 10 target values, we get 10 rules. Following that,

we convert these rules, which are in text files, into the same format as the rules that we get from the Gradient Boosting Trees. To do so, we first put each rule in the disjunctive normal form and then separate the literals of the rules that have 1 or more disjunctions into different rules, so we get simplified rules with no disjunctions (just like the ones we get from the trees). This may result in ending up with more than 10 rules.

Since we always prepare for the worst-case scenario, having 10 rules would not be enough for us, the regression model would not have many rules to choose from. So, to have more and varied rules, we repeat the previous step 5 times, this is, we create 5 new datasets to create different knowledge base and configuration files for DL-Learner instead of just 1 and run it 5 times. This results in having a minimum of 50 rules, which was already better for us.

A new problem arose when we checked the rules from running DL-Learner 5 times. We noticed that many of the rules were the same as can be seen in image 4.10. In order to try to obtain more varied and different rules from DL-Learner, in addition to feeding it files relative to the entire dataset 5 times, we fed it files relative to only parts of the original dataset. To feed DL-Learner files that were relative only to parts of the original dataset, we created smaller datasets that only have part of the original dataset to be able to create those files. To do so, we choose the 15 most common contiguous intervals of values of the waiting time in the original dataset and created 15 extra datasets to generate new configuration and knowledge base files to feed to DL-Learner. By feeding DL-Learner these new files, we get at least 150 extra rules. Since these rules are trying to predict different observations, we obtain a greater variety of rules, and thus, avoid the original problem where many of the rules were the same.

Type	Size	Value
str	31	|Month <= 1.5| and |Day > 30.5|
str	12	|Day > 30.5|
str	12	|Day > 30.5|
str	12	|Day > 30.5|
str	15	|Weekday > 0.5|
str	31	|Month <= 2.5| and |Day > 30.5|
str	15	|Weekday > 0.5|
str	15	|Weekday > 0.5|

Figure 4.10: Example of rules given by DL-Learner

Following the conversion of the rules, we clean the redundant rules again, just like we did with the 2000 rules that we extracted from the Gradient Boosting Trees so we get simplified and shortest rules, to make sure there are no redundant rules.

After getting the rules ready, we obtain the coefficients for each one of them by running different forms of regularization in order to ensemble them after. This motivated us to build three different models, one using the LASSO regression solution, one using the ElasticNet regression solution, and a third using the Ridge regression solution.

4.4.2.1 RDLL: Interpretable Rules by DL-Learner and LASSO Regression

The first model with this architecture that we built uses the same form of regularization as Rulefit, the LASSO regression. As we saw in Chapter 2.1.1 this regression solution can be described as 2.1. The name RDLL derives from the combination of Rules with DL-Learner and LASSO.

4.4.2.2 RDLE: Interpretable Rules by DL-Learner and ElasticNet Regression

The second approach to see if we could get better performance with our model was to use the ElasticNet regression instead of the LASSO regression. The ElasticNet combines the penalty from LASSO, this is, the $L1$ penalty with the penalty from Ridge regression, this is, the $L2$ penalty, to overcome certain limitations of the LASSO algorithm, as seen in Section 4.4.1. Using ElasticNet the coefficients for the rules are obtained with the formula (4.26). The name RDLE derives from the combination of Rules with DL-Learner and ElasticNet.

4.4.2.3 RDLR: Interpretable Rules by DL-Learner and Ridge Regression

Finally, our last model uses Ridge regression after obtaining the rules from DL-Learner. As we saw in Chapter 2.1.2, the Ridge regression should give us more stability than LASSO. The coefficients with this regression are obtained using the formula (2.2). The name RDLR derives from the combination of Rules with DL-Learner and Ridge.

5

Results

5.1 Accuracy

In this section, we compare the different models concerning their accuracy. The accuracy measure that we chose to analyze was the Mean Absolute Error (MAE).

5.1.1 Non-Interpretable Models

For the non Interpretable Models, the Accuracy was measured through a 5-fold cross-validation procedure. To guarantee that the models did not consider information from the future when making the predictions, each prediction was done by training the model with the data until the day before that prediction. Each model predicted the waiting time for the days April 18, April 19, April 20, April 21, and April 22 of 2019. When predicting the waiting time on April 18 the model was trained with data until April 17, the day before, then, to predict the mean waiting time on April 19, the model was trained with data until April 18, and so on. The MAE was computed for each fold, and then the average resulted in the final score for each model.

To find the best model for each emergency level for the methods using neural networks, this is, the LSTM, the GRU, and the Transformer, we also used a 5-fold cross-validation when testing different parameters to find the best model.

5.1.1.1 ARIMA

The first non-interpretable model implemented was the ARIMA model. The results of this model can be seen in Table 5.1. We notice clearly that the values for emergency level 1 were the hardest to predict, which was expected since it is the emergency level with the highest variation on the time a patient may have to wait. This variation makes it harder to make accurate forecasts of the waiting time. On the other hand, emergency level 4 was the easiest for this model to predict, which was also expected since it is the emergency level with the lowest variation in the waiting time.

Emergency Level	1	2	3	4
MAE	35.46	20.45	17.16	9.34

Table 5.1: MAE value for each emergency level in the dataset without the service column for the baseline model.

5.1.1.2 SARIMA

The results obtained with SARIMA (Table 5.2) show us that emergency level 1 was once again the hardest to predict. Unlike in the ARIMA, where the higher the emergency level, the better the forecast, the SARIMA model had a better performance on emergency level 2 than on emergency level 3, something that was not expected, since the variation on emergency level 2 is higher than on the emergency level 3. Moreover, when we compare the results from the SARIMA (Table 5.2) and the ARIMA (Table 5.1) models, we notice that the extra parameters of SARIMA (regarding the seasonality) made the accuracy worse for the emergency levels 1 and 3.

Emergency Level	1	2	3	4
MAE	36.38	15.72	19.07	9.07

Table 5.2: MAE value for each emergency level in the dataset without the service column for the SARIMA model.

5.1.1.3 Prophet

Just like ARIMA, Prophet had smaller MAE values the higher the emergency level. The MAE values on emergency levels 1 and 4 were similar to the ARIMA and SARIMA values. Differently, for emergency levels 2 and 3, Prophet had significantly higher MAE values than ARIMA and SARIMA. On these emergency levels, Prophet had a MAE of approximately more 10 minutes than SARIMA on emergency level 2 and approximately more 3 minutes than ARIMA on emergency level 3.

Emergency Level	1	2	3	4
MAE	35.40	25.65	20.64	9.77

Table 5.3: MAE value for each emergency level in the dataset for the Prophet model.

5.1.1.4 LSTM

The LSTM was the first recurrent neural network we implemented. When hyper tuning the LSTM parameters, as stated in Chapter 4.2.4.1, for the different emergency levels, we tested different learning rates, different sizes of feature maps in each hidden state, different dropout values, and different loss functions. For emergency level 1, the best set of parameters was a learning rate of 0.001, a size of 5 for the feature maps in each hidden

state, a dropout of 0, and the $L1$ loss. For emergency level 2 the parameters were the same as for emergency level 1, except for the learning rate which was 0.00001, and the loss, which was the MSE loss. For emergency level 3, the parameters were the same as for emergency level 2, except the learning rate which was 0.001. Lastly, for emergency level 4, the best set of parameters were a learning rate of 0.001, a size of 200 for the feature maps in each hidden state, a dropout of 0, and the $L1$ loss. The MAE values of LSTM are depicted in Table 5.4. Although the MAE value for emergency level 1 was again the highest, it was significantly better than the previous models. LSTM also achieved very low errors on emergency levels 2 and 3 when compared to the previous models, but had a slightly higher MAE value on emergency level 4.

Emergency Level	**1**	**2**	**3**	**4**
MAE	24.76	11.09	7.13	10.76

Table 5.4: MAE value for each emergency level in the dataset for the LSTM model.

5.1.1.5 GRU

As mentioned, the GRU was tuned on the same parameters as the LSTM. The best set of parameters for the GRU models was a learning rate of 0.001, a size of 5 for the feature maps in each hidden state, a dropout of 0, and the $L1$ loss for emergency level 1; a learning rate of 0.00001, a size of 5 for the feature maps in each hidden state, a dropout of 0 and the MSE loss for emergency level 2; a learning rate of 0.001, a size of 5 for the feature maps in each hidden state, a dropout of 0 and the MSE loss for emergency level 3 and a learning rate of 0.001, a size of 200 for the feature maps in each hidden state, a dropout of 0 and the $L1$ loss for emergency level 4. The results of the different GRU models can be observed in Table 5.5. We notice that the MAE values in every emergency level were higher than in the LSTM model, especially on the emergency level 4, which had almost more 4 minutes of error than LSTM. Nevertheless, the accuracy of GRU was still better on emergency levels 1,2, and 3 than the ARIMA, SARIMA, and Prophet forecasts.

Emergency Level	**1**	**2**	**3**	**4**
MAE	26.80	12.39	7.60	14.41

Table 5.5: MAE value for each emergency level in the dataset for the GRU model.

5.1.1.6 Transformer

As stated in Chapter 4.2.5, the Transformer was tuned on the number of heads in the multi-head attention sub-layer, the number of encoder layers in the encoder, the number of decoder layers in the decoder, and on the dimension of the feedforward network models. For emergency level 1, the best set of parameters were 8 heads in the multi-head attention sub-layer, 4 encoders, 4 decoders, and a dimension of 2048 for the feedforward network

model. For emergency level 2, the best set of parameters were 2 heads in the multi-head attention sub-layer, 5 encoders, 5 decoders, and a dimension of 1024 for the feedforward network model. For emergency level 3 the best set of parameters was the same as for emergency level 2 except for the number of heads in the multi-head attention sub-layer, which was 4. Lastly, for emergency level 4, the best parameters were 8 heads in the multi-head attention sub-layer, 5 encoders, 5 decoders, and a dimension of 256 for the feedforward network model. The results of Transformer can be seen in Table 5.6. These values show us that Transformer had similar behavior as LSTM and GRU, this is, its highest MAE value was on emergency level 1 and the lowest on emergency level 3. The values were similar both to GRU and LSTM on emergency level 1 and worse on the other emergency levels, but still better than ARIMA, SARIMA, and Prophet except on emergency level 4.

Emergency Level	**1**	**2**	**3**	**4**
MAE	25.44	14.26	10.37	13.95

Table 5.6: MAE value for each emergency level in the dataset for the Transformer model.

5.1.2 Interpretable Models

For the interpretable models, first, to have a fair comparison with the Time-Series Models, which were tested on datasets that do not include a column with the service people are going to, we measured their accuracy on the datasets that grouped the data by day, meaning that we grouped the different services of each day into a single row so we could have the same dataset as the TimeSeries models, as seen in Chapter 4.1. After, in an attempt to give the information about the past to the interpretable models, we added a column with the waiting time of the previous day. Lastly, we got the different models accuracies for the datasets including the column "Service", to have a waiting time according to the service people would go to. As stated in 4.1, the variation of the waiting time in this dataset is higher, and thus, we got higher MAE values than on the dataset without the column "Service", as we will see.

For all the models except the automatic implementation of RuleFit, using the Python package *rulefit*, and SIRUS, we hyper tuned the alpha value to achieve less relevant rules than the ones obtained by the respective model cross-validation method, which gives more importance to the accuracy rather than the number of rules. This alpha is the value that multiplies the penalty term in the regression equation. To get a lower number of significant rules that have an impact on the final result than the original number we experimented with different alpha values. Higher alpha values mean higher penalty terms, and consequently, less relevant (with a non-null weight) rules. A study about the number of relevant rules and respective accuracy in each model for each dataset can be seen in A, where we present a plot for each model for each dataset that shows the relation of the models between the number of relevant rules and accuracy. Here we will show the

results of the best compromise, in our opinion, of the number of rules and accuracy. In Chapter 5.1.3 we will see a plot of both the training accuracies and validation accuracies according to the number of rules combining all models.

The results obtained can be seen in the following sections separated by models.

5.1.2.1 RuleFit

For the RuleFit model first, we compare the results from the dataset without the service column, between the two different implementations, the automatic implementation using the Python package "*rulefit*" and our manual implementation. From Tables 5.7 and 5.8 we notice that the values are very similar and there is no considerable discrepancy of trying different implementations for this model except in emergency level 1, where our implementation of Rulefit got a slightly better MAE (38.31 *versus* 39.7). In all other emergency levels, the difference in the MAE value was less than 1 minute.

Emergency Level	**1**	**2**	**3**	**4**
MAE	39.7	21.02	14.24	7.76

Table 5.7: MAE value for each emergency level in the dataset without the service column for the RuleFit model using the Python package "*rulefit*".

Emergency Level	**1**	**2**	**3**	**4**
MAE	38.31	20.59	14.91	8.07

Table 5.8: MAE value for each emergency level in the dataset without the service column for the RuleFit model using our implementation.

The Tables 5.9 and 5.10 show the accuracies achieved by the RuleFit models on the dataset without the service column, but with a column regarding the waiting time of the previous day. We notice that this extra column improved the accuracy in all emergency levels in both implementations. Moreover, we noticed a considerable worse accuracy on the emergency level 1 on our implementation of RuleFit having a MAE value of over 2 minutes higher (MAE $\approx$ 37.72) than the MAE value of the automatic implementation of RuleFit (MAE $\approx$ 35.44). For the other emergency levels, the accuracy of our implementation of RuleFit was slightly worse on emergency levels 3 and 4 (MAE $\approx$ 13.7 and MAE $\approx$ 7.92, respectively) compared to the automatic implementation (MAE $\approx$ 13.01 and MAE $\approx$ 7.66, respectively) and slightly better on the emergency level 2 (MAE $\approx$ 20.08 on our implementation and MAE $\approx$ 20.19 on the automatic implementation).

Lastly, we compare the results from the dataset with the service column, which served only to compare interpretable models between themselves. We can see the results from the automatic implementation in the Table 5.11 and the results from our own RuleFit implementation in Table 5.12. We observe that the implementation using the Python package has better accuracies in all emergency levels. Nevertheless, As we can see in

Emergency Level	1	2	3	4
MAE	35.44	20.19	13.01	7.66

Table 5.9: MAE value for each emergency level in the dataset without the service column and with a column regarding the waiting time of the previous day for the RuleFit model using the Python package "*rulefit*".

Emergency Level	1	2	3	4
MAE	37.72	20.08	13.7	7.92

Table 5.10: MAE value for each emergency level in the dataset without the service column and with a column regarding the waiting time of the previous day for the RuleFit model using our implementation.

Appendix A, if using a smaller alpha value and thus, use more rules, we would achieve better accuracy for the emergency level 1 on the manual implementation of RuleFit than on the implementation using the Python package *rulefit*.

Emergency Level	1	2	3	4
MAE	52.91	22.64	15.75	9.76

Table 5.11: MAE value for each emergency level in the dataset with the service column for the RuleFit model using the Python package "*rulefit*".

Emergency Level	1	2	3	4
MAE	54.05	25.12	18.56	12.17

Table 5.12: MAE value for each emergency level in the dataset with the service column for the RuleFit model using our implementation.

5.1.2.2 SIRUS

The results shown here were obtained using the cross-validation method implemented in the "*sirus*" package, which chooses the model with fewer rules. We still analyzed how changing the $p0$ value in SIRUS affected the number of rules and the achieved accuracies in Appendix A, where we can see the plots created and studied, that compare the accuracy according to the number of relevant rules for each of the emergency levels in each dataset.

The accuracies of the SIRUS model regarding the first group of datasets can be seen in Table 5.13 and are very similar to the values obtained with the RuleFit model, showing that the two state-of-the-art rule-based methods perform very similar in this dataset.

Emergency Level	1	2	3	4
MAE	39.84	20.71	14.54	8.27

Table 5.13: MAE value for each emergency level in the dataset without the service column for the SIRUS model.

When we added the column with the past values of the waiting time, the accuracy improved in all but one emergency level. On emergency level 3 the MAE value was higher than in the dataset without the past value of the waiting time, which shows that it is not always linear that adding past information improves the model performance. Nevertheless, the accuracy improved on all the other 3 emergency levels, as can be seen in Table 5.14, although, not as much as in RuleFit.

