Universidade do Minho
Escola de Engenharia
Departamento de Informática

Pedro Jorge Rito Lima

Machine Learning applied to Fault Correlation

October 2021

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Pedro Jorge Rito Lima

Machine Learning applied to Fault Correlation

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Professor Rui Mendes: Universidade do Minho
Engenheiro Carlos Guilherme Araújo: Altice Labs

October 2021

ACKNOWLEDGEMENTS

To my supervisors at industry and academy, Eng. Guilherme Araújo and Professor Rui Mendes, for their advice and guidance. I would also like to thank Guilherme for the numerous well-paced briefings, that allowed me to better understand Altice Labs' tool and the project's main challenges, and Professor Rui Mendes for the scientific suggestions to improve my work.

To Altice Labs for including me on the team and providing me a scholarship for developing this project.

In a more relaxed writing style:
To my parents who supported and endured me during the most difficult times of this project and all previous ones. In particular, to Solange who, if I don't give a direct thanks to, will disown me. In fact, she deserves it for guiding me in key academic and life decisions.

To my grandmother, who was on the frontline of my education from a young age, and pushed me to never settle below greatness. I believe she would be proud of me, however, for sure she would point out things to improve, as always.

To my friends who have always been by my side, providing memories of a lifetime, and without whom this dissertation would have been finished a month ago.

To my sisters, always creating chaos that prevents me from being relaxed at home. Indirectly, they helped me by leaving me no choice but to isolate myself to finish the dissertation. Inês deserves a special bonus for encouraging me in pursuing AI research.

Thank you all, and to myself, who did the hard work without ever losing much mental sanity.

STATEMENT OF INTEGRITY

ABSTRACT

Over the last years, one of the areas that have most evolved and extended its application to a multitude of possibilities is Artificial Intelligence (AI). With the increasing complexity of the problems to be solved, human resolution becomes impossible, as the amount of information and patterns that can be detected is limited, while AI thrives on the dimension of the problem under analysis. Furthermore, as nowadays more and more traditional devices are computerized, an increasing number of elements are producing data that has many potential applications. Consequently, we find ourselves at the height of Big Data, where huge volumes of data are generated, being entirely unfeasible to process and analyze them manually.

Additionally, with the increasing complexity of network topologies, it is necessary to ensure the correct functioning of all equipment, avoiding cascade failures among devices, which can lead to catastrophic consequences depending on their use. Thus, Root Cause Analysis (RCA) tools become fundamental since these are developed to automatically, through rules established by its users, realize the underlying causes when some equipment malfunctions. However, with the growing network complexity, the definition of rules becomes exponentially more complicated as the possible points of failure scale drastically.

In this context, framed by the Altice Labs RCA and network environment use case, the main objective of this research project is defined. The aim is to use Machine Learning (ML) techniques to extrapolate the relationship between different types of equipment alarms, gathered by the Alarm Manager tool, to have a better understanding of the impact of a failure on the entire system, thus easing and helping the process of manual implementation of RCA rules. As this tool manages millions of daily alarms, it becomes unfeasible to process them manually, making the application of ML essential. Furthermore, ML algorithms have tremendous capabilities to detect patterns that humans could not, ideally exposing which specific failure causes a series of malfunctions, thus allowing system administrators to only focus their attention on the source problem instead of the multiple consequences.

The ML approach proposed in this project is based on the causality among alarms, instead of their features, and uses the cartesian product of a specific problem, the involved technology, and the manufacturer, to extrapolate the correlations among faults. The results achieved reveal the tremendous potential of this approach and open the road to automatizing the definition of RCA rules, which represents a new vision on how to manage network failures efficiently.

KEYWORDS Root Cause Analysis, Fault Correlation, Machine Learning, Artificial Intelligence, Network Topologies, Rules automation, Alarm Manager

RESUMO

Ao longo dos últimos anos, uma das áreas que mais tem evoluído e estendido a sua utilização para uma infinidade de possibilidades é a Inteligência Artificial (IA). Com a crescente complexidade dos problemas, a resolução humana torna-se impossível, uma vez que a quantidade de informação e padrões que podem ser detectados é limitada, enquanto a IA prospera na dimensão do problema em análise. Além disso, como hoje em dia cada vez mais dispositivos tradicionais são informatizados, um número crescente de elementos está a produzir dados com muitas potenciais aplicações. Consequentemente, encontramo-nos no auge do *Big Data*, onde enormes volumes de dados são gerados, sendo totalmente inviável processá-los e analisá-los manualmente. Esta é uma das razões que tem levado à prosperidade da IA.

Além disso, com a crescente complexidade das topologias de rede, é necessário assegurar o correcto funcionamento de todos os equipamentos, evitando falhas em cascata entre dispositivos, o que pode levar a consequências catastróficas dependendo da sua utilização. Assim, as ferramentas de *Root Cause Analysis* (RCA) tornam-se fundamentais, uma vez que são desenvolvidas para, através de regras estabelecidas pelos seus utilizadores, se aperceberem automaticamente das causas subjacentes quando algum equipamento apresenta anomalias. No entanto, com a crescente complexidade da rede, a definição de regras torna-se exponencialmente mais complicada, uma vez que os pontos possíveis de falha escalam tremendamente.

Neste contexto, enquadrado pelo ambiente de rede e cenários de RCA da Altice Labs, foi definido o principal objectivo deste projecto de investigação. Este objectivo consiste na aplicação de técnicas de *Machine Learning* (ML) para extrapolar a relação entre os diferentes tipos de alarmes dos equipamentos, geridos pela ferramenta *Alarm Manager*, para ter uma melhor compreensão do impacto de uma falha em todo o sistema, facilitando e ajudando assim o processo de implementação manual das regras RCA. Como esta ferramenta gere milhões de alarmes diários, torna-se inviável processá-los manualmente, tornando essencial a aplicação do ML. Além disso, os algoritmos ML têm uma enorme capacidade para detectar padrões que os humanos não conseguem detectar, idealmente expondo quais as falhas específicas que causam uma série de falhas, permitindo assim que os administradores do sistema apenas concentrem a sua atenção no problema de raiz em vez das suas múltiplas consequências.

A abordagem ML proposta neste projecto baseia-se na causalidade entre os alarmes, em vez das suas características, e utiliza o produto cartesiano de um problema específico, da tecnologia envolvida, e do fabricante, para extrapolar as correlações entre falhas. Os resultados alcançados revelam o enorme potencial desta abordagem e abrem o caminho para automatizar a definição de regras RCA, o que representa uma nova visão sobre como gerir eficazmente as falhas da rede.

PALAVRAS-CHAVE Root Cause Analysis, Correlação entre Falhas, Aprendizagem Máquina, Inteligência Artificial, Topologias de Rede, Automatização de Regras, Gestão de Alarmes

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AI	Artificial Intelligence
AUC	Area Under the Curve
ACC	Accuracy
FN	False Negative
FP	False Positive
FPR	False Positive Rate
ML	Machine Learning
PR	Pattern Recognition
RCA	Root Cause Analysis
ROC	Receiver Operating Characteristic
SLA	Service Level Agreement
TN	True Negative
TO	Telecommunications Operator
TP	True Positive
TPR	True Positive Rate
XAI	Explainable Artificial Intelligence

1 INTRODUCTION

Over the last decades, the evolution of the Artificial Intelligence field and the emergence of society into a global network have highlighted a potential and promising relationship between these two areas. While the former depends on a high volume of data for its success and accuracy, the latter is known for its Big Data generation; thus, a perfect symbiosis is envisioned.

Nowadays, telecommunication networks are increasingly more complex, with more elements connected and, consequently, subject to more failures and the generation of more alarms. Moreover, as most of the connected nodes of the network are essential to its proper function, all alarms need to be processed in order to understand the underlying problem and its correction, so that the anomalous device can proceed with its functions. However, as stated before, due to the increase in network complexity and number of devices, it is no longer humanly feasible to analyze all alarms generated, creating significant entropy and malfunction due to the delay in fixing them.

Hence, Root-Cause Analysis (RCA) tools are becoming increasingly more crucial to the efficient and smooth support of operational networks. However, the manual definition of RCA rules is becoming more and more difficult. Not only the increased complexity generates more problems and alarms, but it also makes it impossible to fix them manually. This creates a problem that needs to be solved to maintain the correct functioning of communication platforms needed to support all technical equipment essential to our daily lives. Therefore, as aforementioned, it is imperative to combine with the RCA tools Machine Learning (ML) techniques capable of detecting patterns, which a mere human would not be capable of.

1.1 MOTIVATION AND OBJECTIVES

As a leader in the telecommunications innovation sector in Portugal, Altice Labs understands and operates on network topologies of high complexity with numerous alarms from different sources. Currently, Alarm Manager manages more than 25 million events per day worldwide, making its full resolution based on a human component completely impossible. Solving the problems that arise in a timely manner is also crucial to satisfy the Service Level Agreements (SLAs) established with customers.

Either in scenarios of success (fault correction) or failure (fault is still active or was forcibly terminated), alarm instances are saved in a database. This implies that during the Alarm Manager continuous operation time, a vast amount of data is stored and, currently, not further processed. On the other hand, it is known that ML techniques, although extremely powerful and able to detect patterns invisible to the human eye and brain, require a large volume of data for their success, hence the colossal explosion of ML in research fields involving big data [1]. Thus, it became evident that the Alarm Manager tool would benefit from such a combination, allowing the discovery of relationships between previously unknown failures, providing system administrators with new information about fault relationships and hierarchies, thus easing and speeding up the manual process of RCA rule creation. This would be a considerable contribution to the consistent and correct operation of the tool.

In this context, the main objective of this research project is to apply ML techniques to the Alarm Manager data, to analyze and identify the relationships between failures, in order to assess and extrapolate which faults cause which, thus extracting helpful information from the fault history stored. Furthermore, by extracting information from the developed ML models, manual creation of correlation rules is simplified, since new information about the system is now available, allowing a system administrator to significantly reduce the number of alarms that he has to process manually. Therefore, this project deals with a RCA problem, where the models developed throughout this thesis will be trained on real datasets, made available by a telecommunications operator (TO), with a high volume of real alarms, essential to train and determine the models' pattern detection ability.

The achievement of the proposed objective requires the attainment of multiple partial objectives, such as:

- the study of state of the art in ML for the resolution of RCA problems, thus taking advantage of the scientific knowledge available in this area for a better implementation of the proposed problem resolution;
- the understanding, categorization and pre-processing of the data, as only after its complete interpretation will it be possible to apply feature engineering techniques to extract and enhance all helpful information that data may contain;
- the implementation, validation and testing of the models developed, to ensure that the information and patterns detected by the models are reliable, thus avoiding the creation of incorrect rules, which would further delay manual fault processing.

1.2 DISSERTATION LAYOUT

This dissertation is structured in seven different chapters to better expose all the different strands involved, so the goals described in the previous section can be achieved.

In chapter 1 - Introduction - an overview of the motivation that led to the need and creation of this project is provided. Additionally, the objectives that need to be fulfilled for the success of the project and the structure of the dissertation are mentioned.

In chapter 2 - State of the Art - different solutions and approaches in the literature to identical problems are explained, since before starting the development phase of a project, it is necessary to ascertain the state of the art in the application area. Additionally, some introductory concepts necessary for a complete understanding of the developed project are mentioned.

In chapter 3 - Alarm Manager - is explained the current Altice Labs' product which manages the data to be used, showcasing its operation workflow and all information that categorizes an alarm instance and its corresponding type. The perception of this tool is essential, as it addresses the constitution of each alarm instance, which is the main entity of the project.

In chapter 4 - Data Processing - is explained how the dataset to be used during the project was created from the raw data provided by a TO. Additionally, it also mentions all the processing applied to the data in order to purify it, thus generating the final dataset for the application of ML techniques.

In chapter 5 - Proposed ML Approaches - the primary approach used in the project and its sub-variants are explained in depth. These variants are worth mentioning as they were not a change of direction of the project but a refinement process to extrapolate as much helpful information as possible from the data provided.

In chapter 6 - Results Discussion - all the results obtained during the most intensive phase of model testing are exposed in multiple degrees of granularity. In addition, the system used throughout the development stage is characterized, as it is a factor to be taken into consideration in ML and big data processing fields. The automation process developed, to obtain results for all unique problems and areas of the entire country, thus extrapolating as many relationships and patterns as possible, is also exposed.

In chapter 7 - Conclusion and Future Work- an analysis and retrospective of the objectives proposed in this project and its achievement are presented. Additionally, it is mentioned what contributions in the real world were accomplished and envisioned. Lastly, a debate on the possible improvements towards a robust, individual ML tool is provided.

2

STATE OF THE ART

In order to ground the conceptual and practical decisions of this project, it is first necessary to study the main concepts and state of the art in the involved areas. Therefore, this chapter's purpose is twofold: i) to identify and explain the crucial concepts used throughout the thesis; ii) to analyze state of the art techniques used on similar RCA problems. While the former allows to deepen and expose concepts essential to developing the project at hand, the latter serves to better understand how to tackle the problem and what solutions have already been implemented for similar scenarios.

2.1 BACKGROUND CONCEPTS

Throughout this project, numerous tools, models, metrics, and methods related to the ML field were used. However, for a complete understanding of the dissertation, from the variety of techniques used to the testing phase applied, it is necessary to have certain core notions well established. Thus, several concepts will be explained in this section, from algorithms and ML models to their respective metrics, referred to and used throughout the thesis.

2.1.1 *Algorithmics*

In the ML area, there is a wide range of possible learning algorithms to be used, so it is necessary to understand the problem in analysis and its data to identify better the optimal model for the use case. In total, there are six different types of machine learning algorithms [2]; however, two of these are variants of the four best known and worth of briefly highlighting:

- S*upervised learning*: when the data of the problem under analysis has labels, i.e., it is known for a certain input given to an ML model what is the correct output, a supervised learning algorithm can be used. These labels work as a supervisor of the algorithm as through them the model can correct its learning phase if the labels' predictions are wrong, based on the deviance (error) from its true label. In this way, the model optimizes itself to understand the data better and maximize the labels' correctness. Within Supervised Learning, there are two different paths regarding the type of variable the model has to predict:

  - *Continuous Target Variable*: the output value expressed by the model is, as the name implies, a continuous variable [3]. It has a continuous spectrum of possible values that it can take, and there is a direct relationship between the different values. This algorithm is used for regression problems [4];
  - *Categorical Target Variable*: the target variable is categorical [3], so, unlike continuous, it can take on one of a limited and usually fixed number of possible values. Any problem with labels and where it is desired to predict a categorical variable, i.e., a state, uses this learning algorithm. This algorithm is used for classification problems [5].

- *Unsupervised Learning*: Contrary to the previous algorithm, that is, when the data to be used by the model does not have labels, and the model does not know which output it should reach, we are faced with unsupervised learning. Thus, the algorithm must be able to aggregate information/data according to its characteristics and patterns to extract useful information. It can be used for clustering [6] or association [7] problems;

- *Semi-supervised learning*: while in the previous algorithms the existence of labels was binary: either they existed, or they did not; this algorithm is used when there is an intermediate scenario: labels exist for some inputs, but not all, and the cost of labeling them all is too high. This is possible when the existing labels are categorical, where the success scenario of the algorithm depends on being able, through the existing labels, to identify the patterns of that group/state, thus aggregating the entries that have no inputs in their corresponding groups [8];

- *Reinforcement Learning*: this algorithm is quite different from the previous ones as the label concept does not apply. In this technique, a new concept arises, an actor will perform an action and have a respective reward depending on the chosen action [9]. This algorithm applies this action process continuously and learns according to its reward, aiming to maximize it. This continuous process is based on four simple steps: i) the agent observes the state of the input; ii) according to the received state it makes a decision, thus performing an action; iii) the agent receives a reward or reinforcement according to the action it previously took; iv) a pair (initial state, action performed) on the received reward is stored. During this last step, the algorithm tries to extract knowledge about what just happened so that it can get an equal or better reward in a future similar input.

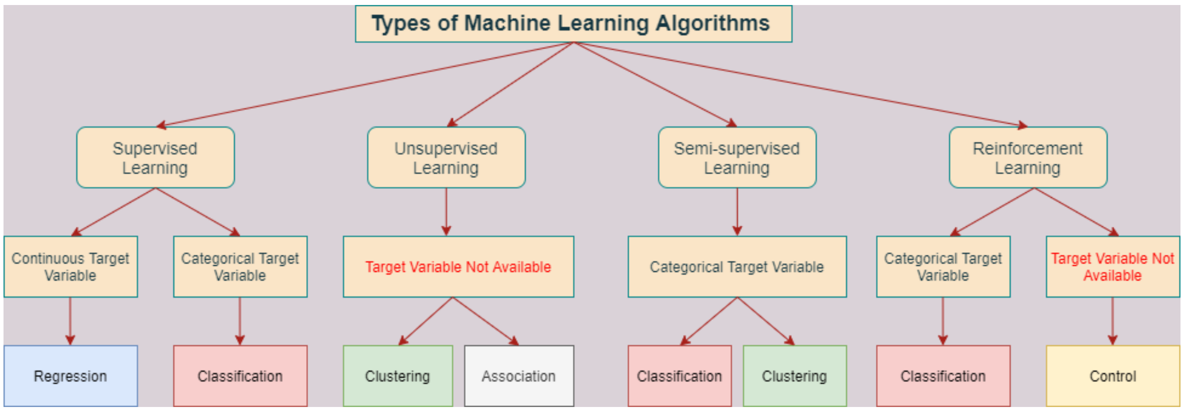

Types of Machine Learning Algorithms
Supervised Learning
Unsupervised Learning
Semi-supervised Learning
Reinforcement Learning
Continuous Target Variable
Categorical Target Variable
Target Variable Not Available
Categorical Target Variable
Categorical Target Variable
Target Variable Not Available
Regression
Classification
Clustering
Association
Classification
Clustering
Classification
Control

Figure 1: Diagram of the four different types of machine learning algorithms (adapted from [10])

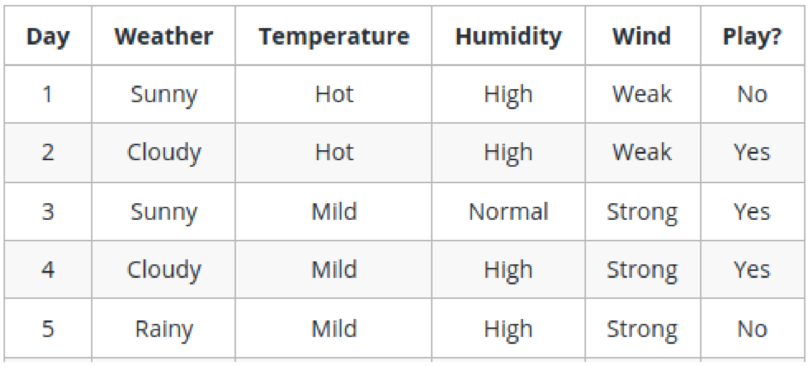
Considering the spectrum of ML algorithm types previously presented, illustrated in Figure 1, and the supervised learning task resulting from the approaches explained in Section 5, the present section also discusses the ML algorithm used to fulfill it. In this sense, the fundaments of Random Forests are further explored in this section, along with how the models' features importance contribute to the analysis.

Random Forests

As an ensemble algorithm, to better understand random forests' principles, it is pivotal to recognize its primary component: decision trees [11]. Random forests comprise a variable number of decision trees (hence being an ensemble algorithm) being the final result the weighted score amongst them (average value or majority voting).

Decision Trees get their name from the tree-like format they use to make decisions [12]. Thus, they are composed of a root, the top node of the entire tree, multiple branches, intermediate nodes, and decisions, the last node/leaf. Its behavior is to distribute features of the dataset across the tree branches so an input can traverse the tree according to the values of its fields, reaching a leaf, i.e., the decision/prediction of the model [13].

To better understand its operation, a brief theoretical example of its workflow will be presented below. Thus, assuming the data shown in Figure 2 as the base dataset for the problem, the implementation of this learning algorithm could lead, since from the same data, it is possible to generate an infinity of different trees, to the decision tree seen in Figure 3.



Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No

Figure 2: Small sample dataset

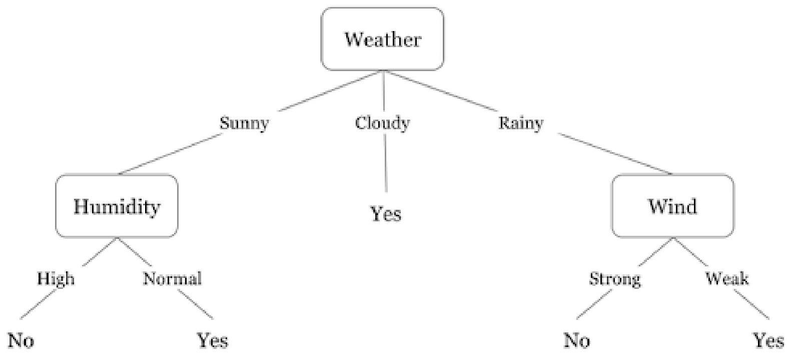


Figure 3: Decision tree generated from the small sample dataset

As can be seen, all nodes except the leaves (nodes at the lowest level) represent variables of the dataset, and the path applied is relative to the value an input has for each field. That is, as an example, if entry number 5 of the dataset in Figure 2 was used for testing, i.e., the model did not know the value of "Play", the path would be as follows: (i) the first feature under analysis is "weather", as for this entry, the value is "Rainy", the path used is the one on the far right. ii) now, the feature under analysis is "wind", as for this entry, the value is "Strong", so now the left path is used iii) the last level of the tree is reached, so the model has reached its decision, in this case: "No". As can be seen in the dataset, the decision obtained by the ML algorithm is correct.

This example is purely theoretical, and as can be seen, only three of the four variables were used as decision nodes. The depth of the tree depends on the number of estimators used. However, these can be repeated; in the above example, after the "Humidity" node, there could be a node for the "weather" feature to filter its value once again.

Having understood how a decision tree works, it is possible to understand the workings and benefits of random forests. These are composed of nothing less than multiple decision trees. While in the last example, only one decision tree was built to forecast, in random forests, several are implemented. So at the end of the process, the model's final forecast is not the one of a single tree, but a weighted forecast of all decision trees [14], as shown in figure 4. Thus, numerous trees are implemented in random forests, all with the same base dataset but with different construction, i.e., different decision nodes.

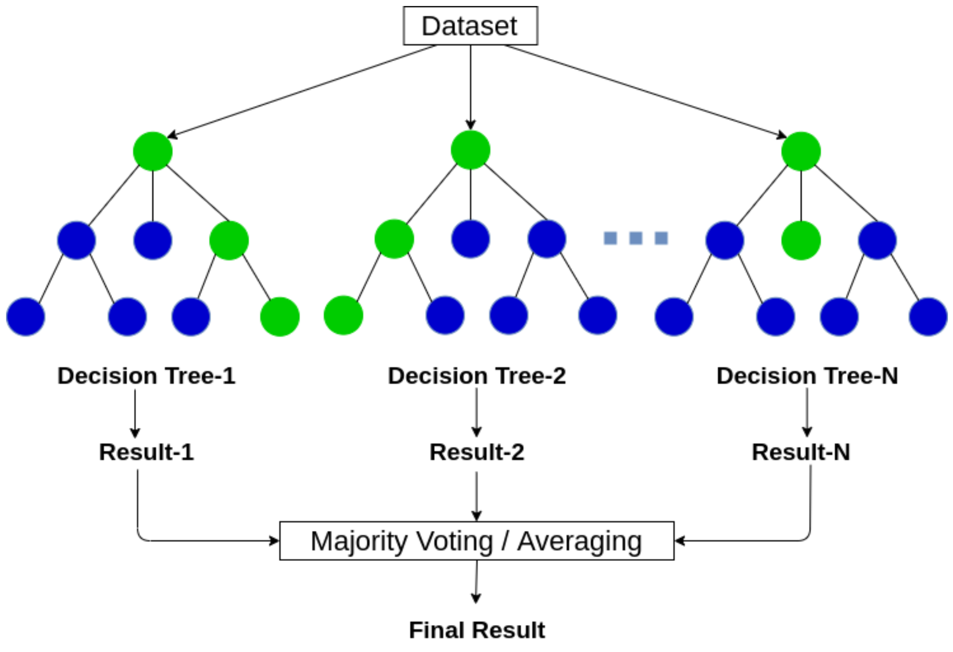
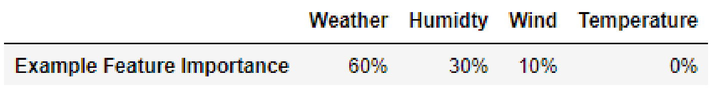


Figure 4: Example of a random forest architecture

Feature Importance

The increasing growth of ML applications in academic, industrial, and health contexts has been triggering the need for explanations regarding the behavior of machine learning algorithms [15]. In this sense, Explainable AI (XAI) is an emerging field [16, 17]. Besides fostering the general public trust in ML [17], studying and assessing the machine learning internal decision criteria can help unveil meaningful novel insights [15] in business/industrial contexts.