Emergency Level	**1**	**2**	**3**	**4**
MAE	38.27	20.07	15.08	7.87

Table 5.14: MAE value for each emergency level in the dataset without the service column and with a column regarding the waiting time of the previous day for the SIRUS model.

The last SIRUS model included a column with the type of service each day. The results were very similar to RuleFit in emergency level 1 but significantly better in all other emergency levels as can be observed in Table 5.15.

Emergency Level	**1**	**2**	**3**	**4**
MAE	53.80	20.96	14.83	7.92

Table 5.15: MAE value for each emergency level in the dataset with the service column for the SIRUS model.

5.1.2.3 REN

By running the ElasticNet regression only one time in the REN model, ElasticNet was attributing significance to a lot of rules (maybe because the quality of the rules was not the best). In an attempt to make it choose fewer rules, besides hyper tuning the alpha value that multiplies the penalty term, we experimented with using the ElasticNet regression multiple times to see how it would affect the results both in terms of accuracy and number of rules. First, we used it only one time, then after using it one time, we extracted only the relevant rules (the ones with a non-null weight) and fed only those rules again to ElasticNet a second time. Lastly, on a third model, we repeat the previous step one more time (thus, running ElasticNet three times). Here we will see the results of the three different REN models. We will refer to the model that uses the ElasticNet regression solution one time as REN(1), the model that uses it two times as REN(2), and to the model that runs it three times as REN(3).

As previously stated, the MAE values shown here were chosen from the plots shown in Appendix A since they were the best compromise, in our opinion, between the number of rules and accuracy, and a different value was chosen for each of the three models.

The accuracy of our three different REN models on the first dataset is shown in Table 5.16. We notice that the accuracy on the emergency level 1 got slightly worse the more times we ran the ElasticNet regression, although, the number of relevant rules also got significantly lower as we will see further.

Emergency Level	1	2	3	4
REN(1) MAE	39.55	20.92	14.66	8.65
REN(2) MAE	39.65	20.91	15.02	8.57
REN(3) MAE	40.55	20.92	15.03	8.56

Table 5.16: MAE value for each emergency level in the dataset without the service column for the REN models.

On the datasets with the information about the past waiting time, we achieved the results presented in Table 5.17. We notice that for the emergency level 2, the MAE value was significantly lower on the model running the ElasticNet regression solution two times (MAE $\approx$ 18.71) compared to the other two variations of our REN model, which achieved a MAE of $\approx$ 19.7 when running the regression one time and a MAE of $\approx$ 20.73 when running it three times.

Emergency Level	1	2	3	4
REN(1) MAE	39.59	19.7	14.89	7.9
REN(2) MAE	39.53	18.71	15.11	8.03
REN(3) MAE	40.83	20.73	15.18	8.37

Table 5.17: MAE value for each emergency level in the dataset without the service column and with a column regarding the waiting time of the previous day for the REN models.

In the dataset that included the type of service, we obtained the accuracies seen in Table 5.18. Here we can notice a very significant increase in the MAE values the more times we ran the ElasticNet regression solution. Nevertheless, as we will see further, the number of relevant rules also significantly decreases. Moreover, running the ElasticNet regression solution just one time gave us a better MAE value for the emergency level 1 ($\approx$ 51.68) than the values obtained by RuleFit and SIRUS for this same dataset, $\approx$ 52.91 and $\approx$ 53.80, respectively.

Emergency Level	1	2	3	4
REN(1) MAE	51.68	27.71	17.28	9.84
REN(2) MAE	55.75	28.27	18.43	10.82
REN(3) MAE	60.67	30.77	21.04	13.39

Table 5.18: MAE value for each emergency level in the dataset with the service column for the REN models.

5.1.2.4 RDLL, RDLE, and RDLR

RDLL, RDLE, and RDLR were developed and implemented with the goal of getting rules with better quality and thus achieving better accuracy with a lower number of rules. In that sense, looking at the accuracy alone in these models is not enough, as we must take into consideration the number of rules they use, which we will see in Section 5.2. The

accuracy achieved by these models in the first dataset can be seen in Table 5.19. We notice that the MAE values of the RDLE model were higher for almost every emergency level (1,2 and 3) when compared to RDLL and RDLR. For emergency level 4, the MAE values were very similar. It is interesting to notice that the RDLR achieved a considerably better accuracy on the emergency level 2 when compared to RDLL and RDLE.

Emergency Level	**1**	**2**	**3**	**4**
RDLL MAE	39.01	20.69	14.89	8.14
RDLE MAE	40.21	21.05	15.36	8.07
RDLR MAE	39.09	18.68	14.82	7.96

Table 5.19: MAE value for each emergency level in the dataset without the service column for the RDLL, RDLE, and RDLR models.

Regarding the second set of datasets, after adding the column with the past information about the waiting time, we observe (Table 5.20) that once again the RDLE model achieved the worse accuracy of the 3 in all emergency levels except in emergency level 3, where RDLL performed worse. The RDLR model also outperformed the other two in these datasets (in terms of accuracy), and was the model out of the three that improved its performance with the new information, especially in emergency levels 1 and 3.

Emergency Level	**1**	**2**	**3**	**4**
RDLL MAE	39.51	18.74	13.52	7.87
RDLE MAE	40.94	20.42	15.02	8.35
RDLR MAE	37.72	18.57	13.69	7.84

Table 5.20: MAE value for each emergency level in the dataset without the service column and with a column regarding the waiting time of the previous day for the RDLL, RDLE, and RDLR models.

Lastly, in the dataset with the service column, the performance of the 3 models, regarding their accuracy, can be seen in Table 5.21. Once again the RDLE model achieved the worse accuracy out of the three in every emergency level. Moreover, the accuracies of the RDLL and the RDLR model were very similar in every emergency level except in emergency level 1, where RDLR achieved slightly better accuracy.

Emergency Level	**1**	**2**	**3**	**4**
RDLL MAE	52.88	24.79	18.32	10.95
RDLE MAE	58.68	29.49	21.71	12.59
RDLR MAE	51.19	24.28	17.85	11.17

Table 5.21: MAE value for each emergency level in the dataset with the service column for the RDLL, RDLE, and RDLE models.

5.1.3 Comparison

In Figure 5.1 we can compare the MAE values of the top three non-interpretable models and the top three interpretable models implemented among the different emergency levels. It is important to notice that this comparison is not completely fair since the interpretable models have access to different information from the non-interpretable models. While the non-interpretable models have access to information about all the previous waiting times in the dataset, the interpretable models only have access to information about the current date they are predicting or the current date and the waiting time of the previous day.

The three methods using neural networks, LSTM, GRU, and Transformer, not only have better accuracy than the other non-interpretable models, but they also stand out in this figure having low errors on emergency levels 1, 2, and 3. Contrarily, on emergency level 4, the interpretable methods were able to outperform the non-interpretable methods. This plot shows us that it is possible to have an interpretable model with better accuracy than a non-interpretable model when predicting the waiting time in the emergency department of a hospital and is something that should be studied before implementing a non-interpretable model.

Figure 5.2 compares the MAE values of the interpretable models among the different emergency levels on the dataset with the extra information regarding the type of service on each day. On emergency level 1 the RDLR model had the best accuracy, closely followed by REN(1). RDLL, SIRUS, and RuleFit had very similar MAE values on this emergency level, which were not far from the ones achieved by RDLR. For emergency levels 2,3, and 4, SIRUS achieved the lowest MAE values, closely followed by RuleFit on emergency levels 2 and 3. RDLL and RDLR had very similar MAE values on emergency levels 2,3, and 4 and were always lower than the ones achieved by RDLE. REN(3) had constantly one of the worst MAE values, being the worst in emergency levels 1,2, and 4.

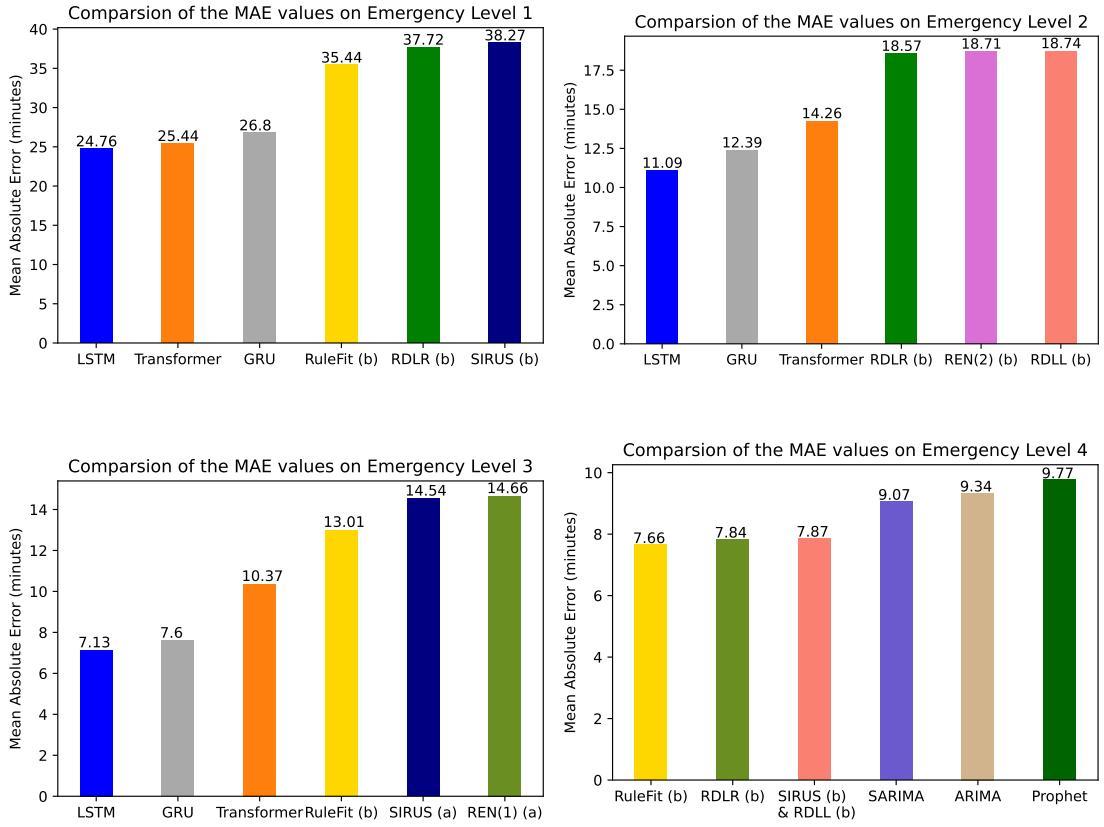


Figure 5.1: Comparison of the MAE values of the best 3 non-interpretable models and the best 3 interpretable models in the different emergency levels. The prefix (a) after the interpretable model name means that the best accuracy was achieved on the dataset without the information about the waiting time of the previous day and the prefix (b) means that it was achieved on the dataset with the information about the previous day.

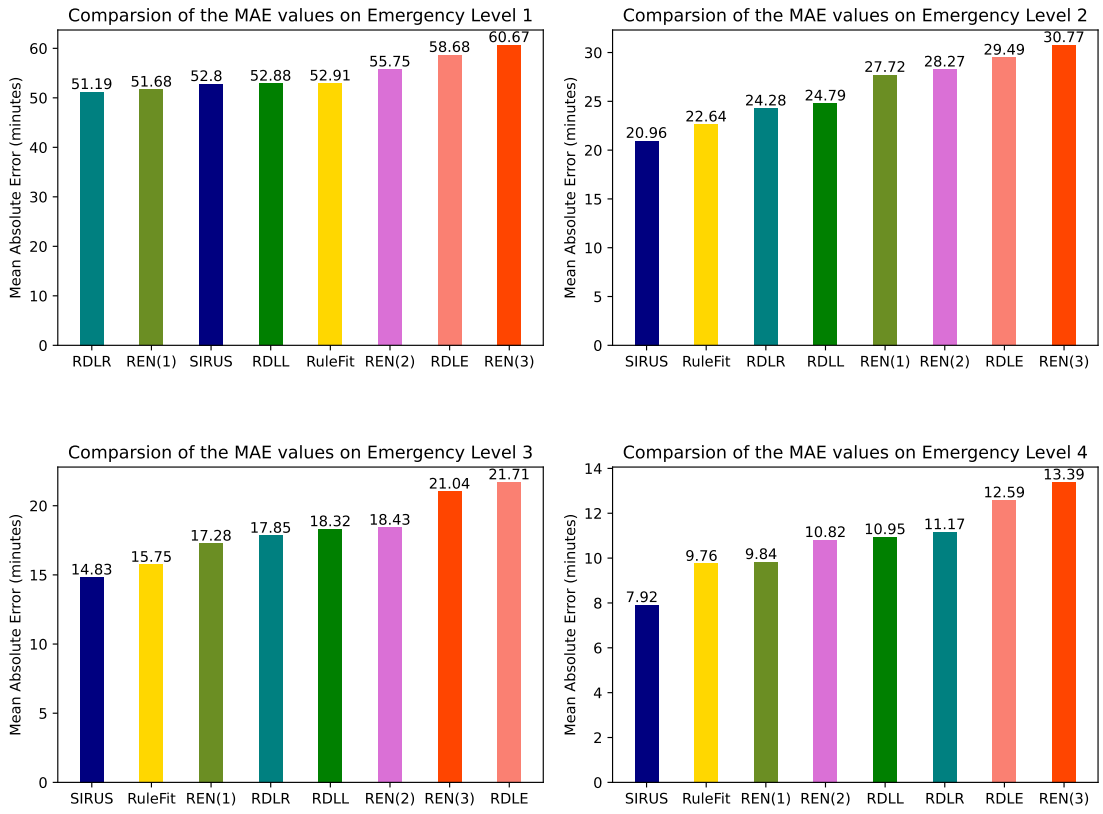


Figure 5.2: Comparison of the MAE values of all the interpretable models in the different emergency levels on the dataset with the column regarding the type of service.

5.2 Complexity

In this chapter, we compare the complexity of all models in terms of how difficult it is to understand them. For the interpretable models, this means comparing their sparsity and the number of rules each one uses. For the non-interpretable models, we can use the number of parameters each model uses as a proxy for this measure.

5.2.1 Interpretable Models

Regarding the interpretable models, a model with too many rules, even if all the rules are individually understandable, may not be interpretable as a whole.

On the first dataset, the one without the "service" column and without the information about the past, the number of rules each algorithm assigned a non-null weight, this is, the number of relevant rules, in the different emergency levels, can be seen in Table 5.22. The RDLR model included a very high number of rules for all the emergency levels (86, 81, 93, and 87), and thus, we can not consider it interpretable for this dataset. The same happened to REN (1) in emergency levels 1 and 3 and REN (2) in emergency level 1. The best models in this dataset, regarding sparsity, were SIRUS, REN (3), and RDLL, which achieved the lowest mean number of rules for all the emergency levels, 2.75, 2.25, and 4 respectively. Nevertheless, RDLE achieved the second lower number of rules (together with REN(3)) for emergency level 1, and our implementation of RuleFit achieved the second lowest number of rules for emergency level 4 (together with SIRUS, REN(1) and RDLL).

Emergency Level	**1**	**2**	**3**	**4**	**Mean**
RuleFit (1)	12	8	5	7	8
RuleFit (2)	22	7	4	3	5.5
SIRUS	2	3	3	3	2.75
REN (1)	185	11	49	3	62
REN (2)	66	5	6	7	21
REN (3)	4	1	2	2	2.25
RDLL	7	3	3	3	4
RDLE	4	10	5	6	6.25
RDLR	86	81	93	87	86.75

Table 5.22: Number of Rules in each model for the dataset without the "service " column. RuleFit (1) - Toolbox implementation of RuleFit, RuleFit (2) - Our implementation of RuleFit.