In the case of random forests, one can use feature importance as a form of XAI [18]. As the name implies, feature importance allows identifying what impact each feature has on the model's decision-making process. When applying feature importance to a model, an array of size equal to the number of predictors used by the model is obtained, where each field has a percentage value of the impact of the feature on that index. The value for each feature can range from 0%, the feature has no impact on the decision, to 100%, that unique feature has all the decision power over the data (implying that all others have 0%, as the sum of the feature importance array must be 100%). For the decision tree represented in Figure 3, an example of applying feature importance is present in Figure 5.



	Weather	Humidty	Wind	Temperature
Example Feature Importance	60%	30%	10%	0%

Figure 5: Example of the application of Feature Importance

The tree creation process is not random: choosing for the top levels of the tree, the features that minimize entropy [19], it is expected, as seen in the figure above, that the feature with the highest impact is the root of the tree, as being the feature with the highest impact and being the root of the tree, the depth of the tree is minimized as the forecast/leaf is reached faster.

In the context of this project, as will be seen later in this dissertation, the final approach used is based on the ML model of random forests, using the trios of past incidents as features. The goal is to train a model for each unique trio to predict whether a specific series of trios will cause the model's trio. Thus, having a trained model that can process trios from the network and identify whether its trio occurred or not. However, this model's prediction is not the information sought by this project. Thus, this approach aims to have a model trained to analyze the causality of a particular trio and then dissect it to see how the decision was made. That is, what patterns the model discovered to identify the correlations between the trios correctly.

Since the features of each trio model are the other existing trios, the application of Feature Importance allows drawing for each trio what is the impact of all others. Thus, in the successful scenario of training the models, it is then possible to extrapolate the correlations between all different trios, completing the objective proposed in this dissertation.

2.1.2 *Metrics*

Just as critical as the development of an approach is the analysis of its results. However, before analyzing the results obtained by a model, it is mandatory to assess its quality since if the model has not been able to learn effectively, all the results obtained are useless and most likely false. Therefore, this sub-section will explain what a confusion matrix is and the metrics it allows to calculate.

Confusion Matrix

A confusion matrix [3] is a technique used to summarize the performance of classification algorithms [20]. Its size is equal to the number of target classes, i.e., in a binary problem, it will be 2x2, and in multiclass NxN, where N corresponds to the number of values the categorical target variable can take. For the project at hand, since a model will be created for each trio, it only needs to predict whether or not the failure will occur, resulting in many binary problems. Thus, in Table 1, the appearance and content of a models' confusion matrix can be seen.

	Actual Positive	Actual Negative
Predicted Positive	Number of True Positive (TP)	Number of False Positive (FP)
Predicted Negative	Number of False Negative (FN)	Number of True Negative (TN)

Table 1: Contents of a confusion matrix

As shown in the table above, the confusion matrix distributes the model predictions into four different classes [3]:

- *True Positive*: the model predicted the positive class, and it was the actual class. Shown in the upper-left corner (2nd quadrant).
- *False Positive*: the model predicted the positive class; however, it was wrong as the actual class is negative. Shown in the upper-right corner (1st quadrant)
- *False Negative*: the model predicted the negative class; however, it was wrong as the actual class is positive. Shown in the lower-left corner (3rd quadrant)
- *True negative*: the model predicted the negative class, and it was the actual class. Shown in the lower-right corner (4th quadrant)

Having understood the distribution of predictions within the confusion matrix, it becomes evident that a model is as good as the more significant distribution of values is on the $y = -x$ diagonal (correct predictions) and minor on the $y = x$ (wrong predictions).

From the confusion matrix is possible to calculate several metrics, of which is worth highlighting [3]:

- *Accuracy*: represents the ratio between the number of correct predictions over the total number of predictions. This metric answers the question: "How often is the model successful". It can be calculated as follows:
$$Accuracy(ACC) = \frac{TP + TN}{TP + TN + FP + FN}$$
- *Precision*: represents the ratio of the number of correct positive predictions over the total number of positive predictions. This metric answers the question: "Having predicted the positive class, how often is the model successful". It can be calculated as follows:
$$Precision = \frac{TP}{TP + FP}$$
- *True Positive Rate (TPR)*: represents the ratio between the number of correct positive predictions over the total number of positive entries on the dataset. This metric, also known as recall, answers the question: "If the actual class is positive, how often is the model successful". It can be calculated as follows:
$$TPR = \frac{TP}{TP + FN}$$
- *False Positive Rate (FPR)*: represents the ratio between the number of wrong positive predictions over the total number of negative entries on the dataset. This metric answers the question: "If the actual class is negative, how often is the model unsuccessful". It can be calculated as follows:
$$FPR = \frac{FP}{FP + TN}$$
- *ROC-AUC*: a Receiver Operating Characteristic curve (ROC curve) is a metric that reflects how well a classification model performs across all categorization levels. The TPR vs. FPR at various categorization

criteria is plotted, in form of a graph on a ROC curve. As the classification threshold is lowered, more items are classified as positive, increasing both False Positives and True Positives. The Area Under the Curve (AUC) is the term used to describe the area beneath the ROC Curve. The AUC score can vary between 0 and 1, being a 0.5 AUC score associated with a model with random prediction,i.e., that has not captured the data interdependencies. The ideal model reaches a AUC score of 1 whereas a model whose all prediction are contrarily predicted has a 0 AUC score.

Throughout the development, the ROC and ACC metrics will be used to ascertain the quality of a model as they are the metrics with the most significance [21]. Thus, to the confusion matrix presented previously, two new columns are added, as shown in Table 2, so that these two metrics will always be present when evaluating a model's quality.

	Actual Positive	Actual Negative	Metrics	
Predicted Positive	-	-	Accuracy	ROC-AUC
Predicted Negative	-	-	-	-

Table 2: Template of the confusion matrix and metrics considered

2.2 RELATED WORK

The work to be carried out during this project focuses on applying machine learning techniques to root cause analysis, thus being able to extrapolate correlations among multiple failures automatically. Through this newly acquired knowledge about the system, an administrator is able to manually define new RCA rules that can more efficiently manage the countless alarms produced by the system topologies.

Therefore, thorough evaluation of previous studies conducted in the field is paramount for adequate methods, tools and techniques' selection. Through root cause analysis, these past studies aim to identify the relation of causality between multiples interdependent variables, in order to assess, predict and therefore prevent fault occurrence.

Root cause analysis can be applied to a wide range of fields of research, from biological science and medicine [22] to manufacturing engineering [23, 24, 25] and telecommunication industries [26, 27, 28, 29, 30].

Being able to determine the root cause of any given problem is determinant to properly and permanently address and eradicate it. Doing so in business contexts can have substantial impact perceived service quality of its users and consequent business stability.

Although being a widely active field of research, the concrete application of root cause analysis to tracking devices failure in network topology is still an open issue. Nevertheless, data processing pipelines and learning algorithms across multidisciplinary fields can be transposed and adjusted to maximize information retrieval for RCA.

The study carried out in [26] targets a root cause problem with a similar area of application to the one of the present work, however simplified, both in terms of data dimensionality, root problems and number of devices. In this sense, some strategies and techniques presented inspired the strategies used for problem characterization and approach definition for this thesis. In this study, Random Forest models are created for each type of event, in order to predict whether a specific fault on a specific device will occur or not. The analysis correlates all devices faults within a 5 minutes time frame of each occurrence.

Similarly, [31] also resorts to random forests in order to properly detect anomalies in Internet-based services. Through multiple detectors functions, it is possible to extract features from the data, which cooperatively with the operator's labels, allows for a continuous retraining of the anomaly classifier. This approach was able to achieve an accuracy of 66%, which is a reasonable value considering the field of application, that is known for the complex data patterns to recognize.

The potential of applying random forests in root cause analysis fostered the use of a variation of this algorithm, in particular a Random Decision Forest Regressor, to cancer analysis [22].

With a similar goal as [31], the study carried out in [23] applied a different approach based on Bayesian networks for Root Cause Analysis in the field of quality deviation in manufacturing. In this study a Bayesian networks model represented by direct, acyclic graphs, was developed in order to identify the causal relationship between manufacturing stages.

Considering other root cause analysis strategies applied to other fields, [32] also uses Bayesian networks to networking monitoring and security, whereas [33] focuses on predicting service failure resorting to neural networks.

Neural networks are also the core algorithm underneath the [34] study, where root cause analysis aims to determine the failure of power transformers. Although of a divergent field, this type of power failure problem deeply relates with some of the key root problems that are targeted in this thesis project.

The [35] study is also based on neural networks, however focusing more on identifying the correlation of variables more than the root of the problem itself. Complementary, the research method resorted to clustering in an attempt to handle the unsupervised nature of the dataset considered.

Furthermore, within the field of RCA for the telecommunication sector, [30] focuses primarily on alarm correlation discovery, having resorted to a graph-based ML algorithm to help detect fault patterns in order to reduce network maintenance costs. Similarly, [29] also resorts to graph theory, having developed a hybrid causal graph-based algorithm, tested in both artificial data and real telecom data.

Conversely to previously approaches used for RCA, a Principal Component Analysis (PCA) technique is used in [28] for data dimensionality reduction, as well as to, in an unsupervised learning strategy, detect anomalies and the root cause of telecommunication network problems.

Lastly, the study carried out in [27] explores an alternative solution to a simplified version of fault correlation and RCA problems in the telecommunication field. The algorithm used is different from the previously described, where to solve this issue was used generalized sequential patterns, which allow a time-gape constraint, thus being optimal for mining alarm databases.

The study was able to achieve a 70% accuracy in detecting failures. However, the frequency and types of such occurrences have increased over the past few years, as well as the number of devices spread throughout the country. The technological developments in the telecommunication sector over the past few years may also increase the layers of dependencies between devices, and therefore the complexity of tracking the source of the problem.

The specific problem of root cause analysis applied to fault correlation in real network topologies can be perceived and handled differently. Although ultimately the fault analysis can be recognized as a binary classification problem, since the malfunction either occurs or not [26], in the context of the specific problem under assessment, the fault occurrence can be of multiple natures or sources, and therefore, a multiclass analysis can be also adequate [33].

Another factor to take into consideration regarding root cause problems refers to the frequency of fault occurrences. As in most failure and anomaly detection problems, being able to determine the root cause of failure in devices connected in a massive network topology composed of a variety of different equipment is greatly affected by the imbalanced nature of fault occurrences in the datasets.

Resorting to the valuable principles, processing techniques and machine learning algorithms considered in previous works in the field, the current thesis project can focus on devising a strategy and pipeline to adequately and accurately identify the root cause for fault and malfunction in a real world network context.

2.3 SUMMARY

In this chapter, various approaches used in problems identical to the one outlined for this project have been exposed. The analysis of these allowed to conceive a plan of the best way to tackle this problem, thus speeding up the following development stage.

Additionally, concepts regarding the types of ML algorithms and their respective metrics were also exposed, which are essential for the complete understanding of the thesis.

3 ALARM MANAGER

Due to the increasing complexity of network topologies, RCA tools have become vital for their correct operation. Nevertheless, network knowledge is required to manually define rules necessary for alarm aggregation and correction, which is increasingly complex due to the growing number of devices in the network. Thus, as mentioned before, in this project, ML techniques will be applied to extrapolate new information from the network, helping and improving the subsequent rule creation process.

In this case study, the RCA tool used and developed by Altice Labs is Alarm Manager [36]. This is the tool responsible for generating and storing all the alarm instances related to the failure of its customers' equipment. So, before going into more details regarding the project development, it is necessary to properly understand how Alarm Manager works, its workflow and what data is stored for each alarm instance.

3.1 DESCRIPTION

Alarm Manager is the product of the NOSSIS suite responsible for the Fault Management area, which concentrates the reception of alarms from the network, storage, treatment, monitoring and correlation of faults. It carries out the management of states, severity levels and other standardised alarm parameters. It allows defining the cycle of all alarms created in the system and monitors the processing infrastructure, as well as the various collection channels.

Built on the experience of Altice Labs and its customers in the area of Telecommunications, it guarantees fault monitoring, problem detection and the opening of fault tickets, streamlining processes and making them more efficient. This translates into operational gains and an expeditious adjustment to the business through flexibility the introduction of the management of new platforms and/or network elements.

3.2 FUNCTIONING

As the name indicates, the Alarm Manager controls the alarms of numerous instances of equipment, so if any problem arises, it is quickly reported and corrected. For multiple reasons, such as good practice and data persistence, no alarm is changed after its emission. Instead, whenever necessary, a new alarm is issued with new or updated information on the same malfunction. All alarms have four different types of events, regarding its current point of situation, so they can be: *NEW*, the moment an alarm is created; *CHANGE* whenever any information has to be updated or added; *CLEAR* the moment the problem is solved and the equipment can register its back to the desired state issuing the end of that alarm instance; *PURGE* where instead of the equipment detecting the problem is solved, the alarm instance is manually closed. The life cycle of a common problem generated on the network, accompanied by images of the tool's documentation [37], is as follows: at an arbitrary moment, a piece of equipment starts to malfunction, thus initiating the "point of failure" - the instant when the equipment detects that it is malfunctining and issues the alarm, as can be seen in Figure 6 on the first timeline.

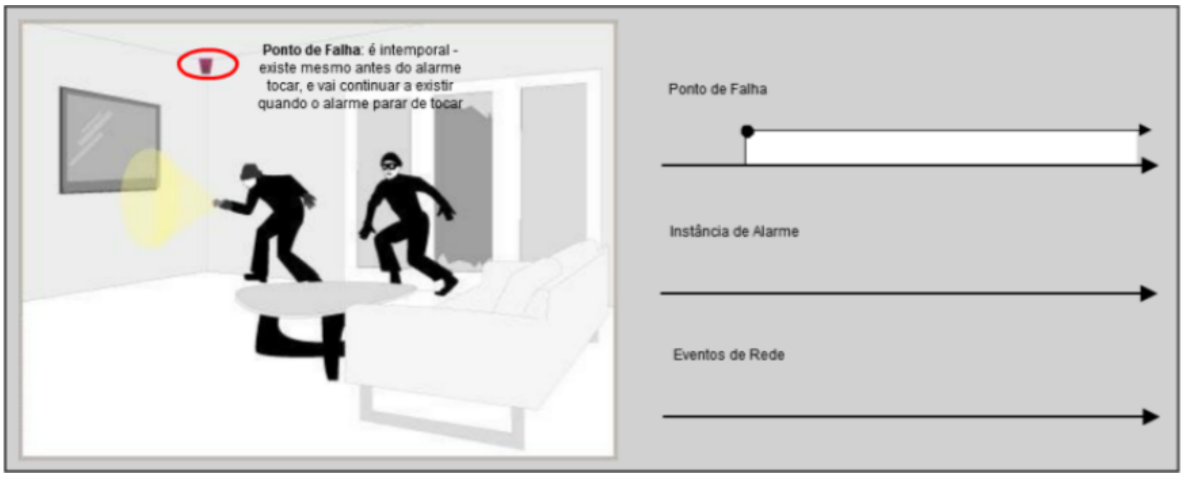

Ponto de Falha: é intemporal - existe mesmo antes do alarme tocar, e vai continuar a existir quando o alarme parar de tocar
Ponto de Falha
Instância de Alarme
Eventos de Rede

Figure 6: Issue of detection of a problem by network equipment [37]

Thereafter, the alarm is received by the Alarm manager servers, which immediately create an "Alarm Instance", as can be seen in Figure 7 on the second timeline. This instance identifies which equipment has detected irregular behaviour and needs to be fixed and generates a network's event, as seen in the figure's third timeline.

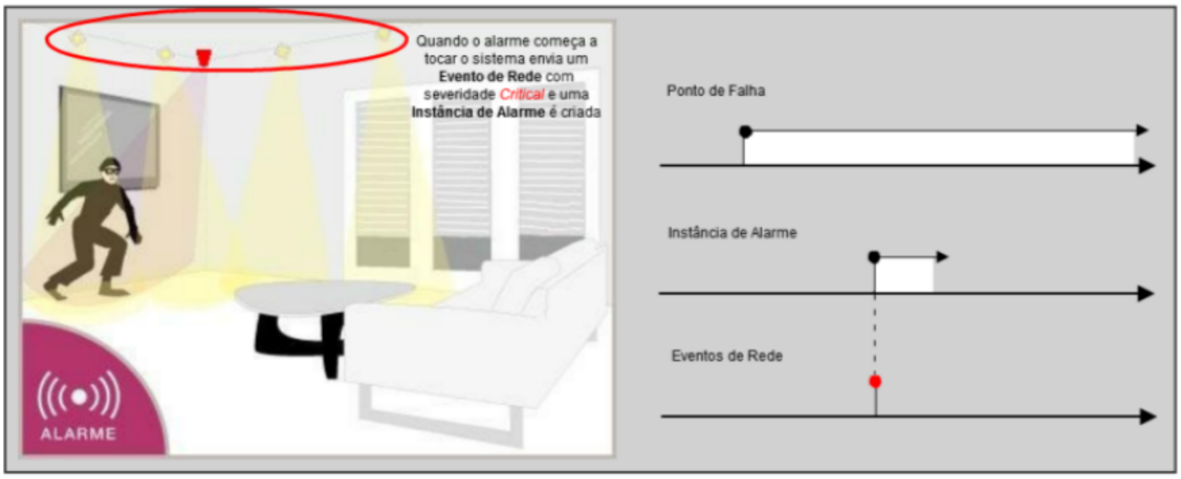

Quando o alarme começa a tocar o sistema envia um Evento de Rede com severidade Critical e uma Instância de Alarme é criada
ALARME
Ponto de Falha
Instância de Alarme
Eventos de Rede

Figure 7: System creates an alarm instance entry with type NEW ans severity Critical [37]

This alarm instance persists in the network until its resolution or manual closing by a system administrator. In both cases, a network's event is generated, as can be seen in Figure 8 on the third timeline, identifying the resolution of the problem but not yet the termination of the instance. Then, after a pre-defined guard time, in order to ensure that the equipment has not reported any more failures, the instance can be terminated, and the problem is considered solved, as can be seen on the figure's second timeline.

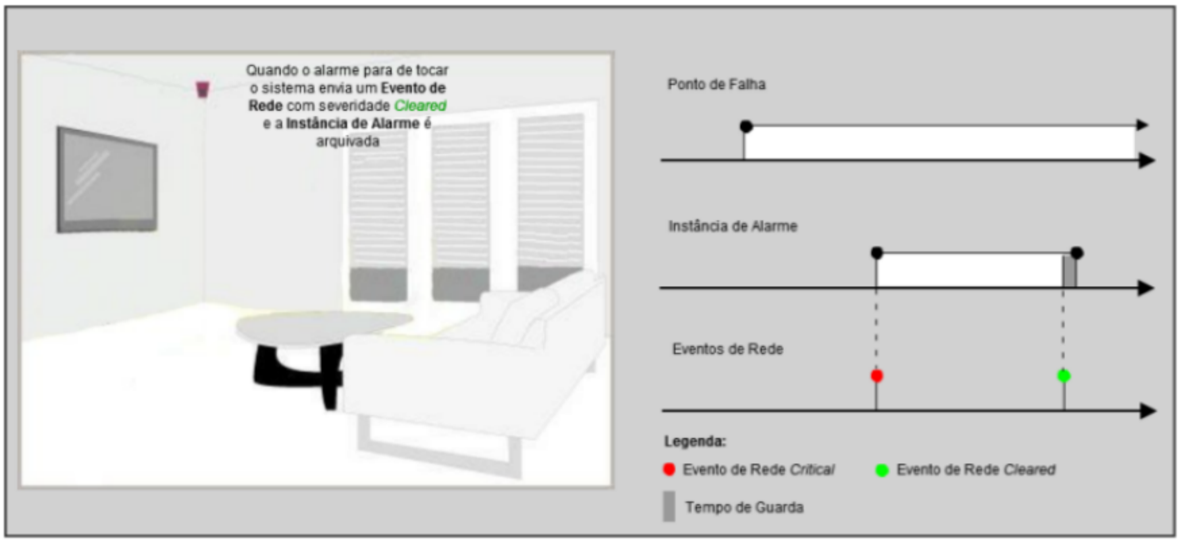

Quando o alarme para de tocar o sistema envia um Evento de Rede com severidade Cleared e a Instância de Alarme é arquivada
Ponto de Falha
Instância de Alarme
Eventos de Rede
Legenda:
Evento de Rede Critical
Evento de Rede Cleared
Tempo de Guarda

Figure 8: Reception of a CLEARED alarm for the same instance, referring the problem is no longer active [37]

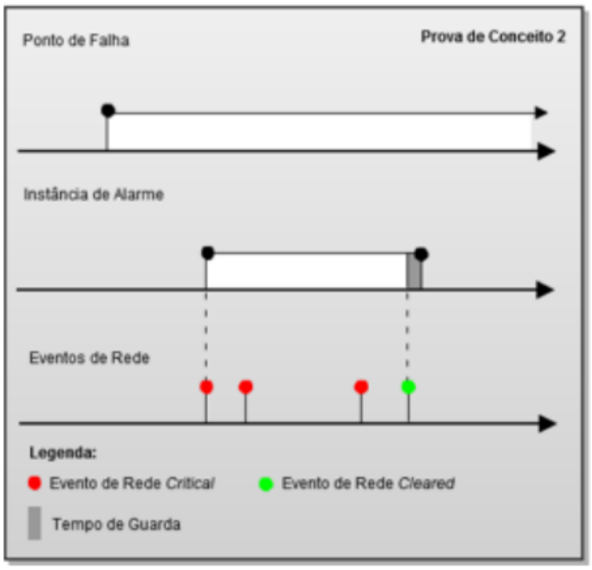
However, the alarm instance's flow is not always so straightforward, so scenarios with different behaviors than described above may arise. Since it is essential to understand the possible behaviors of the system, for better modeling of the problem, some common alternative scenarios will be exposed, also documented in the Alarm Manager's manual as proof of concept.

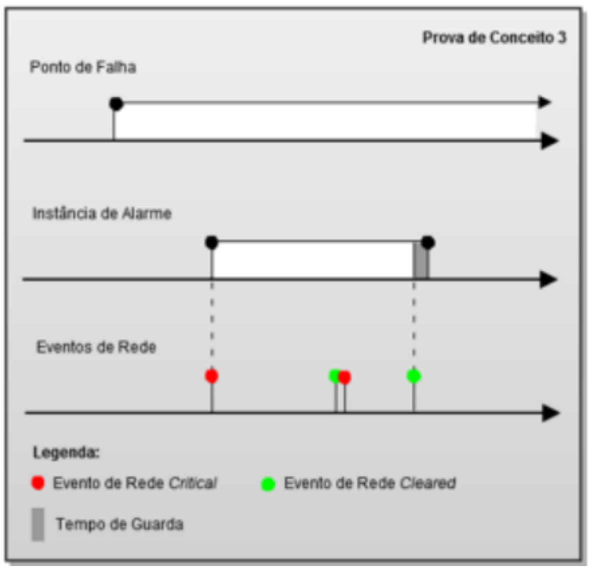
- More than one alarm about the same malfunction is issued before CLEARED. This can be multiple NEW's, occurring from synchronization problems between AM's servers, or multiple CHANGE's if the alarm needed to update or add any new info to better detail the problem. In this scenario, the same alarm instance will have multiple entries besides the default/minimum 2. Such behavior can be seen in Figure 9a.

- As stated before, all alarms have a guard period that needs to be complied with after the reception of the CLEARED event in order for its alarm instance to be closed. If after the system receive a CLEARED, it receives another critical event, before the guard time is over, the CLEARED event is ignored, and such instance remains active, needing another CLEARED or PURGE to terminate the instance. Such behavior can be seen in Figure 9b.

- After an alarm instance is closed, either by failure correction or being forced by a system administrator, if another problem, same as before, occurs in the same device, another alarm instance is open. So, no matter what may happen in the future, when an alarm instance is closed, it cannot be opened again. If a malfunction is repeated, another alarm instance will be created, thus exposing the problem again, without affecting the previous one that has already been resolved. Such behavior can be seen in Figure 9c.

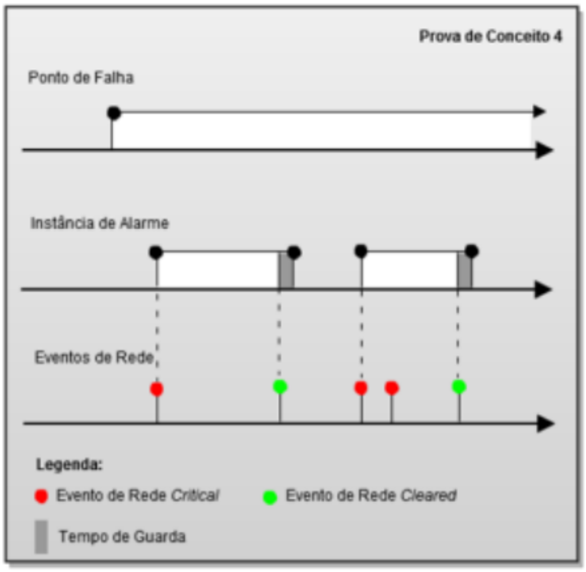
- Sometimes, due to network issues, the event CLEARED may appear before the NEW alarm. In this scenario, the instance is created at the reception of CLEARED and only closed when another CLEARED alarm is received and guard time is over, thus avoiding its termination in an error situation. Such behavior can be seen in Figure 9d.