Regarding the second dataset, with the column containing information about the waiting time of the previous day, the number of significant rules for each model can be seen in Table 5.23. The first thing we notice when we compare the Tables 5.22 and 5.23 is the significant increase in the number of rules that the automatic implementation of RuleFit has in this dataset, its mean number of rules went from 8 to 85.5. Moreover, we notice a great difference in the number of rules from our implementation of RuleFit (mean of

9.25) to the implementation using the Python package "*rulefit*" on every emergency level. Once again the RDLR model achieved a very high number of rules at every emergency level, thus, we can not consider it interpretable again in this dataset. The same thing happened with REN(1) and REN(2) which also achieved a very high number of rules for every emergency level in this dataset. SIRUS, REN(3), and RDLL were once again the models with the lowest mean number of rules. Even though our implementation of RuleFit got the lowest number of rules for emergency level 1 (3), it has a higher number of relevant rules for the other emergency levels.

Emergency Level	**1**	**2**	**3**	**4**	**Mean**
RuleFit (1)	48	160	48	86	85.5
RuleFit (2)	3	15	7	12	9.25
SIRUS	4	2	3	5	3.5
REN (1)	337	142	115	90	171
REN (2)	129	75	16	32	63
REN (3)	8	7	5	2	5.5
RDLL	4	4	6	3	4.75
RDLE	6	7	2	24	9.75
RDLR	120	137	138	187	145.5

Table 5.23: Number of Rules in each model for the dataset with the extra column regarding the waiting time of the previous day. RuleFit (1) - Toolbox implementation of RuleFit, RuleFit (2) - Our implementation of RuleFit.

On the third dataset, the one including the "service" column, we notice once again that the toolbox implementation of RuleFit and the algorithms REN(1), REN(2), and RDLR have a very high number of rules (Table 5.24). The difference from the automatic implementation of RuleFit to our implementation of RuleFit is once again huge, showing the importance that hyper-tuning the alpha value on the regression equation had. Once more, the SIRUS, RDLL, and REN(3) were the 3 models with the lowest mean number of rules, although, our implementation of RuleFit and RDLE model were very close to the REN(3) mean number of relevant rules (7.25 *versus* 7). For emergency levels 1 and 4, the SIRUS and the RDLL models achieved the lowest number of relevant rules. SIRUS also achieved the lowest number of relevant rules for emergency level 2, while RDLL achieved the lowest number of rules for emergency level 3.

5.2.2 Non-Interpretable Models

Regarding the complexity of the non-interpretable models, we measure it through the number of parameters each model uses (Table 5.25). We notice an enormous difference between the number of parameters LSTM, GRU, and Transformer use compared to ARIMA, SARIMA, and Prophet. This is because of the neural networks these models use, which increase a lot their number of parameters (and thus, their complexity).

Emergency Level	1	2	3	4	Mean
RuleFit (1)	46	63	60	24	48.25
RuleFit (2)	8	7	8	6	7.25
SIRUS	3	3	4	3	3.25
REN (1)	510	272	348	389	379.75
REN (2)	64	61	52	96	68.25
REN (3)	6	7	5	10	7
RDLL	3	4	3	3	3.25
RDLE	9	8	4	8	7.25
RDLR	172	137	181	174	166

Table 5.24: Number of Rules in each model for the dataset with the "service " column. RuleFit (1) - Toolbox implementation of RuleFit, RuleFit (2) - Our implementation of RuleFit.

Emergency Level	1	2	3	4	Mean
ARIMA	5	5	5	5	5
SARIMA	9	9	9	9	9
Prophet	8	8	8	8	8
LSTM	13300	173000	2900	426	47406.5
GRU	129000	2200	2200	129000	65600
Transformer	2300000	1600000	1600000	583000	1520750

Table 5.25: Number of parameters of the different non-interpretable models in the different emergency levels.

5.3 Sparsity versus Accuracy

The last step of this thesis was a comparison between the accuracy of the interpretable models and their sparsity, this is, the number of rules to which they attribute a non-null weight. To change the number of relevant rules (the number of rules to which the models attribute non-null weight), as stated before, we changed the alpha value, which controls the penalty term in the regression equation, where higher values of alpha result in less relevant rules since the penalty term would be higher, and lower alpha values result in giving a non-null weight to more rules.

First, we make this comparison on the dataset where we do not have the "service" column, secondly on the dataset where each row has extra information about the waiting time of the previous day, and finally on the third dataset, which contains the "service" column. The RuleFit method presented in this section is our implementation of RuleFit, where we could change the alpha value (the value that controls the strength of the penalty term). We did not include the model RDLR in any of these comparison plots since the number of rules never changed when changing the alpha value. It was also opted to leave REN(1) out because of its very high number of rules that would diminish all the other lines and would make them hard to visualize. Nevertheless, RDLR and REN(1) plots can be individually seen in Appendix A together with the plots of all the other models.

On the dataset without the "service" column and without the information about the previous day, we can see how changing the alpha value, and thus, the number of relevant rules (this is, the rules with a non-null weight) affected every model accuracy on Figure 5.3. We notice that for the first emergency level the SIRUS and the RDLL models behave very similarly and have very similar values in accuracy when using the same number of relevant rules. RuleFit seems to have better accuracy when using more rules in this emergency level, while REN(2), REN(3), and RDLE have worse accuracies than the other models, even when using more rules.

On the second emergency level, all models except the REN(2) behave very similarly when using more and less relevant rules, this is, they all start with better and similar accuracies when using more rules and get worse accuracies (and also similar) when using less relevant rules.

For emergency level 3, we notice that SIRUS has always the lowest relation between the number of rules and MAE, especially when using fewer rules. RDLL has a different behavior from every other model in this emergency level, its accuracy seems to get better when using fewer rules, which can be a result of overfitting when using more relevant rules.

On emergency level 4, the RuleFit model always achieves a better accuracy when using the same amount of relevant rules as the other models. Nonetheless, it is closely followed by RDLL when using less relevant rules and its behavior is similar to all the other models in this emergency level.

Regarding the dataset containing information about the previous day, the relation between the number of relevant rules and accuracy for the different models except the REN(1) and the RDLR can be seen in Figure 5.4.

For emergency level 1, RuleFit has a different line from all the other models, it starts with a bad accuracy when using more relevant rules and gets a better accuracy when using less relevant rules. This is because the model is overfitted when using a lot of rules as can be seen in Appendix A, where the training accuracy is very low when using more relevant rules but the validation accuracy is high and generalized better with a lower number of rules. REN(2), REN(3), RDLL, and RDLE all saw their accuracies getting worse the fewer rules they used. SIRUS, on the other hand, behaved differently, its accuracy seemed to be sometimes better and sometimes worse with more rules, which may be caused by the rules the model chooses, some may have been worse than others. RuleFit has the best accuracy on this emergency level when using a lower number of relevant rules.

For emergency level 2 we noticed that the RDLL model did not have significantly better accuracy when using more rules, unlike all the other models which had worse accuracies when using a lower number of rules. The RDLL and the SIRUS models seem to have the best compromise between the number of rules and accuracy in this emergency level, where RDLL has better accuracy when using less relevant rules.

On emergency levels 3 and 4 all models behaved similarly. They all had better accuracies when giving importance to more rules and they all got worse the less relevant rules

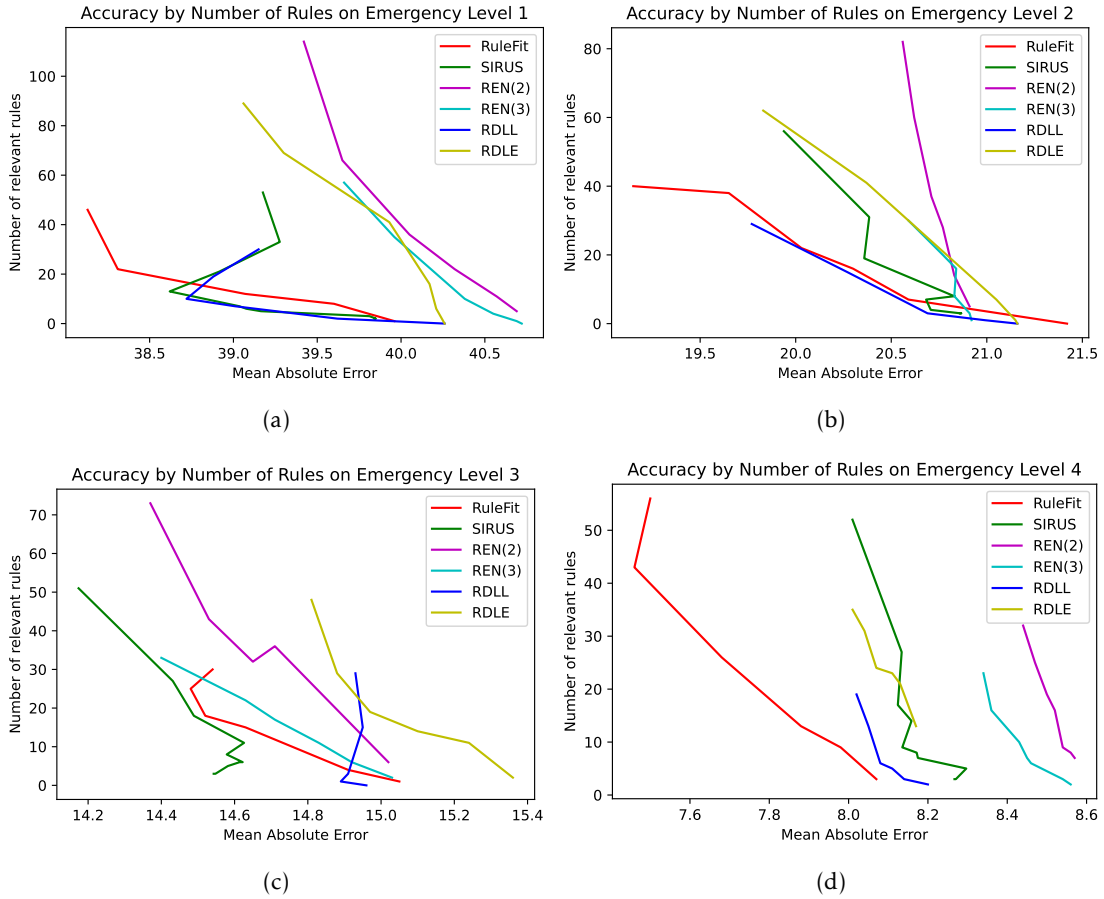


Figure 5.3: Relation between the number of relevant rules (rules given a non-null weight by the regression equation) with the Mean Absolute Error in the different emergency levels for RuleFit, SIRUS, REN(2), REN(3), RDLL, and RDLE on the dataset without the "service" column and without the column containing information about the past. Emergency levels 1 (a) and 2 (b) are shown at the top (left and right, respectively), and emergency levels 3 (c) and 4 (d) are shown on the bottom (left and right, respectively).

they used. The novel RDLL model developed in this thesis has always the best compromise between the number of rules and accuracy for the emergency level 3. Regarding the emergency level, 4 SIRUS appeared to be the best model when using a lower number of relevant rules, closely followed by RDLL which was able to use even fewer rules to obtain similarly (but slightly higher) MAE values.

Regarding the last dataset, which had a column with the type of service, the different relations between the number of relevant rules and accuracies can be observed in Figure 5.5.

On emergency level 1 every model behaved similarly and had a similar plot, they all had better accuracies when considering more rules with a non-null weight. However, RDLL seems to be almost always the model with the MAE value by number of relevant rules, closely followed by RuleFit and SIRUS when considering fewer rules.

SIRUS revealed himself as the best model by a significant difference from the other

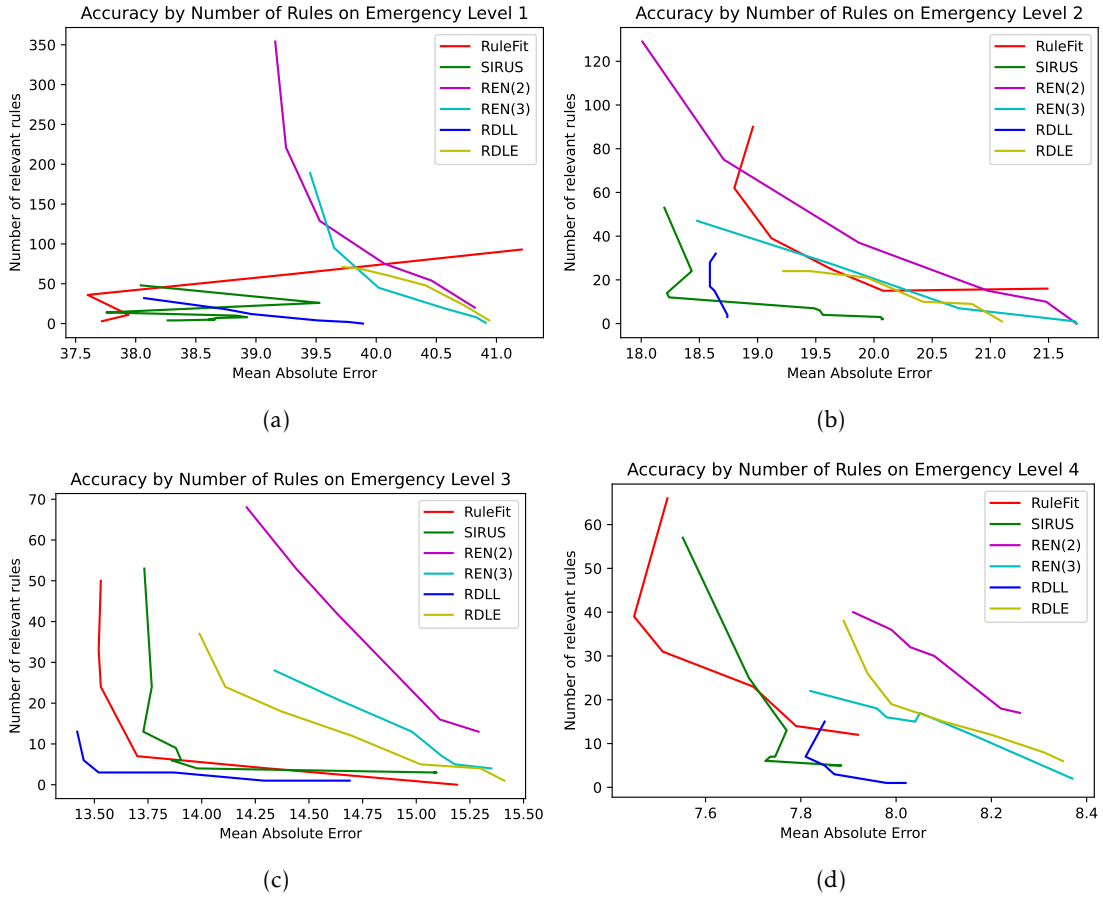


Figure 5.4: Relation between the number of relevant rules (rules given a non-null weight by the regression equation) with the Mean Absolute Error in the different emergency levels for RuleFit, SIRUS, REN(2), REN(3), RDLL, and RDLE on the dataset containing information about the previous day without the "service" column. Emergency levels 1 (a) and 2 (b) are shown at the top (left and right, respectively), and emergency levels 3 (c) and 4 (d) are shown on the bottom (left and right, respectively).

models on emergency level 2. The closest models with the second-best number of relevant rules by accuracy were RuleFit and RDLL. Even though all models had a similar behavior once again, SIRUS had significantly lower values of MAE when considering the same number of relevant rules than every other model.

For emergency levels 3 and 4 SIRUS, once again, revealed much better results than every other model. Nevertheless, RDLL still achieved the second-best accuracy when considering a low number of relevant rules in both of these emergency levels.