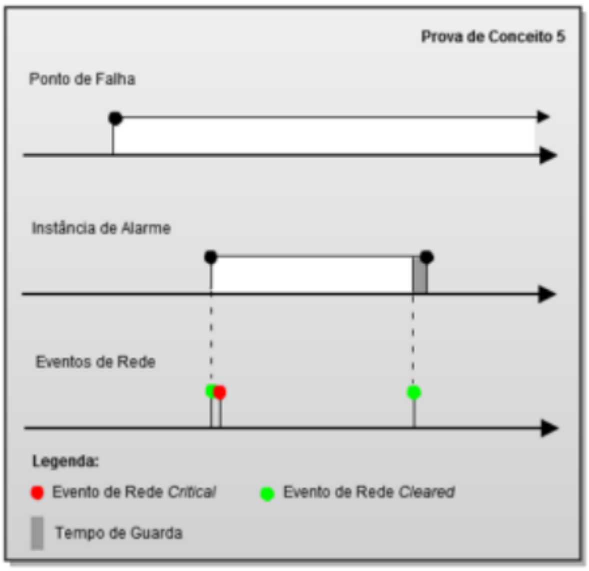




(a) Reception of multiple alarms from the same instance, before CLEARED

(b) Reception of a new alarm from the same instance within the guard period





(c) Successful closure of an alarm instance and re-opening of an identical one after receiving a new alarm for the same issue

(d) Creation of an alarm instance at reception of a CLEARED alarm, instead of the alarm that would trigger the instance creation

Figure 9: Alternative behaviors to the default flow of an alarm instance (adapted from [37])

3.3 ALARM DATA

Since the alarm is the primary entity of the alarm manager, it is composed of a myriad of fields to describe all the necessary information. Since for the project at hand, many of these fields do not have any helpful information, it is necessary to initially ascertain what each one represents for a later pre-processing and field removal phase. The following table 3 shows all the features of an alarm saved by the Alarm Manager, as well as the meaning of each one, which is essential to the perception of the problem and its resolution and the possible range of values that each can take.

Feature	Description	Range of Values / Type
Id	unique identifier of an alarm	$\mathbb{N}_{>0}$
eventType	Describes the event/state of the alarm	New / Change / Clear / Purge
eventCount	How many occurrences of the alarm has entered the system, only increases when aiState opens after closing	$\mathbb{N}_{>0}$
totalEventCount	How many occurrences of the alarm has entered the system, increases with every event type	$\mathbb{N}_{>0}$
systemCreationTime	Creation time of the alarm instance, measured by the Alarm Manager clock	$\mathbb{N}_{>0}$
systemLastUpdateTime	Time of the last alarm input, irrelevant of the event type, measured by the Alarm Manager clock	EqpA
startTime	Creation time of the alarm instance, measured by the equipment clock	$\mathbb{N}_{>0}$
lastStartTime	Time of the last alarm input, irrelevant of the event type, measured by the equipment clock	$\mathbb{N}_{>0}$
systemAckTime	Acknowledgement time by system user	Nan / $\mathbb{N}_{>0}$
ackSystemId	System user's Team responsible for the acknowledgement	None / Maestro
ackUserID	System User's Unique Identifier responsible for the acknowledgement	None / Maestro
severity	Degree of severity of the alarm	Warning / Minor / Major / Critical / Indeterminate
urgency	Degree of urgency of the alarm, similar to severity	Warning / Minor / Major / Critical / Indeterminate
aiState	State of the alarm instance	Open / Close
ackStatus	Whether the alarm was acknowledge or not	Acknowledged / Unacknowledged
additionalText	Additional alarm information, too disparate to be set in a static field	XML
tags	Extra information	String
variableSpecificProblem	Extra Information about the problem identified	String
inhibitted	Flag added by AM to whether it is displayed or not in the network	True / False
manualCloseUserId	System User's Unique Identifier responsible for the alarm instance manually closure	None
manualCloseSystemId	System user's Team responsible for the alarm instance manually closure	None
manualCloseTime	Moment the alarm instance was manually closed	None
notVisible	Whether the alarm is displayed or not in AM	True / False
subsystem	Supplier of the equipment that generates the alarm	String
summary	A brief summary of the alarm - virtual field	String
managedObjectClass	Type of equipment where the failure occurred	String
managedObjectInstance	Specific equipment where the failure occurred	String
localCode	Location of the equipment where the failure occurred	String
node	No information available about this field	Not applicable
nodeAlias	No information available about this field	Not applicable
alarmType	Deprecated feature, no longer used	String
probableCause	Deprecated feature, no longer used	String
probeHost	AM server that received the alarm	String
probeDirectory	Deprecated feature, no longer used	String
extraAttributes	Additional alarm attributes, too disparate to be set in a static field	String
anomaly	Anomaly detected in the alarm, identical anomalies between alarms indicate causality	String
correlatedId	Unique Identifier of the alarm which caused this alarm to occur	$\mathbb{N}_{>0}$
domain	Technology used by the equipment	String
inMaintenance	Whether the equipment is under maintenance or not	True / False
moiFriendlyName	Alias of the specific equipmet	String
technology	Technology used by the equipment, same as domain	String
inToeOrFlap	Indicator if alarm is occurring excessively, causing spam	True / False
sync	Only false alarm are to be considered	True / False

Table 3: All native features from an alarm instance

3.4 DATA BALANCING

For the problem under analysis in this dissertation, the correlation between failures, the most relevant factor in the data is the one that identifies the failure that occurred: specificProblem. The main goal will be to analyze the relationship between different specificProblems, so, in an initial phase, it is helpful to ascertain the distributions of the problems to analyze their balancing.

Before analyzing the distribution of problems in the area with the most occurrences, it is necessary to analyze the distribution of problems in all country zones. Thus, in the interval from May 1st to May 16th, as shown in Figure 10, it was identified for each country's zone: i) how many unique specificProblems were detected (left); ii) how many failures occurred during the period in question (right).

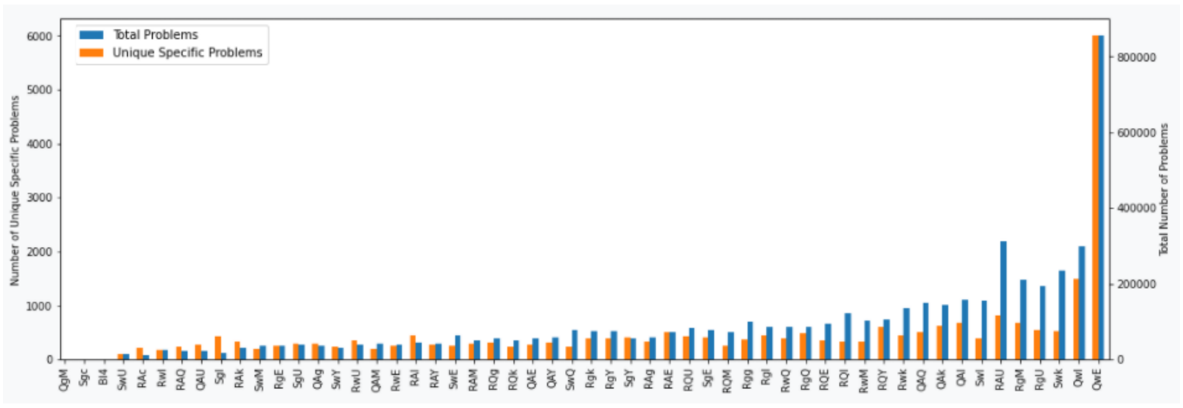


Figure 10: Distribution of the faults across the various zones of the country.

Although there are 55 different zones in the country, only 54 are present in the figure above, as one of the zones, "QwA", due to a system error, has 539396 unique problems in 775316 entries. As can be seen, the distribution is not at all homogeneous, considering that the most recurrent country zone, "QwE", has about 23.5% of all failures in the country. Thus, this will be the zone used henceforth for testing.

Using a dataset sample, relative to the period between May 1st and May 16th, consisting of 2.77 million entries, which, after a filtering process, which will be explained in the next chapter, and the removal of all non-New entries, remains with about 670 thousand entries. Figure 11 shows the number of occurrences of each problem in the top20 and the cumulative percentage of problems in the dataset (i.e., the sum of previous problems with itself). The orange bars represent the number of occurrences (with its scale at the left) of the specific problem of the X-axis, which, as can be seen, is not constant among all problems but significantly decreasing (thus showing a notorious unbalance among the different problems). In blue is represented the cumulative value of the problems (with its scale at the right), i.e., the aggregate sum of the occurrences, thus representing the "analyzed" percentage of the dataset (within the 670k).

As shown in the figure, the distribution is not homogeneous (where the most recurrent problem constitutes almost 21% of the entire sample dataset), and still within the top30, the 70% barrier is quickly surpassed. Although in this sample there are 4036 different problems, as we can see in Figure 12, most are pretty insignificant, since the first 133 problems represent 90% of the dataset, and the top 1191 (which is not even a third of the total number of problems) composes 99% of the dataset's entries.

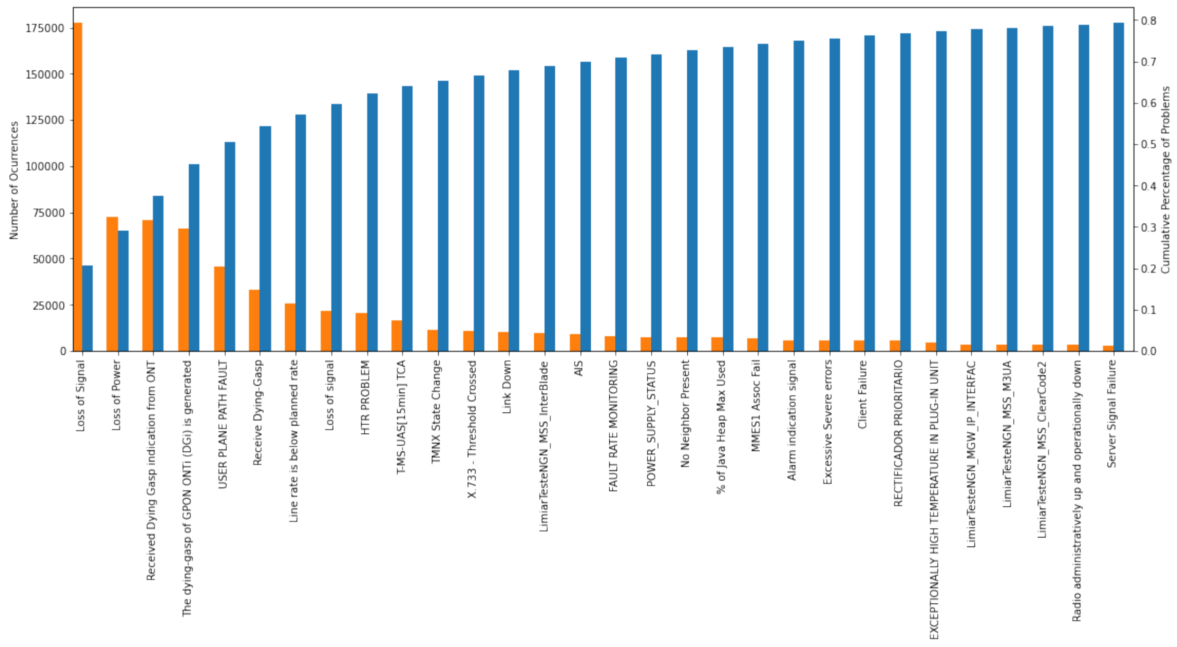


Figure 11: Occurrence distribution and cumulative percentage of the top20 problems in the sample dataset

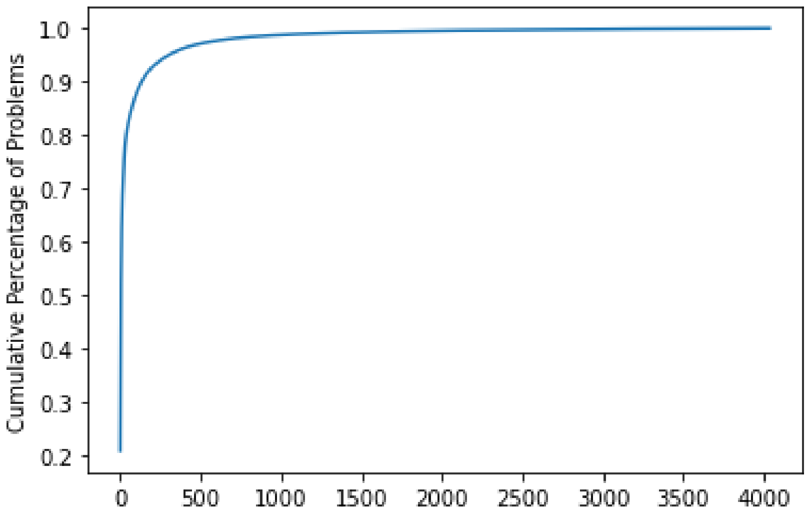


Figure 12: Cumulative percentage of all problems in the sample dataset

3.5 SUMMARY

In this chapter, the motivation for understanding the Alarm Manager tool has been exposed. Thus, it was explained in detail what it is and what it is used for, its typical workflow and behavior in more anomalous scenarios, the data stored for each alarm instance created, and, finally, the distribution of the different types of problems in an arbitrary period of time.