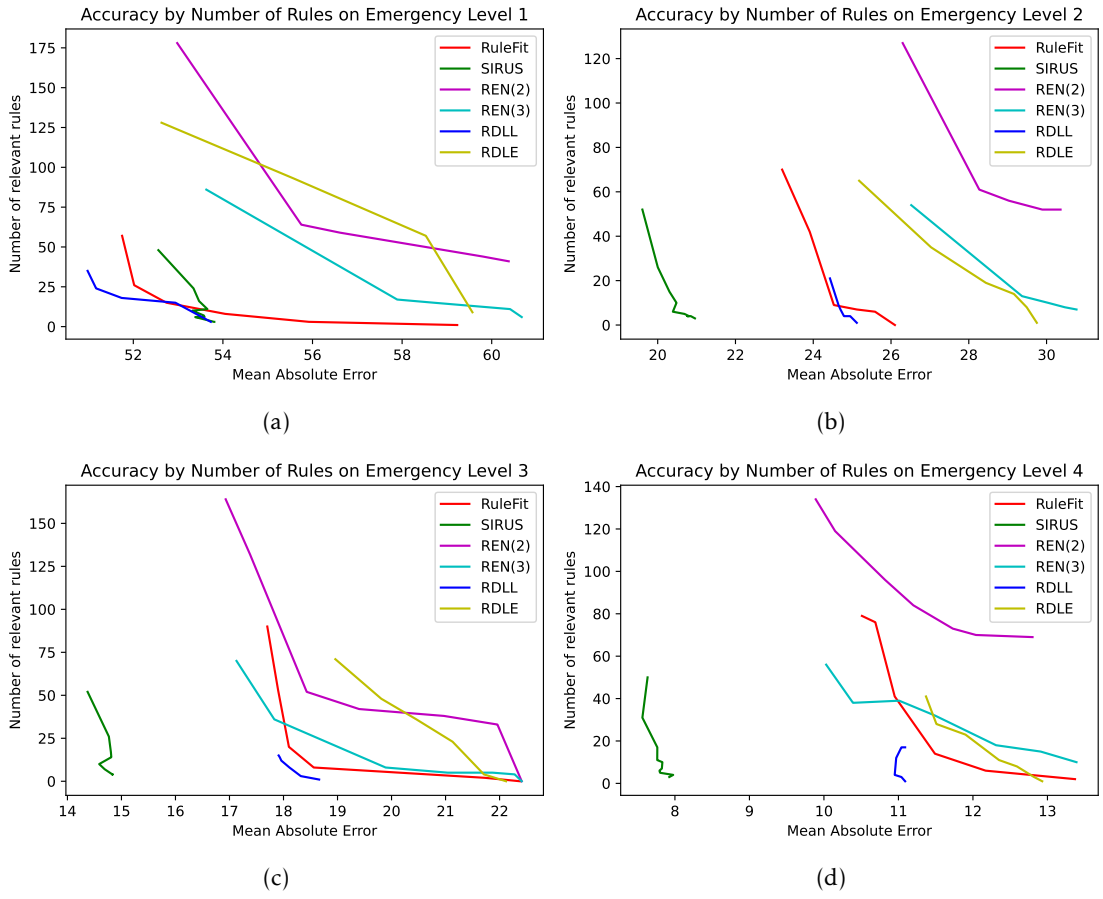


Figure 5.5: Relation between the number of relevant rules (rules given a non-null weight by the regression equation) with the Mean Absolute Error in the different emergency levels for RuleFit, SIRUS, REN(2), REN(3), RDLL, and RDLE on the dataset containing information about the type of service. Emergency levels 1 (a) and 2 (b) are shown at the top (left and right, respectively), and emergency levels 3 (c) and 4 (d) are shown on the bottom (left and right, respectively).

6

Conclusion

Machine learning is present everywhere nowadays, with different algorithms deciding what we see or listen to (for example, which advertisement we see on the internet or music recommendations based on what we listened to). Even though in those activities the output of the algorithm may be harmless, when applying these algorithms in high-stake problems, like the medical sector, we should understand the reason behind that output. Most machine learning algorithms are "black-box", and thus, we can not understand the reason behind the output. This need for understanding gave rise to an interest in interpretable models, which is translating into new and better machine learning models where we can understand the predictions made.

The goal of this thesis was to prove that an interpretable machine learning algorithm could achieve accuracies similar to a "black-box" model. To do so, besides using state-of-the-art interpretable machine learning models, we introduced four novel interpretable approaches. Firstly, we implemented six non-interpretable ("black-box") models (ARIMA, SARIMA, Prophet, LSTM, GRU, and Transformer) to have a baseline comparison, and then six interpretable models (RuleFit, SIRUS, REN, RDLL, RDLE, and RDLR), four of which were introduced in this thesis (REN, RDLL, RDLE, and RDLR). After implementing them, we compared the results from each algorithm, first according to their accuracy, and then according to their complexity, in terms of how hard it is to understand their predictions.

6.1 Contributions and Discussion

Concerning the accuracies of the different models implemented, it was observed that for every model the hardest emergency level to predict the waiting time was level 1, which was normal since it's the one with the highest variability. Moreover, this emergency level has the highest limit of waiting time (240 minutes), this is, the maximum waiting time the Portuguese national health service tries to guarantee to its patients in the hospital "Santa Maria". Emergency levels 2,3, and 4 have a limit of 120, 60, and 10 minutes respectively, and is probably the reason, together with the lower variation, why most models had lower

errors on higher emergency levels.

As seen in Chapter 5, in terms of accuracy, the LSTM, GRU, and Transformer had considerably lower errors in emergency levels 1,2, and 3. Nonetheless, despite having the lowest errors on emergency levels 1,2, and 3, LSTM, GRU, and Transformer had the most complex models, by far, even when compared to the other "black-box" alternatives (ARIMA, SARIMA, and Prophet) and their predictions can not be explained. On emergency level 4, our interpretable approaches were able to outperform every "black-box" model, including LSTM, GRU, and Transformer. This shows us that interpretable models can have even better performances than "black-box" approaches. RuleFit achieved the lowest errors among the interpretable methods on emergency levels 1,2 and 4. In emergency level 2, the best accuracy among the interpretable methods was achieved by RDLR, closely followed by REN(2) and RDLL.

Considering that some of the interpretable models implemented had similar or even better accuracies than ARIMA, SARIMA, and Prophet on emergency levels 1,2,3 and better performance than every "black-box" model in emergency level 4, we believe that interpretable models can be used to address the problem of predicting the waiting times in emergency departments. Moreover, as can be observed in tables 6.1 and 6.2, both implementations of RuleFit, RDLL, and RDLR had a lower mean absolute error across the 4 emergency levels than the ARIMA, SARIMA, and Prophet models.

	ARIMA	**SARIMA**	**Prophet**	**LSTM**	**GRU**	**Transformer**
Mean MAE	20.6	20.06	22.87	16.435	15.3	16.00

Table 6.1: Mean MAE values (in minutes) of the "black-box " models.

	RuleFit	**SIRUS**	**REN(1)**	**REN(2)**	**REN(3)**	**RDLL**	**RDLE**	**RDLR**
Mean MAE	19.08 | 19.86	20.19	20.45	20.32	21.17	19.79	20.93	19.46

Table 6.2: Mean MAE values (in minutes) of the interpretable models. The two MAE values on RuleFit represent the MAE of the automatic implementation on the left and our implementation on the right.

Regarding the complexity of the models, LSTM, GRU, and Transformer, had by far the highest complexity. When comparing just the "black-box" models, these 3 models had tens of thousands of parameters compared to the 5,8 and 9 of ARIMA, Prophet, and SARIMA, respectively, which makes it hard to just explain how the models are working. Concerning the sparsity of interpretable models, the results showed us that some of them may not be considered interpretable as a whole, since they have a very high number of rules to explain the model predictions. SIRUS, RDLL, and REN(3) had the lowest number of rules across every dataset tested, which makes them the easiest models to understand.

The novel RDLL model introduced in this thesis not only has one of the lowest numbers of rules to explain its predictions, but it also has one of the lowest mean errors among the interpretable models, as seen in table 6.2 (including a lower mean error than SIRUS and REN(3)). We believe that this model had the best relation between sparsity and accuracy across all models studied, followed by SIRUS and RuleFit.

We can conclude that even though the prediction of the waiting time in an emergency department is extremely hard due to its high variation, it is possible to use an interpretable machine learning model instead of a "black-box" one to make these predictions. This way every person can understand the reason behind the predicted waiting times without the need of being a Data Scientist or a domain expert. Moreover, if the reason for the predictions doesn't make sense, it is possible to change the model so that it gives less importance to the rules that don't make sense.

All the code used in this thesis, including all the models' implementation, is publicly available on *https://github.com/LourencoVasconcelos/master-thesis*.

6.2 Limitations and Future Work

The work on this thesis has some limitations that we must take into consideration. The first limitation was the size of the dataset, having only 1 year and a half of observations (531 days), we could not obtain yearly patterns. Secondly, as we saw in some other studies in Section 2.2, introducing weather variables can improve the models' performance, and we did not have this data in this work. Moreover, we also did not have any information about the emergency department itself, which was seen in Section 2.2 to be extremely helpful in predicting the emergency department waiting times. This lack of extra information may mean that the models here presented may have to be altered to work even better with that information.

Following the progress made in this thesis we propose some future work. Firstly, regarding the prediction of the waiting times in emergency departments, including more features with information about the weather, information about the emergency department, information regarding the national health service, or some other information that domain experts think may influence these waiting times in the models could improve the forecasts, as was seen in other studies. Secondly, the newly proposed methods RDLL, RDLE, and RDLR could be studied in different areas where an interpretable model may be needed, as it was seen that these novel methods can have accuracies similar to or even better than RuleFit and SIRUS, which are the currently best state-of-the-art rule-based methods. Moreover, the DL-Learner software may be an extremely helpful tool for interpretable methods, something that was investigated in this thesis, and should be further investigated in different problems and different solutions.

Bibliography

[1] B. Yu and K. Kumbier. "Three principles of data science: predictability, computability, and stability (PCS)". In: *arXiv preprint arXiv:1901.08152* 23 (2019) (cit. on p. 1).

[2] C. Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215 (cit. on pp. 1, 7, 8).

[3] S. Lapuschkin et al. "Unmasking Clever Hans predictors and assessing what machines really learn". In: *Nature communications* 10.1 (2019), pp. 1–8 (cit. on p. 2).

[4] R. Wexler. "When a computer program keeps you in jail". In: *New York Times* (2017) (cit. on pp. 2, 8).

[5] H. Lakkaraju and O. Bastani. ""How do I fool you?"Manipulating User Trust via Misleading Black Box Explanations". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 2020, pp. 79–85 (cit. on p. 3).

[6] A. A. Freitas. "Comprehensible classification models: a position paper". In: *ACM SIGKDD explorations newsletter* 15.1 (2014), pp. 1–10 (cit. on p. 3).

[7] T. Miller. "Explanation in artificial intelligence: Insights from the social sciences". In: *Artificial intelligence* 267 (2019), pp. 1–38 (cit. on p. 3).

[8] C. Molnar. "A guide for making black box models explainable". In: *URL: https://christophm. github. io/interpretable-ml-book* (2018) (cit. on p. 3).

[9] W. J. Murdoch et al. "Definitions, methods, and applications in interpretable machine learning". In: *Proceedings of the National Academy of Sciences* 116.44 (2019), pp. 22071–22080 (cit. on p. 3).

[10] F. Doshi-Velez and B. Kim. "Towards a rigorous science of interpretable machine learning". In: *arXiv preprint arXiv:1702.08608* (2017) (cit. on p. 3).

[11] Z. C. Lipton. "The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery." In: *Queue* 16.3 (2018), pp. 31–57 (cit. on p. 4).

[12] C. Rudin et al. "Interpretable machine learning: Fundamental principles and 10 grand challenges". In: *Statistics Surveys* 16 (2022), pp. 1–85 (cit. on pp. 5–7, 10).

[13] M. A. M. Afnan et al. "Ethical implementation of artificial intelligence to select embryos in in vitro fertilization". In: *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*. 2021, pp. 316–326 (cit. on p. 6).

[14] C. Rudin and J. Radin. *Why Are We Using Black Box Models in AI When We Don't Need To? A Lesson From An Explainable AI Competition. Harvard Data Science Review, Vol. 1, 2 (2019)*. 2019 (cit. on pp. 6, 7).

[15] E. Angelino et al. "Learning certifiably optimal rule lists for categorical data". In: *arXiv preprint arXiv:1704.01701* (2017) (cit. on p. 6).

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012) (cit. on p. 6).

[17] Q. Zhang, Y. N. Wu, and S.-C. Zhu. "Interpretable convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8827–8836 (cit. on p. 6).

[18] B. Dimanov et al. "You Shouldn't Trust Me: Learning Models Which Conceal Unfairness From Multiple Explanation Methods." In: *SafeAI@ AAAI*. 2020 (cit. on p. 7).

[19] Y. Zhang et al. ""Why Should You Trust My Explanation?"Understanding Uncertainty in LIME Explanations". In: *arXiv preprint arXiv:1904.12991* (2019) (cit. on p. 7).

[20] B. Mittelstadt, C. Russell, and S. Wachter. "Explaining explanations in AI". In: *Proceedings of the conference on fairness, accountability, and transparency*. 2019, pp. 279–288 (cit. on p. 7).

[21] G. A. Miller. "The magical number seven, plus or minus two: Some limits on our capacity for processing information." In: *Psychological review* 63.2 (1956), p. 81 (cit. on p. 9).

[22] T. Elomaa. "In defense of C4. 5: Notes on learning one-level decision trees". In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 62–69 (cit. on p. 9).

[23] J. Angwin et al. "Machine Bias. ProPublica (2016)". In: *URL: https://www. propublica. org/artic le/machine-bias-risk-assessments-incriminal-sentencing* (2016) (cit. on p. 10).

[24] Y. Lou et al. "Accurate intelligible models with pairwise interactions". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 623–631 (cit. on p. 10).

[25] H. Binder and G. Tutz. "A comparison of methods for the fitting of generalized additive models". In: *Statistics and Computing* 18.1 (2008), pp. 87–99 (cit. on p. 11).

[26] Y. Lou, R. Caruana, and J. Gehrke. "Intelligible models for classification and regression". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, pp. 150–158 (cit. on p. 11).

[27] J. H. Friedman and B. E. Popescu. "Predictive learning via rule ensembles". In: *The Annals of Applied Statistics* 2.3 (2008), pp. 916–954 (cit. on pp. 13–15, 54).

[28] C. Bénard et al. "Interpretable random forests via rule extraction". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 937–945 (cit. on pp. 17, 20, 21).

[29] Y. Sun et al. "Forecasting daily attendances at an emergency department to aid resource planning". In: *BMC emergency medicine* 9.1 (2009), pp. 1–9 (cit. on p. 21).

[30] W.-C. Juang et al. "Application of time series analysis in modelling and forecasting emergency department visits in a medical centre in Southern Taiwan". In: *BMJ open* 7.11 (2017), e018628 (cit. on p. 22).

[31] S. S. Jones et al. "Forecasting daily patient volumes in the emergency department". In: *Academic Emergency Medicine* 15.2 (2008), pp. 159–170 (cit. on p. 22).

[32] H. J. Kam, J. O. Sung, and R. W. Park. "Prediction of daily patient numbers for a regional emergency medical center using time series analysis". In: *Healthcare informatics research* 16.3 (2010), pp. 158–165 (cit. on p. 22).

[33] M. Carvalho-Silva et al. "Assessment of forecasting models for patients arrival at emergency department". In: *Operations Research for Health Care* 18 (2018), pp. 112–118 (cit. on p. 23).

[34] Y.-H. Kuo et al. "An integrated approach of machine learning and systems thinking for waiting time prediction in an emergency department". In: *International journal of medical informatics* 139 (2020), p. 104143 (cit. on p. 23).

[35] Y. Sun et al. "Real-time prediction of waiting time in the emergency department, using quantile regression". In: *Annals of emergency medicine* 60.3 (2012), pp. 299–308 (cit. on p. 24).

[36] A. Pak, B. Gannon, and A. Staib. "Predicting waiting time to treatment for emergency department patients". In: *International Journal of Medical Informatics* 145 (2021), p. 104303 (cit. on p. 24).

[37] E. Ang et al. "Accurate emergency department wait time prediction". In: *Manufacturing & Service Operations Management* 18.1 (2016), pp. 141–156 (cit. on p. 24).