Having properly understood the tool's behavior and data, which led to this project's origin, it becomes possible to move on to a pre-processing data phase, thus transforming the raw data exposed in this chapter into helpful information that an ML model can process.

4

DATA PROCESSING

As the dataset to be used for the development of this project, as mentioned before, is the actual output of Alarm Manager, it has an enormous variety of fields and information that is not necessary nor useful for this project. So, for the application of machine learning techniques, it is necessary to understand what the goal is so that it is viable to apply data processing techniques, thus extracting as much helpful information as possible. Therefore, in this chapter, all techniques, and processing applied to the original dataset will be addressed, thus creating a base dataset used to train the ML models, ergo achieving the stipulated project goal.

4.1 DATASET IMPORT

Although the data to be used is, as mentioned above, the actual output of Alarm Manager, it is not aggregated in a single file but a vast set of parquet files, with several for each day. In order to facilitate the application of ML techniques in python, it is quite helpful to read/convert the data to a dataframe, thus allowing a more agile use [38]. On a first impression, the solution for merging the files seems relatively trivial: i) reading the parquet files to dataframe; ii) recursive application of the append method [39], concatenating the dataframes into a single one. However, although dataframes are mutable [40], their length is unchangeable after creation, so the append method does not add rows to a dataframe; instead, it returns the desired concatenated dataframe [39].

Due to this property, the linear concatenation of dataframes becomes quite heavy and slow, as it has to continuously rewrite the same lines along the concatenation of the numerous dataframes in the same variable. To mitigate this issue, a concatenation mechanism based on the divide and conquer algorithm, similar to the one used on quicksort [41], was implemented, working as follows: recursively, divide the number of files to be imported by half, until the moment when there are only 2 or 1 in the case of the division of an odd number of files; concatenate those two in an auxiliary dataframe; recursively repeat this process upwards, thus concatenating the auxiliary dataframes of the previous iteration into a new dataframe. In this way, the number of lines repeated per append is drastically reduced. This concatenation process can be seen in the diagram of Figure 13. For the initial development of this project, since it has no use until the final solution is found to use the entire dataset available, it was imported 64 parquet files that represented 15 days of issues (same dataset sample used to analyze the problems' distribution in Section 3.4).

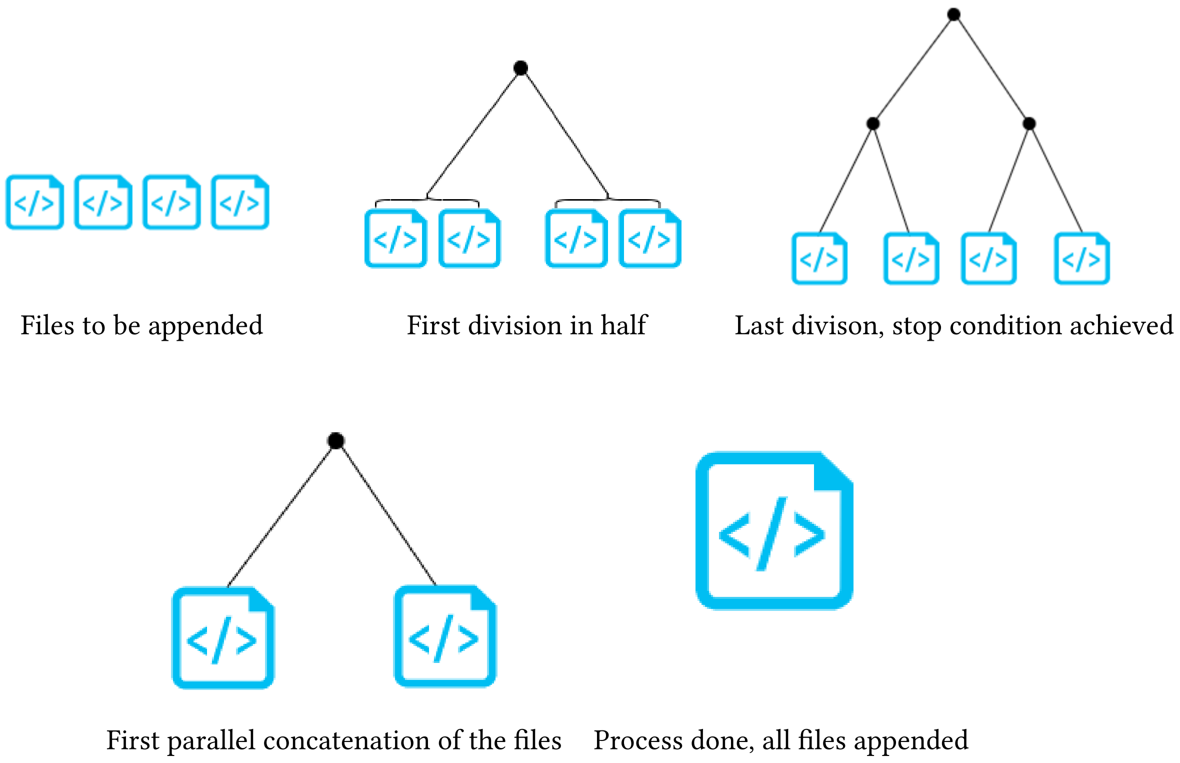
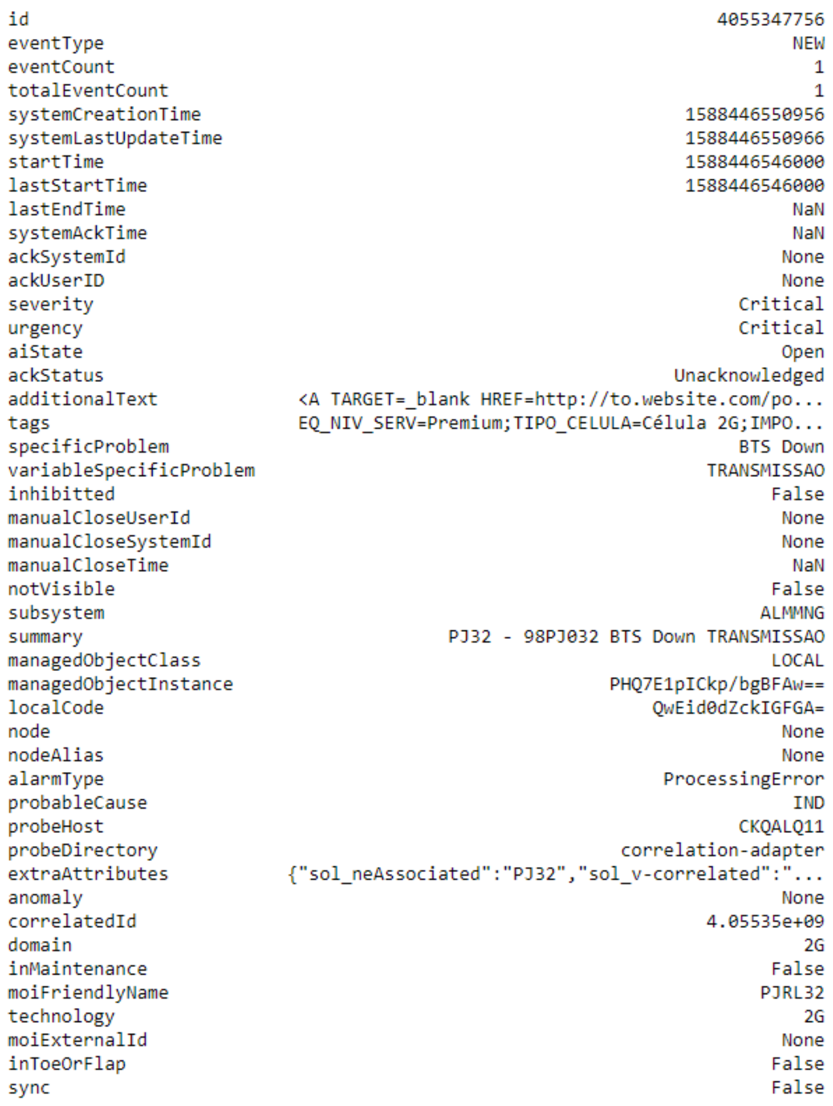

Files to be appended
First division in half
Last divison, stop condition achieved
First parallel concatenation of the files
Process done, all files appended

Figure 13: Parallel concatenation process developed

4.2 ADDITION AND REMOVAL OF FEATURES

At this point, a dataset corresponding to 2 weeks of Alarm Manager entries is created, thus forming a dataframe with millions of lines, where each row has the information represented in table 3, as can be seen in Figure 15. However, as seen in the table and figure, there is a lot of redundant, unnecessary and deprecated features, so it is important to remove them from the dataset, leaving only useful information for the implementation stage. The following columns have been removed for such purpose: startTime, lastStartTime, lastEndTime, systemAckTime, ackSystemId, ackUserID, ackUserID, ackStatus, inhibited, manualCloseUserId, manualCloseSystemId, manualCloseTime, notVisible, summary, node, nodeAlias, alarmType, probableCause, domain, moiFriendlyName, tags and additionalText.



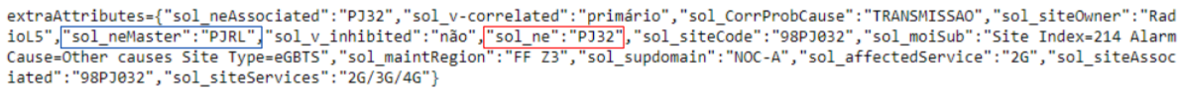
```
id                                                          4055347756
eventType                                                          NEW
eventCount                                                           1
totalEventCount                                                      1
systemCreationTime                                       1588446550956
systemLastUpdateTime                                     1588446550966
startTime                                                1588446546000
lastStartTime                                            1588446546000
lastEndTime                                                        NaN
systemAckTime                                                      NaN
ackSystemId                                                       None
ackUserID                                                         None
severity                                                      Critical
urgency                                                       Critical
aiState                                                           Open
ackStatus                                               Unacknowledged
additionalText           <A TARGET=_blank HREF=http://to.website.com/po...
tags                     EQ_NIV_SERV=Premium;TIPO_CELULA=Célula 2G;IMPO...
specificProblem                                               BTS Down
variableSpecificProblem                                     TRANSMISSAO
inhibitted                                                       False
manualCloseUserId                                                 None
manualCloseSystemId                                               None
manualCloseTime                                                    NaN
notVisible                                                       False
subsystem                                                       ALMMNG
summary                             PJ32 - 98PJ032 BTS Down TRANSMISSAO
managedObjectClass                                               LOCAL
managedObjectInstance                             PHQ7E1pICkp/bgBFAw==
localCode                                             QwEid0dZckIGFGA=
node                                                              None
nodeAlias                                                         None
alarmType                                              ProcessingError
probableCause                                                      IND
probeHost                                                     CKQALQ11
probeDirectory                                     correlation-adapter
extraAttributes         {"sol_neAssociated":"PJ32","sol_v-correlated":"...
anomaly                                                           None
correlatedId                                               4.05535e+09
domain                                                              2G
inMaintenance                                                    False
moiFriendlyName                                                 PJRL32
technology                                                          2G
moiExternalId                                                     None
inToeOrFlap                                                      False
sync                                                             False
```

Figure 14: Alarm's features before preprocessing

The "extraAttributes" feature, being a JSON field as can be seen in Figure 15, may at first glance seem useless since ML models cannot process it. However, it has two information fields that are very relevant to the project at hand:

- sol_ne - identifies, for the equipment that reported the failure, which equipment is hierarchically above it, thus being its aggregator (1 level above). In Figure 15, this field is outlined by the red rectangle;
- sol_neMaster - identifies, for the equipment that reported the failure, which equipment is hierarchically above its aggregator, being thus the aggregator of the aggregator of the equipment that malfunctioned (2 levels above). In Figure 15, this field is outlined by the blue rectangle.