[38] E. Benevento, D. Aloini, and N. Squicciarini. "Towards a real-time prediction of waiting times in emergency departments: A comparative analysis of machine learning techniques". In: *International Journal of Forecasting* (2021) (cit. on p. 25).

[39] L. Semenova, C. Rudin, and R. Parr. "On the existence of simpler machine learning models". In: *2022 ACM Conference on Fairness, Accountability, and Transparency*. 2022, pp. 1827–1858 (cit. on p. 41).

[40] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018 (cit. on p. 41).

[41] S. Hu. "Akaike information criterion". In: *Center for Research in Scientific Computation* 93 (2007) (cit. on p. 42).

[42] H. D.-G. Acquah. "Comparison of Akaike information criterion (AIC) and Bayesian information criterion (BIC) in selection of an asymmetric price relationship". In: (2010) (cit. on p. 42).

[43] R. W. Divisekara, G. Jayasinghe, and K. Kumari. "Forecasting the red lentils commodity market price using SARIMA models". In: *SN Business & Economics* 1.1 (2021), pp. 1–13 (cit. on p. 43).

[44] S. J. Taylor and B. Letham. *Forecasting at scale*. Tech. rep. PeerJ Preprints, 2017 (cit. on p. 43).

[45] L. R. Medsker and L. Jain. "Recurrent neural networks". In: *Design and Applications* 5 (2001), pp. 64–67 (cit. on p. 45).

[46] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on pp. 45–47).

[47] F. A. Gers, J. Schmidhuber, and F. Cummins. "Learning to forget: Continual prediction with LSTM". In: *Neural computation* 12.10 (2000), pp. 2451–2471 (cit. on pp. 45, 48).

[48] H. Sak, A. W. Senior, and F. Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling". In: *INTERSPEECH*. 2014, pp. 338–342 (cit. on p. 48).

[49] K. Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014) (cit. on p. 49).

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. `http://www.deeplearningbook.org`, acessed 29 July 2022. MIT Press, 2016 (cit. on p. 49).

[51] J. Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014) (cit. on p. 49).

[52] X. Li, A. Yuan, and X. Lu. "Multi-modal gated recurrent units for image description". In: *Multimedia Tools and Applications* 77.22 (2018), pp. 29847–29869 (cit. on p. 50).

[53] A. Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 51–53).

[54] A. Natekin and A. Knoll. "Gradient boosting machines, a tutorial". In: *Frontiers in neurorobotics* 7 (2013), p. 21 (cit. on p. 54).

[55] J. H. Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232 (cit. on p. 55).

[56] H. Zou and T. Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the royal statistical society: series B (statistical methodology)* 67.2 (2005), pp. 301–320 (cit. on p. 56).

[57] J. Lehmann. "DL-Learner: learning concepts in description logics". In: *The Journal of Machine Learning Research* 10 (2009), pp. 2639–2642 (cit. on pp. 57, 58).

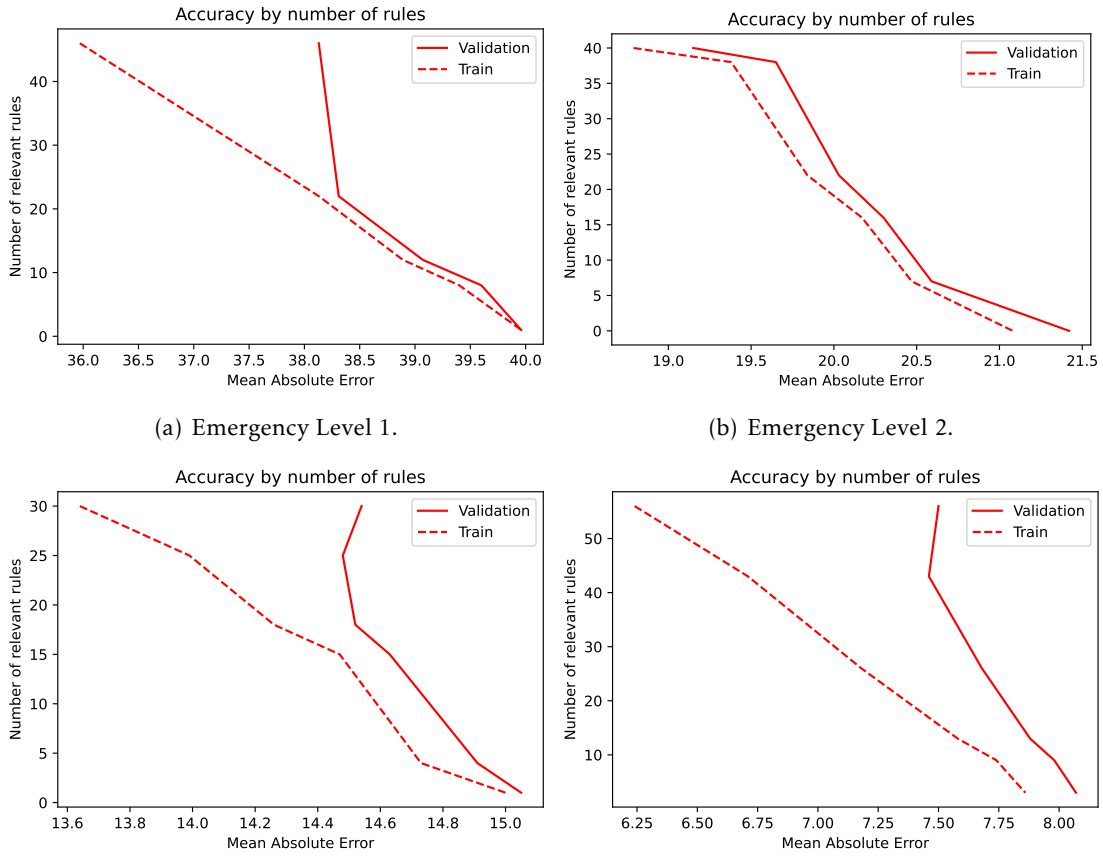
[58] J. Lehmann et al. "Class expression learning for ontology engineering". In: *Journal of Web Semantics* 9.1 (2011), pp. 71–81 (cit. on p. 58).

A

Interpretable Models' Accuracy by Number of Relevant Rules

In this appendix we present the studied plots of each interpretable model of mean absolute error by number of rules with a non-null weight (relevant rules) in each of the emergency levels in the different datasets. Each plot shows two lines, one regarding the accuracy on the training set and a second one regarding the accuracy on the validation set. The plots regarding the REN models show six lines, two for each variation of this model (one regarding the training accuracy and another regarding the validation accuracy). As mentioned before, the RuleFit model presented here is our own implementation of RuleFit. For the variations of the REN model, we noticed that running the ElasticNet solution more times did not necessarily lower the accuracy as observed in A.3. Lastly, for the RDLR model, since changing the alpha value in the Ridge regression equation did not affect the number of rules to which it attributed a non-null weight, we studied how changing this value would affect the attribution of weights and consequently how it changed the model's accuracy (presented in A.6).

A.1 RuleFit



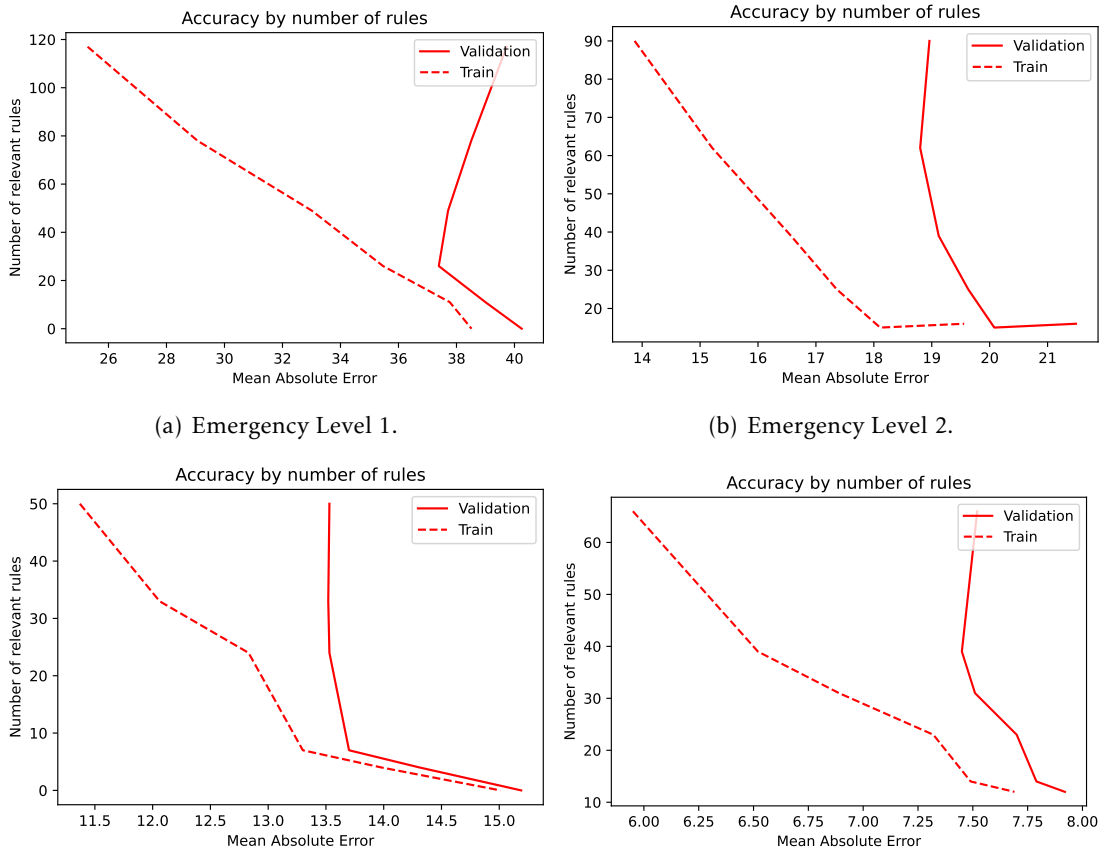
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.1: Relation between the number of rules given a non-null weight and the mean absolute error of our implementation of RuleFit on the dataset without the column regarding the service and without information about the past on the different emergency levels.



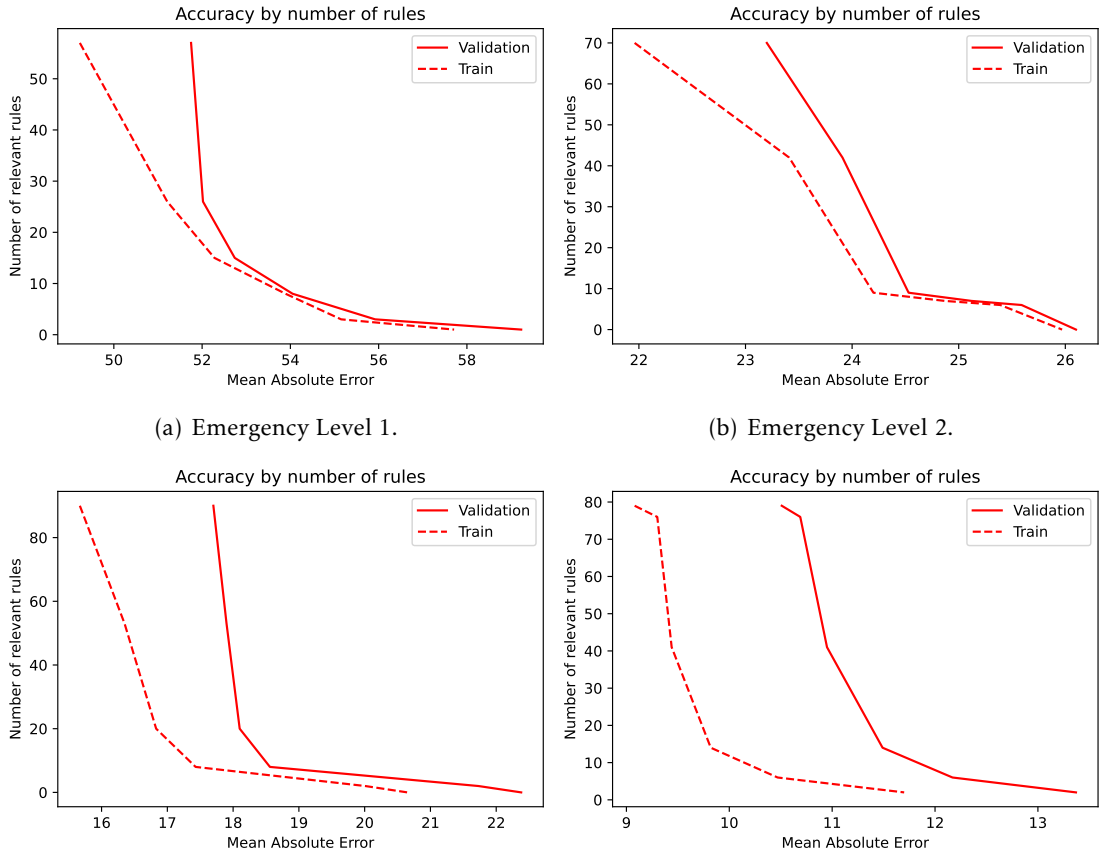
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.2: Relation between the number of rules given a non-null weight and the mean absolute error of our implementation of RuleFit on the dataset without the column regarding the service and with information about the past on the different emergency levels.



(a) Emergency Level 1.

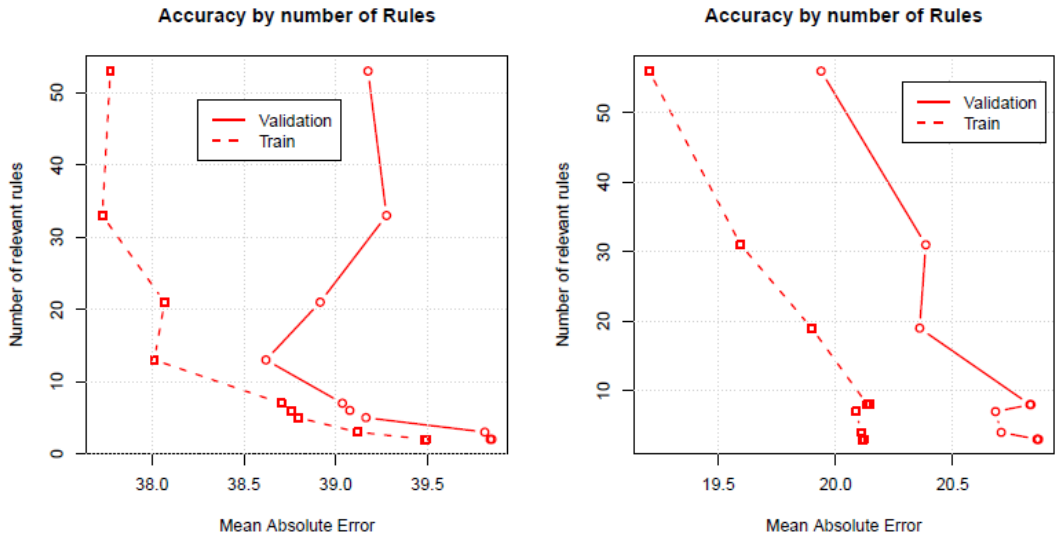
(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

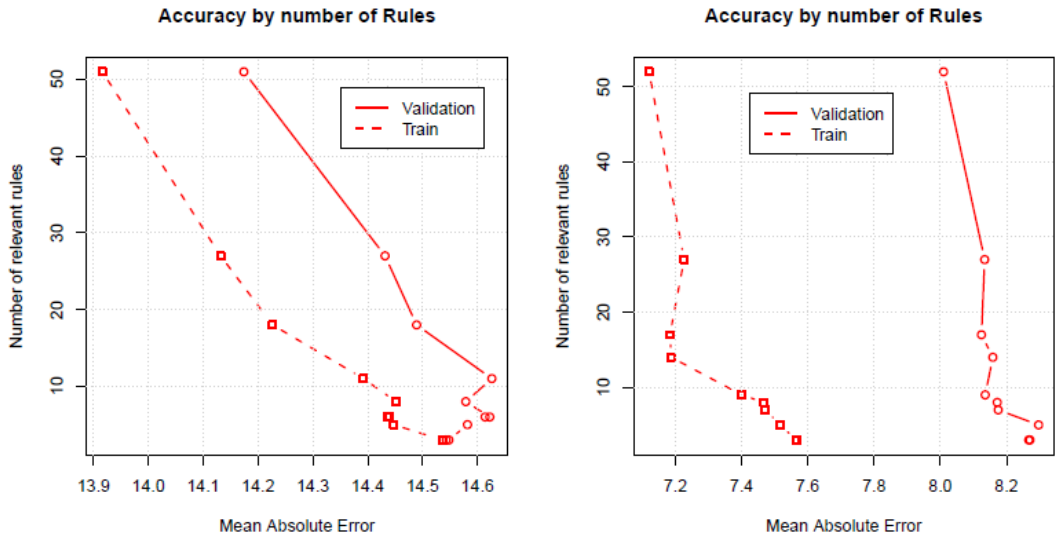
Figure A.3: Relation between the number of rules given a non-null weight and the mean absolute error of our implementation of RuleFit on the dataset with the column regarding the service on the different emergency levels.

A.2 SIRUS



(a) Emergency Level 1.

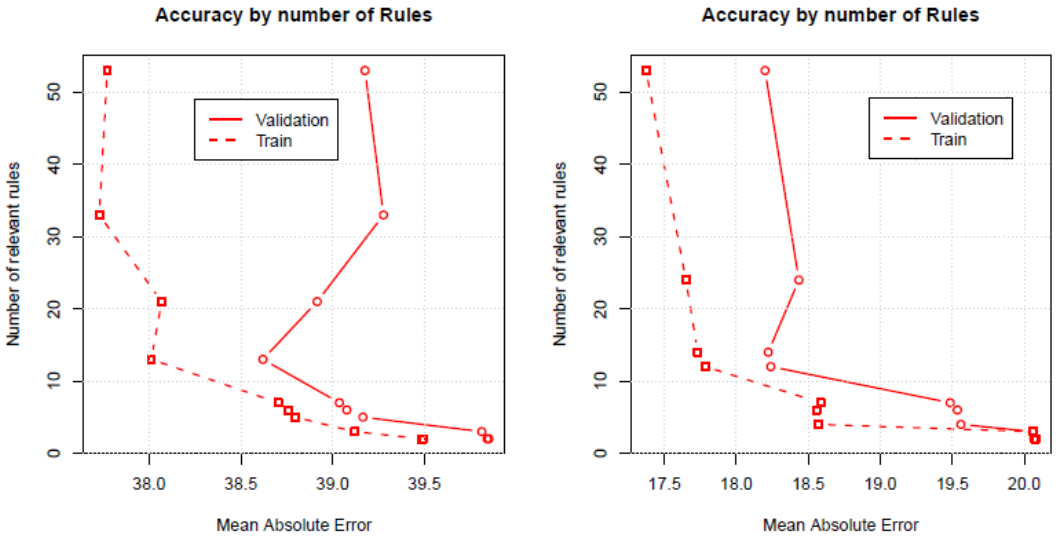
(b) Emergency Level 2.



(c) Emergency Level 3.

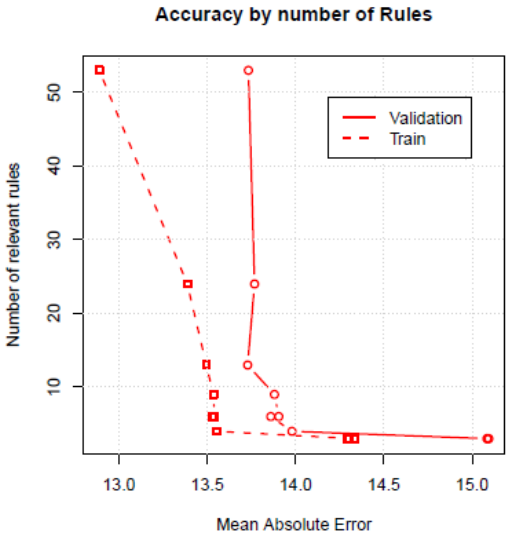
(d) Emergency Level 4.

Figure A.4: Relation between the number of rules given a non-null weight and the mean absolute error of SIRUS on the dataset without the column regarding the service and without information about the past on the different emergency levels.

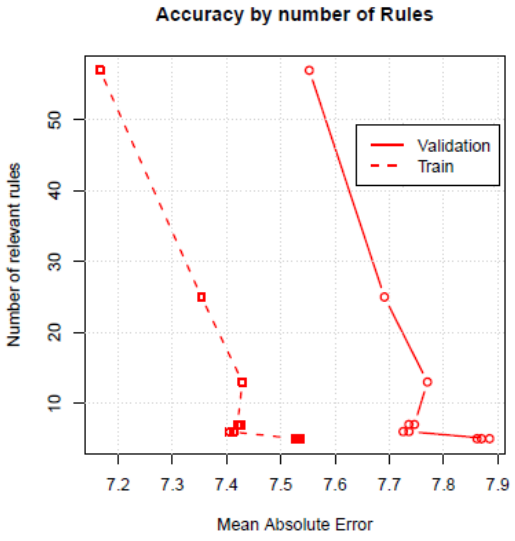


(a) Emergency Level 1.

(b) Emergency Level 2.

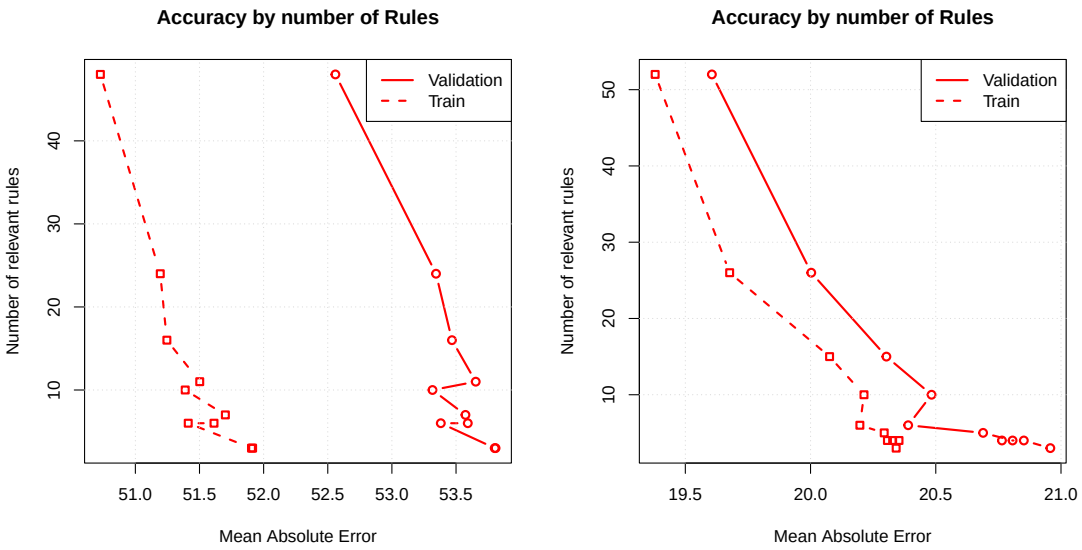


(c) Emergency Level 3.



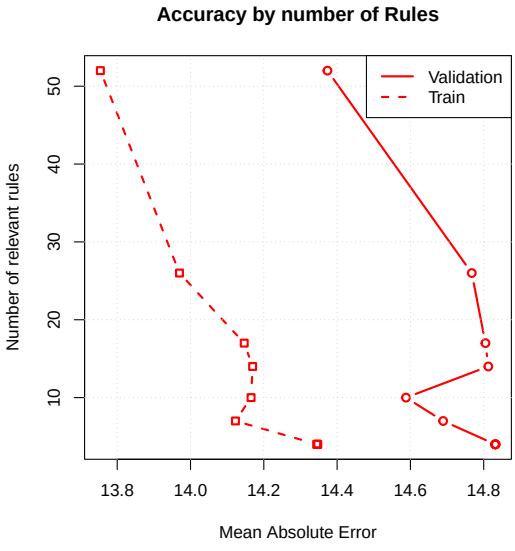
(d) Emergency Level 4.

Figure A.5: Relation between the number of rules given a non-null weight and the mean absolute error of SIRUS on the dataset without the column regarding the service and with information about the past on the different emergency levels.

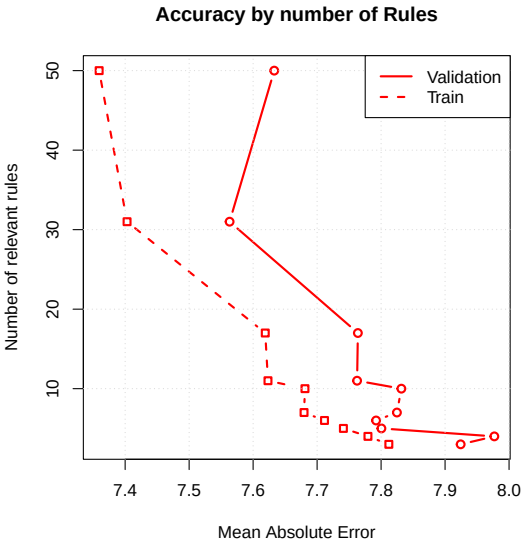


(a) Emergency Level 1.

(b) Emergency Level 2.



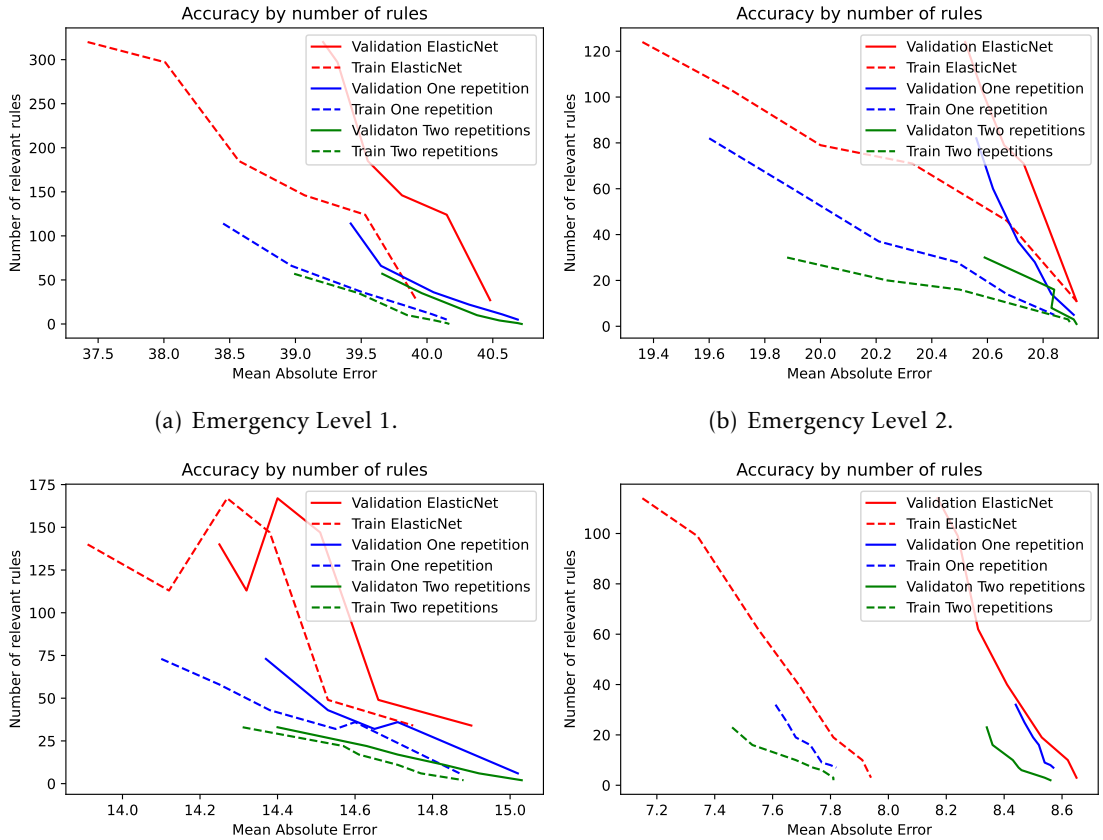
(c) Emergency Level 3.



(d) Emergency Level 4.

Figure A.6: Relation between the number of rules given a non-null weight and the mean absolute error of SIRUS on the dataset with the column regarding the service on the different emergency levels.

A.3 REN



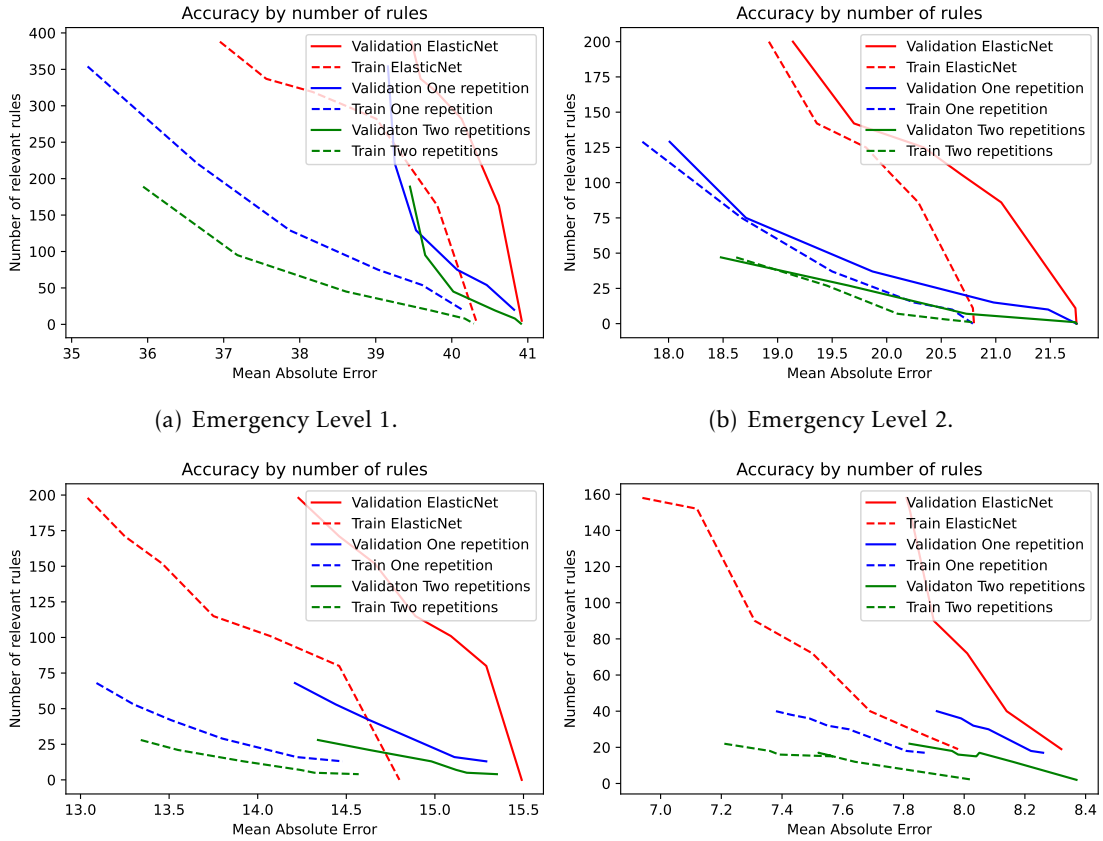
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.7: Relation between the number of rules given a non-null weight and the mean absolute error of our three different REN implementations (running ElasticNet one, two and three times) on the dataset without the column regarding the service and without information about the past on the different emergency levels.