Thus, two new fields are created: "moiExternalId" and "moiMaster". This information can be quite relevant as it demonstrates a hierarchy between components, which can be helpful to solve RCA problems where several pieces of equipment fail because their aggregator fails. Regarding "moiExternalId", this field already existed; however, its information was not consistent with the one presented in extraAttributes, and sometimes the field appeared null while the corresponding information was present in the JSON field. Hence, its creation is mentioned because it was removed and later added through iteration throughout the dataframe to obtain the new values for this field due to inconsistencies with the previous data. To optimize this process, since iterating through massive dataframes is an exhausting process, a lambda expression [42] was used, which updated the two new fields created with the information present in the extraAttributes. Therefore, the feature "extraAttributes" can be removed at the end of this process, as it no longer has valuable information for the project.



```
extraAttributes={"sol_neAssociated":"PJ32","sol_v-correlated":"primário","sol_CorrProbCause":"TRANSMISSAO","sol_siteOwner":"Rad
ioL5","sol_neMaster":"PJRL","sol_v_inhibited":"não","sol_ne":"PJ32","sol_siteCode":"98PJ032","sol_moiSub":"Site Index=214 Alarm
Cause=Other causes Site Type=eGBTS","sol_maintRegion":"FF Z3","sol_supdomain":"NOC-A","sol_affectedService":"2G","sol_siteAssoc
iated":"98PJ032","sol_siteServices":"2G/3G/4G"}
```

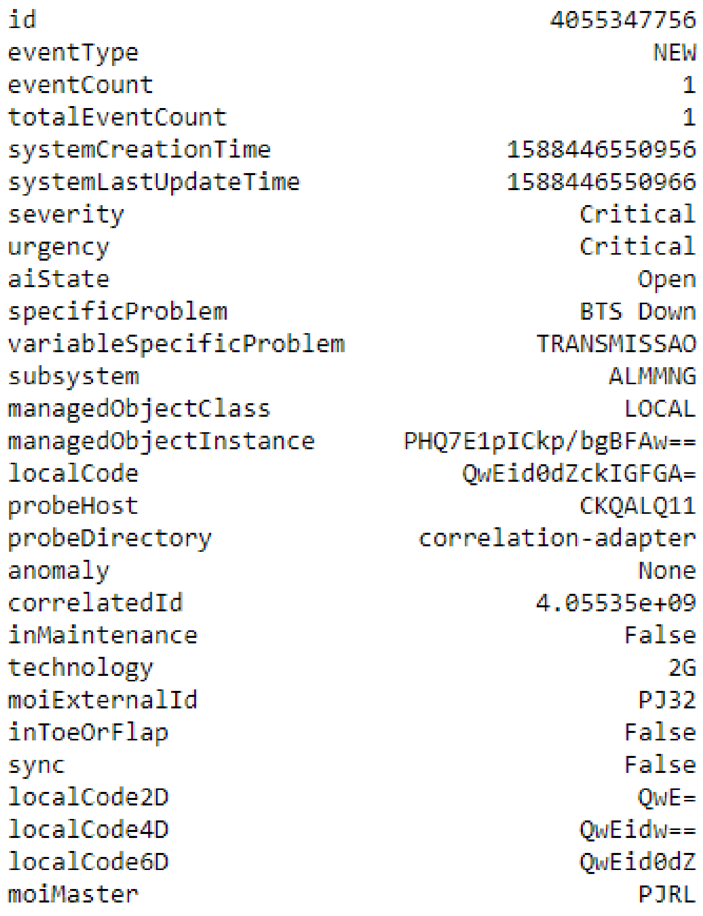
Figure 15: JSON Content of the Feature "extraAttributes"

For data security and privacy reasons, in the datasets provided by a TO, some of the fields have undergone an encryption process, thus maintaining due anonymity regarding critical information from its customers. Such fields were, for example, the equipment id that generated the alarm and its location. While the id being encrypted has no impact on the project development, since this encryption is consistent between all the inputs, the location encryption prevents the larger scale perception of where the failure occurred. For example, if two problems occur in the same street, which is the granularity of the location feature, it is possible to understand the spatial importance of this malfunction, but it will be unnoticeable if they occur in the same block as there is no relationship between the encrypted addresses. Ergo the need to create fields that represented spatial proximity between encrypted addresses arose. So, three new fields were additionally provided by the TO: "localCode2D", "localCode4D" and "localCode6D"; where the higher the number, the greater the granularity of the area represented by that field, thus creating three new per zone aggregation levels for solving the problem. After applying the process aforementioned, an alarm input is represented by the features that can be seen in Figure 16.

4.3 DATASET PURIFICATION

As the dataset has the necessary and valuable features for the project's development, its content must be verified to analyze its quality since a ML model is only as good as the data used for its training. For this purpose, several operations were performed to the dataset to become more consistent, and it was possible to get more critical information from it.

Initially, all "NEW" entries with an eventCount greater than '1' were removed since these result from synchronization issues between Alarm Manager's servers. This is a simple dataframe drop operation, where the referred condition is verified. Additionally, all repeated entries have been removed, as these do not add any extra information to the dataset. Besides the trivial drop duplicates operation, it was necessary to remove repetitions that had different times, which for the event type 'CHANGE' is impossible to distinguish, as they could be effectively different alarms. However, it is relatively easy for the event type 'NEW', as there should only be one entry. Finally, all the inputs of an alarm instance that did not have the event type 'NEW' were removed since that instance



```
id                                       4055347756
eventType                                       NEW
eventCount                                        1
totalEventCount                                   1
systemCreationTime                    1588446550956
systemLastUpdateTime                  1588446550966
severity                                   Critical
urgency                                    Critical
aiState                                        Open
specificProblem                            BTS Down
variableSpecificProblem                 TRANSMISSAO
subsystem                                    ALMMNG
managedObjectClass                            LOCAL
managedObjectInstance      PHQ7E1pICkp/bgBFAw==
localCode                          QwEid0dZckIGFGA=
probeHost                                  CKQALQ11
probeDirectory                  correlation-adapter
anomaly                                        None
correlatedId                            4.05535e+09
inMaintenance                                 False
technology                                       2G
moiExternalId                                  PJ32
inToeOrFlap                                   False
sync                                          False
localCode2D                                    QwE=
localCode4D                                QwEidw==
localCode6D                                QwEid0dZ
moiMaster                                      PJRL
```

Figure 16: Alarm's features after preprocessing

would be biased if the temporal start of the malfunction were unknown. For this removal, it was detected that if it were done through the iteration of a list with the ids to be removed, the process would be highly time-consuming (over eight hours). So, the solution applied was based on calculating the list with all the rows to be removed beforehand, thus applying the drop method only once, which speeded up the process massively, taking only seconds instead.

4.4 SUMMARY

This chapter explains in detail the pre-processing applied to the datasets provided by a TO in order to convert their raw data into usable datasets for ML models.

Due to the Alarm Manager's daily splitting of the data, the import process implemented to minimize the number of dataframes copies and the time to import them was also explained. Finally, a data cleansing process was also exposed, where inputs originating from entropy and Alarm Manager Servers' errors are discarded to maximize the usefulness of the data for the models.

5 PROPOSED ML APPROACHES

Root cause analysis, which aims at understanding and detecting the underlying failure (root cause) in a cascade of its consequent events, is a subfield of the broader area of Pattern recognition. In the past years, pattern recognition (PR) has presented immense growth due to the usage of Machine Learning techniques [43], which are capable of consuming enormous amounts of data (big data) and consistently identify patterns within themselves. For this project, the approach mentioned in [26], seemed an excellent starting point since, although simplified, it has similarities to the challenge proposed here. Furthermore, this article uses Random Forests that shine when using many features that individually have weak predictive power but collectively a much stronger one.

As with many projects in the AI-ML field, the most significant difficulty is not adjacent to the technology stack used, being this knowledge the default premise for development, but rather to how to approach the problem and critically analyze the results. A common misconception in AI is that it only needs substantial data to apply ML techniques. This reasoning is fallacious as it is necessary to have data with helpful information and minimally standardized so that a model can detect patterns and extrapolate information. However, in this use case, it is known *a priori* that there is a causal relationship between the alarms, even if it is not theoretically or empirically known. Thus, a very high standard is set, as now, as hard as it may be, the patterns and information that this project is intended to identify are known to exist.

While in several areas of computer science, one can consider the development process deterministic: the same operation with the same inputs will always lead to the same result; in ML, this does not occur. This requires a deeper understanding of the model implemented and, above all, of the results obtained. Only then is it possible to understand what is happening within the model being trained and how it can identify the existing patterns. During the development of this thesis, varied approaches were used, but one stood out as the most accurate and consistent among the entirety of the dataset.

Finally, in this chapter, the three adaptations of the core approach will be thoroughly explained, showcasing what knowledge it was possible to draw from each one and the reason for the transition between them. These approaches can be considered mutations of the main one, and the transition between them is due to the critical analysis of the results, the problems encountered, and the attempt to maximize the extrapolation of patterns detected in the data. Therefore, all are considered relevant to mention in this chapter to better understand the solution implemented and the reasoning that led to it. Due to the aforementioned reasons, this chapter can be considered a logbook of the development process.

5.1 CARTESIAN PRODUCT: SPECIFIC PROBLEM $\times$ DEVICE

As there is a time causality property in this problem, since, logically, the root problem has to happen before its consequences, such will be exploited in this approach. Additionally, as stated in chapter 3, an alarm is composed of multiple fields, but all that data is disregarded for implementing this approach since the purpose is to understand which alarms trigger others based merely on its occurrence and history. Thereby, a new type of event is created and, henceforth, an alarm is represented by the following tuple: (equipment where it occurred, its specific problem).

The goal is to create a model for each different type of event, so each model will try to predict the occurrence of a specific problem on a specific device through the history of alarms that have occurred. However, initially, due to dimensionality issues, the model is only defined for the problem, and the equipment, in which it occurs, is ignored. For the purpose of this approach, there will be two sliding windows: the observation window, where alarm occurrences are monitored and counted; and the prediction window, where through the alarms captured in the observation window, the model will try to predict if its alarm (the one which characterizes the model) is going to occur or not. For the observation window, it was used a 5-minute gap since it is considered that more than 80% of causality is present in 60 seconds or less [26], thus giving a secure margin for causality.

5.1.1 *Feature Engineering*

As stated before, for this approach, a new dataset has to be generated by applying the same operation to the entirety of the native dataset, which in this case is the scalar product between the specific problem of an alarm and the equipment where it occurred. This process aims to create a dataset where every row represents an observation window for every alarm entry of the default dataset. Therefore, applying this process to the entire dataset will generate a new one of equal dimension, and the value of each column represents how many times a specific problem on a specific device occurred during the observation window that led to the creation of such row.

Understanding this approach to the problem and the way the new dataset for the model is generated, it quickly becomes evident that all datasets (one for each different unique problem) will be identical(since the operation to generate every row is deterministic and equivalent between all models since they are all generated from the same initial dataset) with the only difference being the target variable. As an ML model is developed for every problem, each one only has to predict whether the problem it represents occurs or not, thus being binary classification models.

Having the original dataset concatenated, since Alarm Manager divides its data into multiple parquet files for every day, the process of generating a new dataset for the ML model training can be started. To better understand how it was solved at the code level, it will be explained step by step, thus allowing a more in-depth analysis of the process:

- *Dataset purification* - before using the native dataset to generate the one needed for the model, it is necessary to cleanse it again in order to remove as much noise and entropy as possible. Thereby, firstly, all entries whose type is not "NEW" are removed, as they do not represent the spawn of a new alarm but the update of an existing one, which for the thesis purpose represent no helpful information. Additionally, the entire native dataset is sorted by the attribute *systemCreationTime*, so every alarm entry is chronologically ordered, which will mitigate any possible redundancy during the following steps;

- *Pair Calculation* - as stated before, this approach uses the pair (specific problem detected, equipment where it occurred) as predictors. Thus, before starting the sliding windows process, it is necessary to know and list every existing unique pair. For this, a simple *tolist* of both features and a subsequent *zip* of both lists is enough, as both methods keep the order of each item.