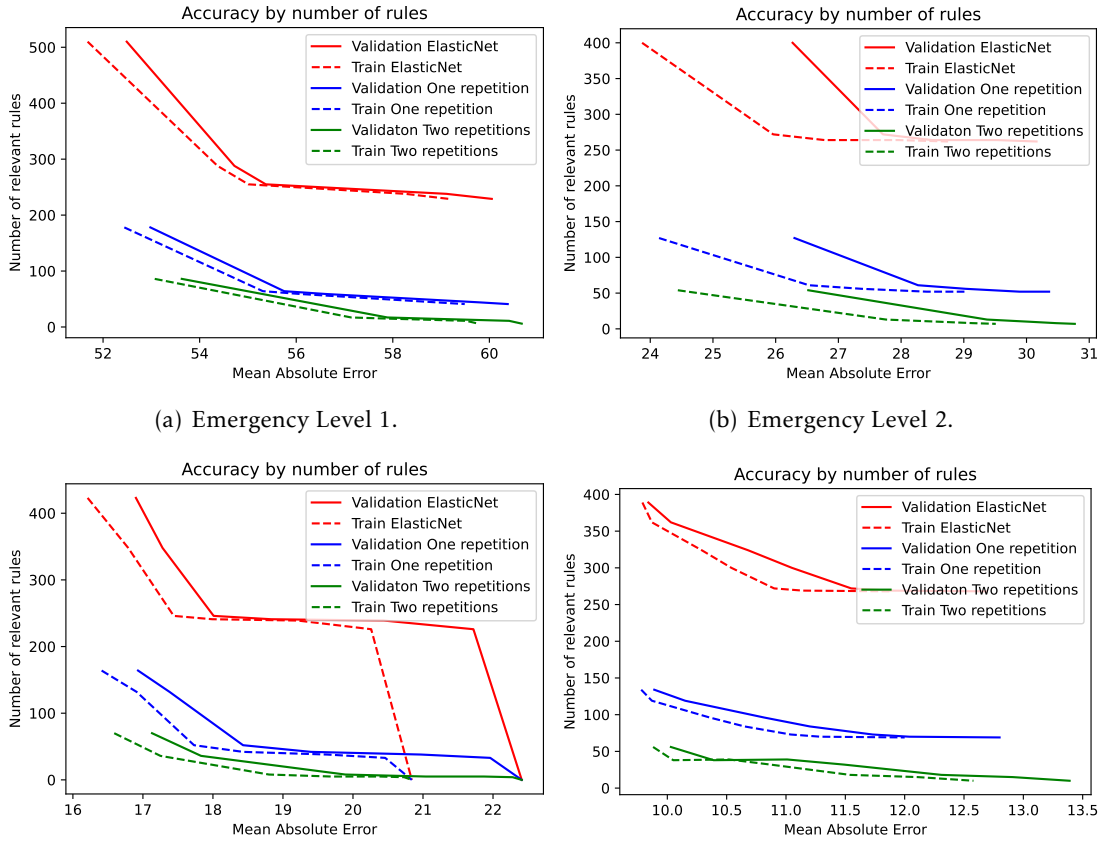
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.8: Relation between the number of rules given a non-null weight and the mean absolute error of our three different REN implementations (running ElasticNet one, two and three times) on the dataset without the column regarding the service and with information about the past on the different emergency levels.



(a) Emergency Level 1.

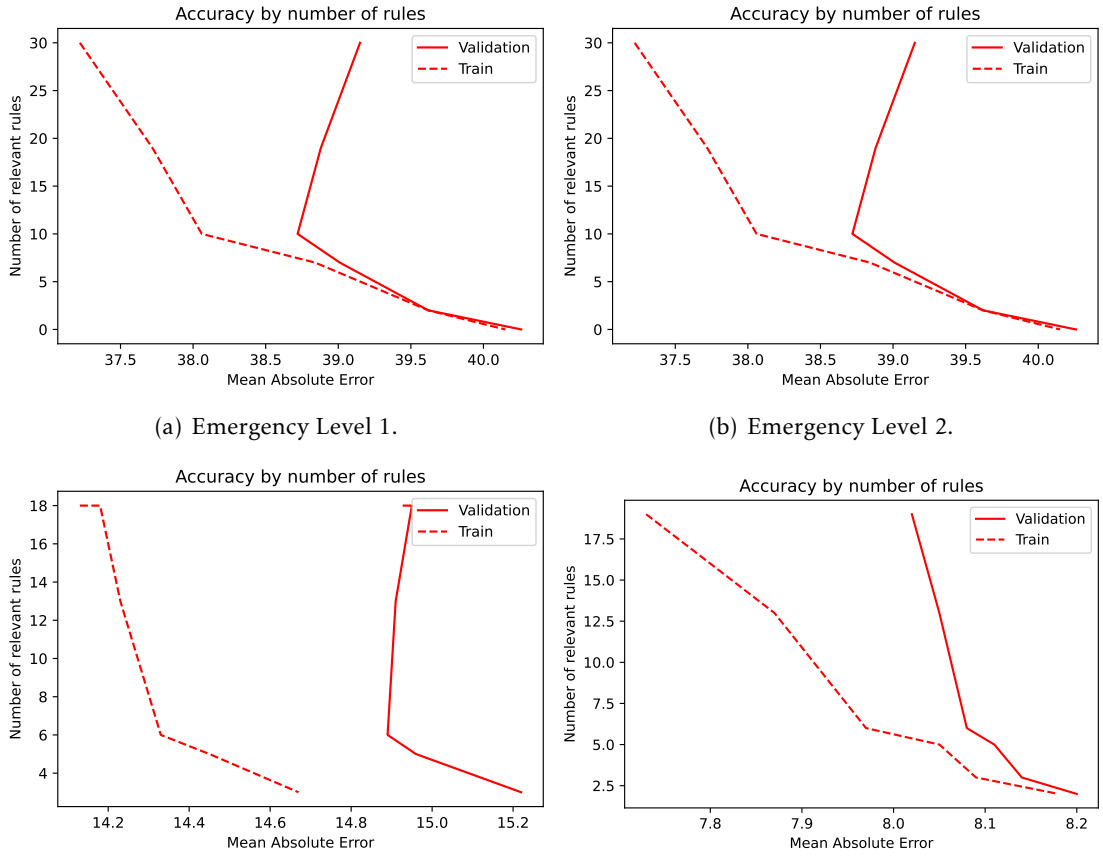
(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.9: Relation between the number of rules given a non-null weight and the mean absolute error of our three different REN implementations (running ElasticNet one, two and three times) on the dataset with the column regarding the service on the different emergency levels.

A.4 RDLL



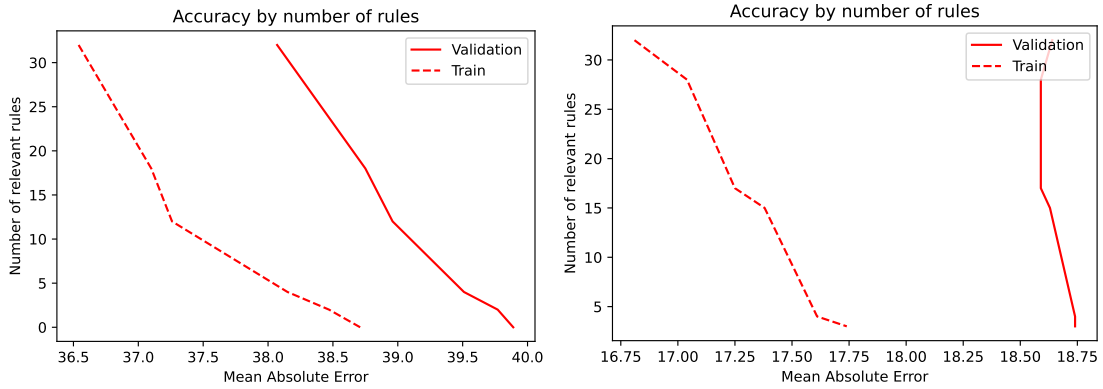
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

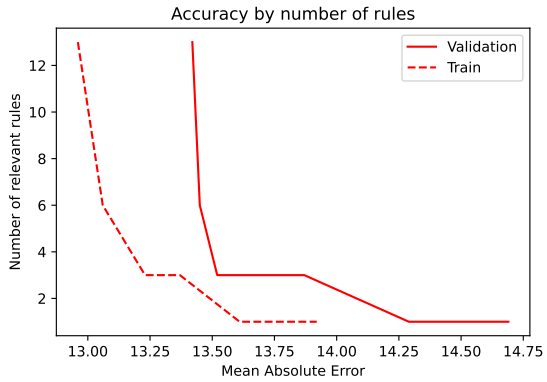
(d) Emergency Level 4.

Figure A.10: Relation between the number of rules given a non-null weight and the mean absolute error of the RDLL model on the dataset without the column regarding the service and without information about the past on the different emergency levels.

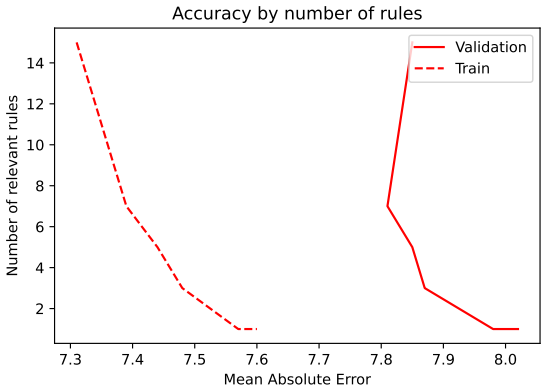


(a) Emergency Level 1.

(b) Emergency Level 2.

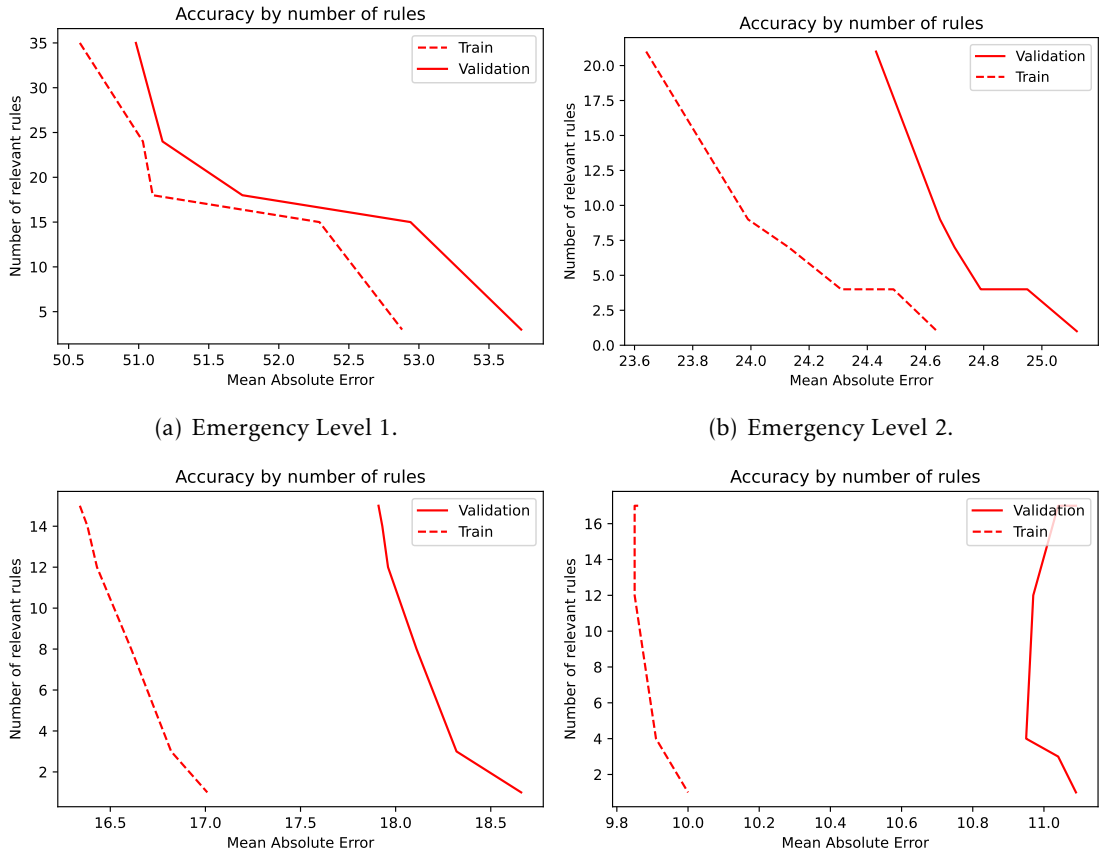


(c) Emergency Level 3.



(d) Emergency Level 4.

Figure A.11: Relation between the number of rules given a non-null weight and the mean absolute error of the RDLL model on the dataset without the column regarding the service and with information about the past on the different emergency levels.



(a) Emergency Level 1.

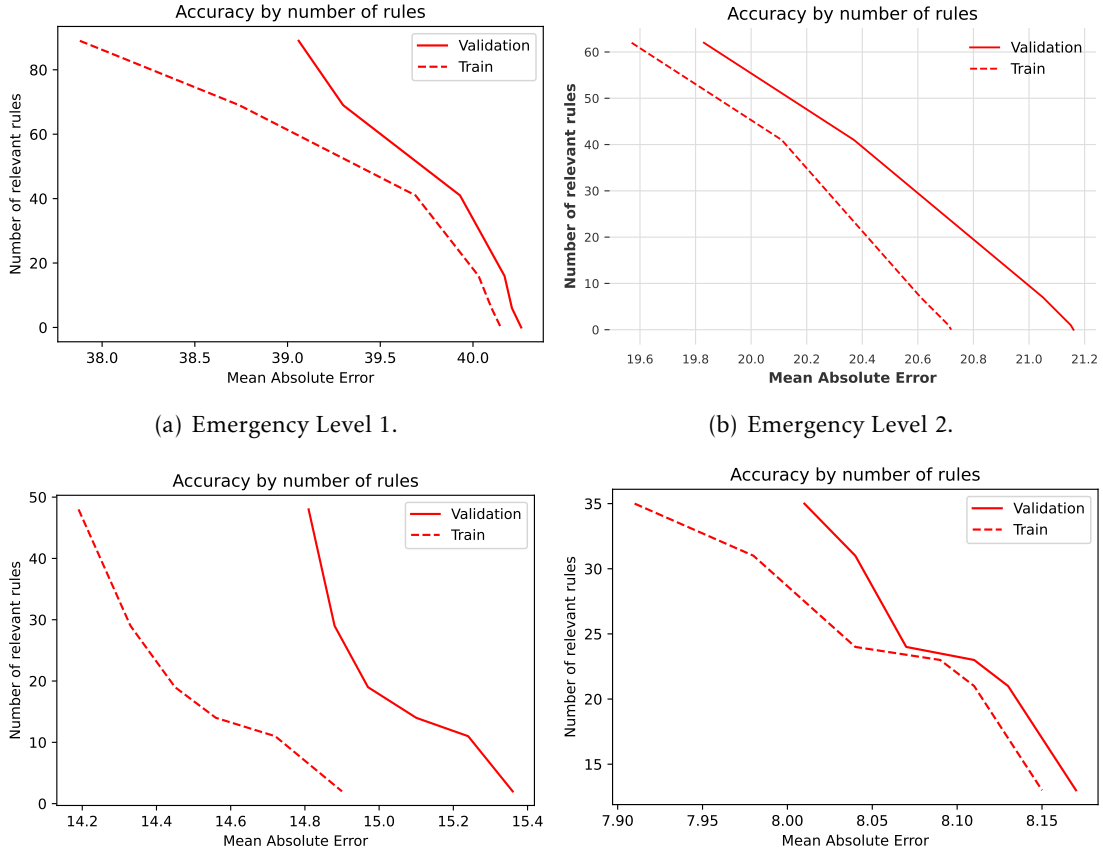
(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.12: Relation between the number of rules given a non-null weight and the mean absolute error of the RDLL model on the dataset with the column regarding the service on the different emergency levels.

A.5 RDLE



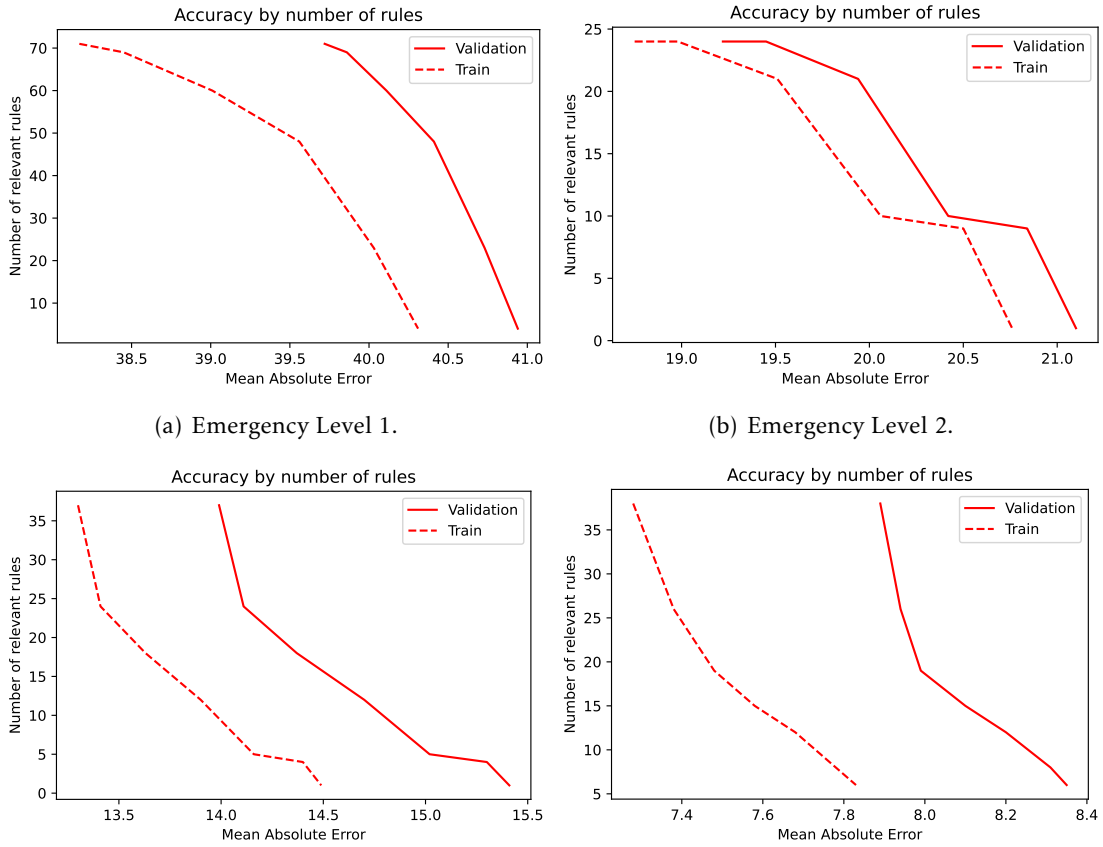
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.13: Relation between the number of rules given a non-null weight and the mean absolute error of the RDLE model on the dataset without the column regarding the service and without information about the past on the different emergency levels.



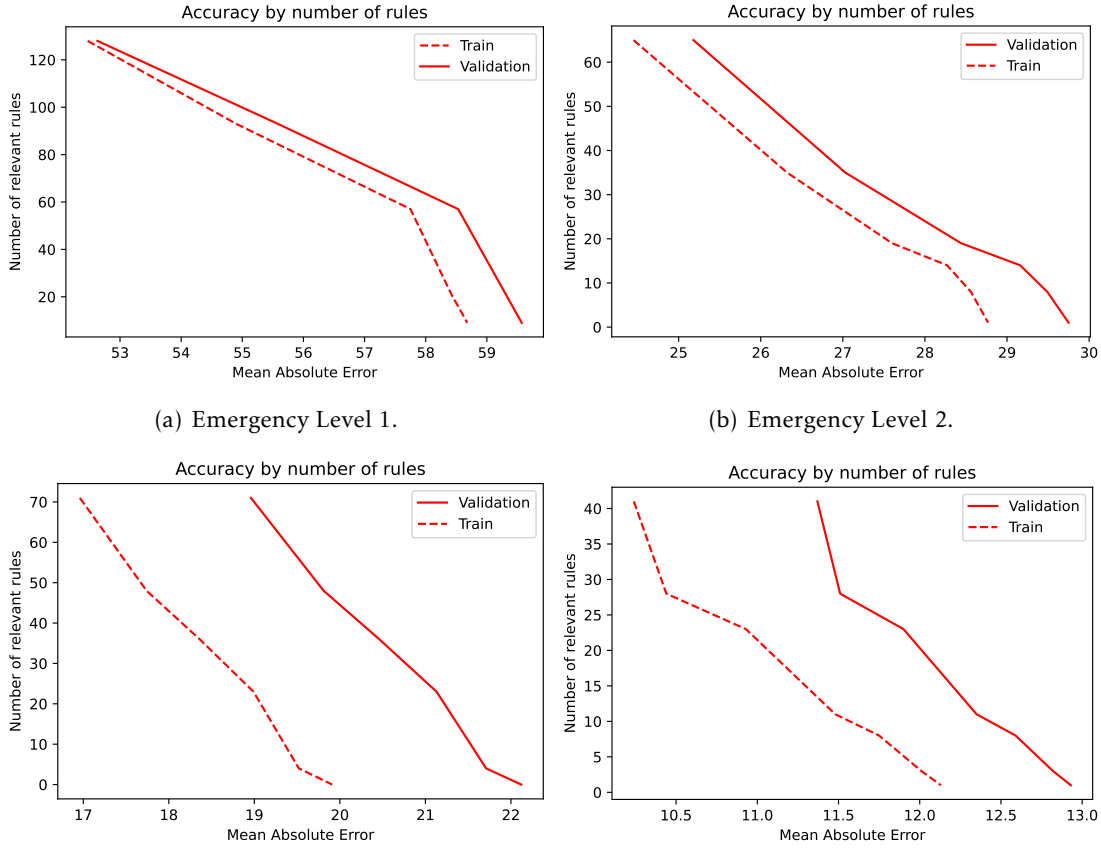
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.14: Relation between the number of rules given a non-null weight and the mean absolute error of the RDLE model on the dataset without the column regarding the service and with information about the past on the different emergency levels.



(a) Emergency Level 1.

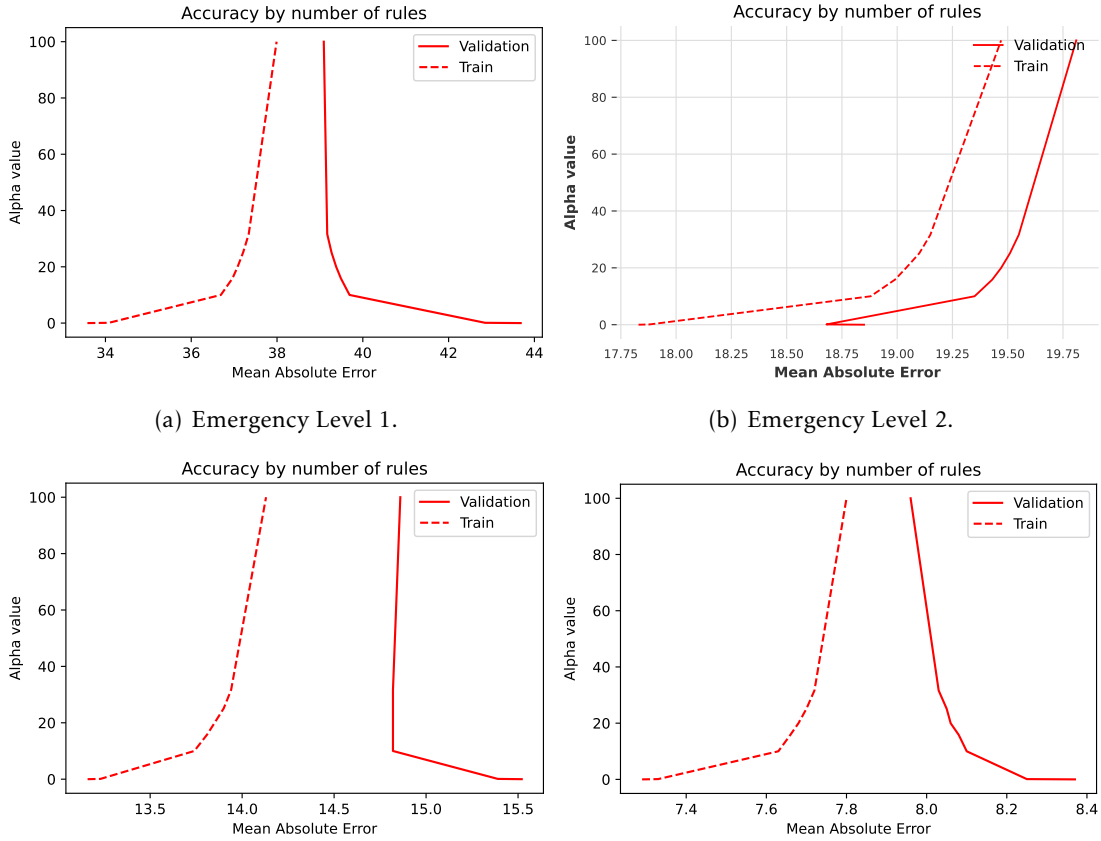
(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.15: Relation between the number of rules given a non-null weight and the mean absolute error of the RDLE model on the dataset with the column regarding the service on the different emergency levels.

A.6 RDLR



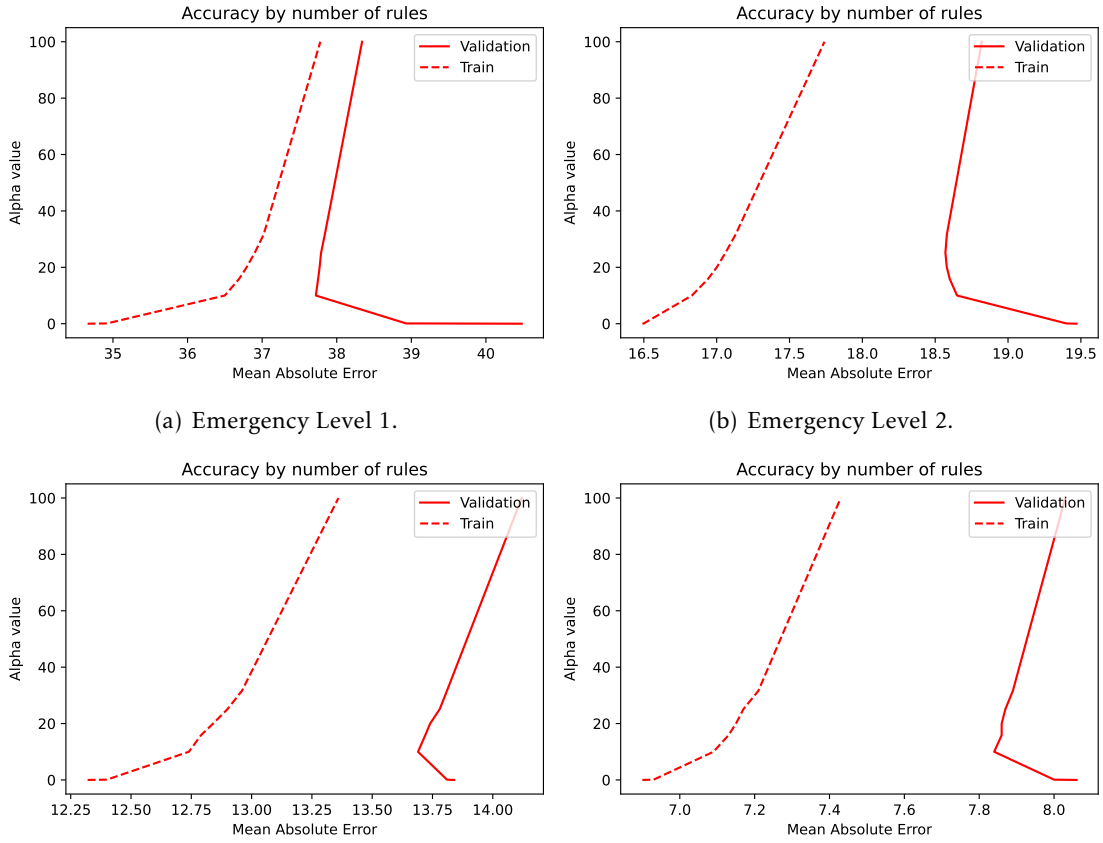
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.16: Relation between the alpha value in the Ridge regression equation and the mean absolute error of the RDLR model on the dataset without the column regarding the service and without information about the past on the different emergency levels.



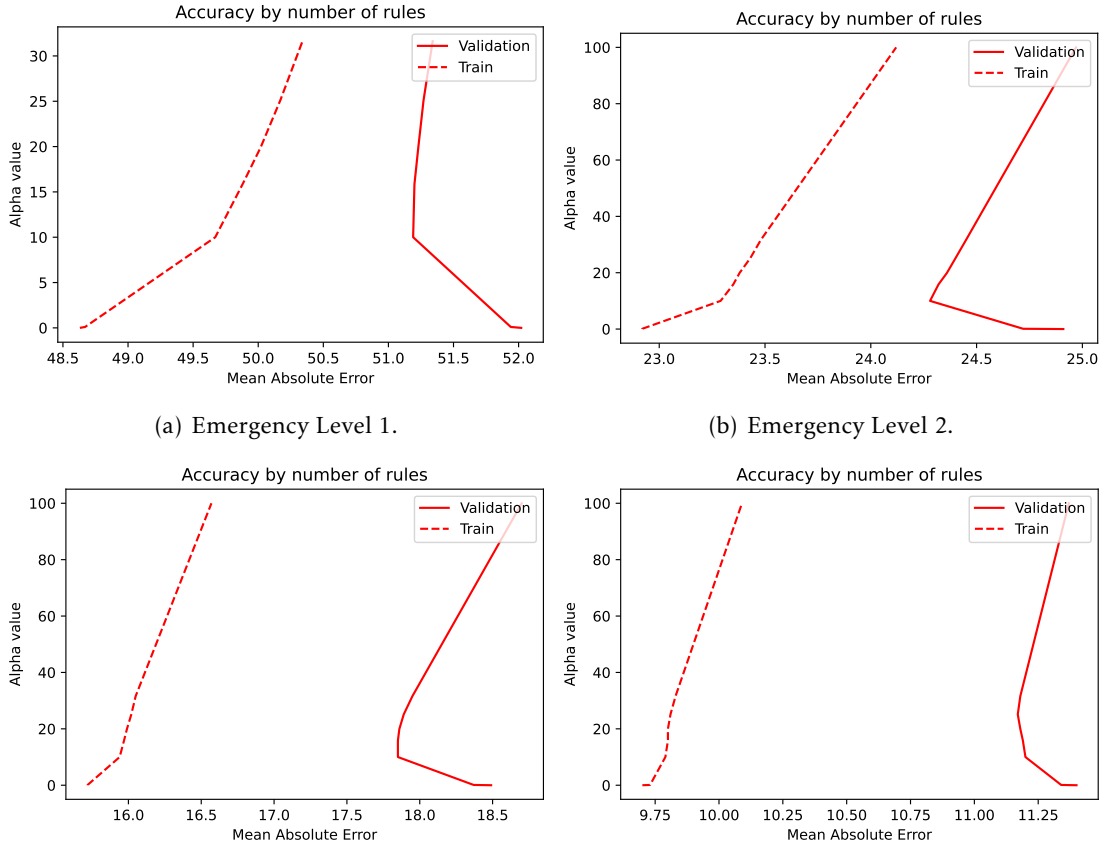
(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.17: Relation between the alpha value in the Ridge regression equation and the mean absolute error of the RDLR model on the dataset without the column regarding the service and with information about the past on the different emergency levels.



(a) Emergency Level 1.

(b) Emergency Level 2.

(c) Emergency Level 3.

(d) Emergency Level 4.

Figure A.18: Relation between the alpha value in the Ridge regression equation and the mean absolute error of the RDLR model on the dataset with the column regarding the service on the different emergency levels.