- *Sliding Window Analysis* - at this point, it is finally possible to start analyzing each alarm history, hence creating the sliding windows. For this purpose, initially, it is created an array with the same length as the number of alarms to be analyzed, and each entry represents an array with the same length as the list of all unique pairs detected, initialized with zeros in every entry. Thereafter, for each alarm, the algorithm iterates through the native dataset and counts how many times each pair has occurred, and updates the value on the array of occurrences. At the end of this procedure, it achieves a matrix where the number of rows is equal to the number of alarms, the number of columns is equal to the number of unique pairs, and each entry represents how many times such pair has occurred during the observation window of its alarm. This step is crucial to be efficient as it is the heaviest and will iterate multiple times through the dataset. If a naive approach was used for this step, the sliding windows algorithm would be $\Theta(N^2)$; however, through the following optimizations, it was possible to lower it close to $\Theta(N)$:
  - *Chronological Order* - as stated in step one, the entire native dataset was chronologically ordered, which helps to reduce the number of iterations through the dataset massively. Having this property means that during iteration through all the alarms, from the first detected (oldest) to the latest (newest), when the alarm that originated the sliding window process is detected, the process can stop as all the alarms forward will be more recent than the originator, so it can never have any degree of causality.
  - *Saving Last Used Index* - since the alarms are temporally organized, it is evident that an event that is not part of the observation window of a past alarm can never be part of one that occurred afterward (since if it already violated the maximum time deviation of 5min, the difference will be even more significant). Thus, by saving the index of the first alarm used in the past observation window, it is possible to know which was the first event to be part of the last observation window. So, instead of having to iterate from the first alarm to fill in the causality matrix, this process can start at the first input used in the last observation window. Thus, reducing the number of iterations significantly through the native dataset for each window.

- *Class balancing* - having the causality matrix created and ready to pass to the random forest model for training, only one last step remains regarding the balancing of the target class. Since a different model is created for every unique specific problem, they are obviously binary models as either the model's desired problem occurred or not, a new setback arises: data unbalancement. Therefore, it is necessary to crop the causality matrix rows to balance the number of entries where the desired problem is detected versus the opposite scenario, simulating a 50/50 distribution, thus avoiding any model bias.

5.1.2 *Data Example*

Since it can be intricate to understand all the processing applied and the dataset obtained, a brief theoretical example, where the real would be too extensive and counterproductive for the purpose of this section, will be presented. Assuming the dataset seen in Table 4 as the one provided by Alarm Manager

Id	Problem	Extra Features	Equipment
1	Energy	...	EqpA
2	Signal	...	EqpA
3	Energy	...	EqpB
4	Signal	...	EqpA
5	Disk Space	...	EqpC
6	Signal	...	EqpA

Table 4: Alarm Manager's simulation dataset to exemplify the first approach

applying the process described in this section, assuming that the time difference between the first and last alarm is less than 5 minutes, generates an intermediate causality matrix and an array of problems, as can be seen in Table 5.

Energy@EqpA	Energy@EqpB	Signal@EqpA	Disk Space@EqpC
0	0	0	0
1	0	0	0
1	0	1	0
1	1	1	0
1	1	2	0
1	1	2	1

Problem
Energy
Signal
Energy
Signal
Disk Space
Signal

Table 5: Intermediate causality matrix and its corresponding array of problems (first approach)

After the process is complete and the intermediate causality matrix is generated, a dataset can be created for each model. As an example, for the model related to the "Energy" problem, the dataset that can be seen in Table 6 would be generated for its training and testing, thus completing the causality matrix for this problem.

Energy@EqpA	Energy@EqpB	Signal@EqpA	Disk Space@EqpC	Target
0	0	0	0	1
1	0	0	0	0
1	0	1	0	1
1	1	1	0	0
1	1	2	0	0
1	1	2	1	0

Table 6: Causality Matrix for the *Energy* Model (first approach)

5.1.3 *Discussion*

In order to have a fair testing standard among the various approaches, a native dataset, relative to the time between 1 and 16 of May, was created, thus assuring every ML model would have the same initial data from which it could extrapolate patterns. Additionally, instead of using standard CPU tools, such as Apache Spark, the RAPIDS.AI library seemed a way better option since it allows the model to run on GPU, which due to its high number of cores, has substantially higher performance in the range of up to 2000x times [44]. In a primary phase of this approach, no significant learning could be detected by the model due to insufficient and unbalanced data. Hence, it was necessary to create, as mentioned above, a consistent dataset (which was used for the multiple approaches) that had many alarm occurrences so that, in the scenarios where it is necessary to force balancing, there were still enough inputs for the model to be able to learn.

After solving this minor setback, a new, much more critical problem arose: dimensionality/scaling issue. As the columns of the causality matrix to be used are not native from the application that generates the data, but instead come from this approach's algorithm calculation, as described in this section, more rows represent more columns, as new pairs (Problem, Device) appear as more rows of the original dataset are processed. Here was detected an almost 2:1 relationship between the growth of rows and columns, as can be seen in the table 7.

Rows	10 000	100 000
Columns	6139	32 809

Table 7: Impact of the increase of rows processed on the number of columns (first approach)

The reasons why this scalability issue is so critical are: i) since the number of columns grows with the number of rows, the approach will always be limited to the amount of data it can process by hardware limitations (RAM), as the causality matrix grows in both dimensions simultaneously; ii) with the variable and increasing number of columns, the maximum column limit allowed by rapids.ai (close to 13k) is quickly reached, making it impossible to use such tool.

Considering the above problems, it was necessary to return to scikit-learn's random forest implementation, which solved the columns limit issue within the scalability problem. With this tool, it was possible to train the model and finally have learning results to analyze. To do so, as previously indicated, the dataset corresponding to all the alarms of the most frequent zone of the country between 1 and 16 of May was used, balancing the causality matrix in order for the alarm to be predicted *Loss of Signal* would represent close to 50% of its entries.

As expected, this approach demonstrated it was able to identify patterns and indeed learn to predict based only on the alarms causality, and quickly achieved accuracy in the 83-87% range. As the dataset is balanced, one can quickly conclude that this value is impossible to achieve if the model, instead of learning, always predicts the same class. However, to ensure that it was indeed learning, which is essential for the next step - *feature importance* - a confusion matrix was developed, as can be seen in Table 8, in order to be possible to visualize the model's behavior.

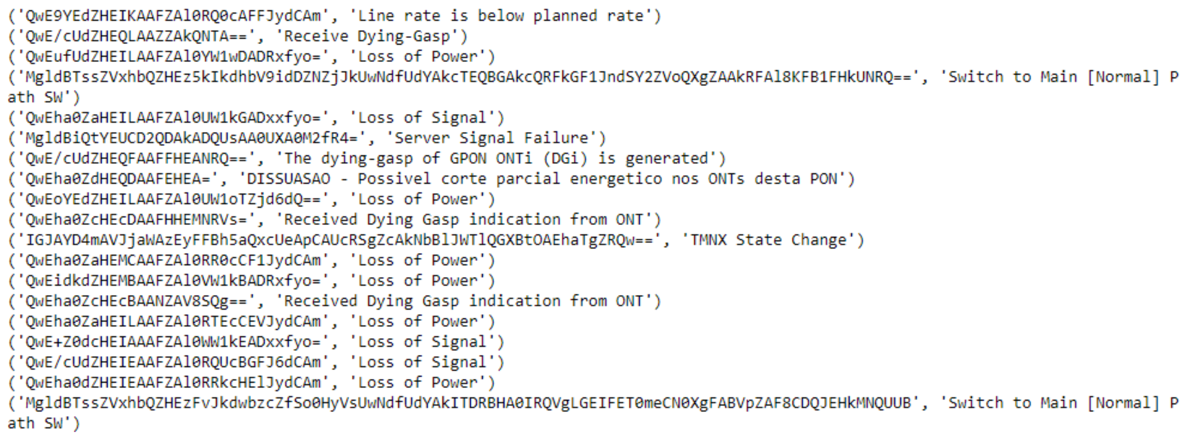
	Actual Positive	Actual Negative	Metrics	
Predicted Positive	15 997	497	Accuracy	ROC-AUC
Predicted Negative	4345	12 148	85.32%	83.5%

Table 8: Confusion Matrix of the *Loss of Signal* model (first approach)

It can immediately be detected that not only is the model effectively trying to predict the correct class target, but it is also achieving a high success rate at it. Additionally, it is visible that the number of false positives is meager, which is excellent for the project because, as something that was discussed with the product development team, it is preferable that events are related and the model is unable to detect the pattern, rather than detecting a pattern, and it is false. So, all errors in the model predictions are preferably false negatives over false positives.

After obtaining these positive results, it was possible to move on to a feature importance phase, which aims to identify how vital each column is in the model's decision making, and can thus be seen, as explainable AI [45]. This step was expected to find most decision certainty in the pair (device, problem) that was the root cause of the observation window in analysis. Using the result of the feature importance stage regarding all features with more than 0.5% impact on the model, as can be seen in figure 17, it is possible to quickly extrapolate two conclusions: i) the *Loss of Signal* problem is recursive/hierarchical, which is quite logical since if one device loses signal, a lower hierarchical device will also lose signal; ii) the model is not able to separate what is effectively the root cause of an alarm from its consequences.

While the first conclusion is positive and allows for new information about the system, the second presents a new setback. However, for the project under development during the curricular internship, it was considered that this root cause detail was not the goal since it is sought the relationship between problems. Thereby, and considering the problem just mentioned and the scalability issue, this was not the way to go, as these issues resolution would be pretty complex for something that is not the objective of the project. Thus, the second variant of this approach emerges, which will be described in detail in the following section, where instead of using the Cartesian product between Specific Problem × Device, only the problem will be used. In a successful training



```
('QwE9YEdZHEIKAAFZAl0RQ0cAFFJydCAm', 'Line rate is below planned rate')
('QwE/cUdZHEQLAAZZAkQNTA==', 'Receive Dying-Gasp')
('QwEufUdZHEILAAFZAl0YW1wDADRxfyo=', 'Loss of Power')
('MgldBTssZVxhbQZHEz5kIkdhbV9idDZNZjJkUwNdfUdYAkcTEQBGAkcQRFkGF1JndSY2ZVoQXgZAAkRFAl8KFB1FHkUNRQ==', 'Switch to Main [Normal] P
ath SW')
('QwEha0ZaHEILAAFZAl0UW1kGADxxfyo=', 'Loss of Signal')
('MgldBiQtYEUCD2QDAkADQUsAA0UXA0M2fR4=', 'Server Signal Failure')
('QwE/cUdZHEQFAAFFHEANRQ==', 'The dying-gasp of GPON ONTi (DGi) is generated')
('QwEha0ZdHEQDAAFEHEA=', 'DISSUASAO - Possivel corte parcial energetico nos ONTs desta PON')
('QwEoYEdZHEILAAFZAl0UW1oTZjd6dQ==', 'Loss of Power')
('QwEha0ZcHEcDAAFHHEMNRVs=', 'Received Dying Gasp indication from ONT')
('IGJAYD4mAVJjaWAzEyFFBh5aQxcUeApCAUcRSgZcAkNbBlJWTlQGXBtOAEhaTgZRQw==', 'TMNX State Change')
('QwEha0ZaHEMCAAFZAl0RR0cCF1JydCAm', 'Loss of Power')
('QwEidkdZHEMBAAFZAl0VW1kBADRxfyo=', 'Loss of Power')
('QwEha0ZcHEcBAANZAV8SQg==', 'Received Dying Gasp indication from ONT')
('QwEha0ZaHEILAAFZAl0RTEcCEVJydCAm', 'Loss of Power')
('QwE+Z0dcHEIAAAFZAl0WW1kEADxxfyo=', 'Loss of Signal')
('QwE/cUdZHEIEAAFZAl0RQUcBGFJ6dCAm', 'Loss of Signal')
('QwEha0dZHEIEAAFZAl0RRkcHElJydCAm', 'Loss of Power')
('MgldBTssZVxhbQZHEzFvJkdwbzcZfSo0HyVsUwNdfUdYAkITDRBHA0IRQVgLGEIFET0meCN0XgFABVpZAF8CDQJEHkMNQUUB', 'Switch to Main [Normal] P
ath SW')
```

Figure 17: Features with more than 0.5% impact on the *Loss of Signal* model's decision

scenario, this change of predictors will allow the feature importance process to indicate which past problems caused the one under analysis, thus showing the relationship between the problems as desired in the project.

5.2 SINGLE PREDICTOR: SPECIFIC PROBLEM

Since the project aims to identify the relationships between alarms and, considering the strengths and weaknesses of the last approach, it becomes evident that using only the problem as the column of the causality matrix is extremely promising.

In this section, the second variant of the aforementioned approach will thus be exposed, and, additionally, a sub-variation of this one will also be examined: division of the columns by temporal deviation from the event under analysis.

As mentioned before, despite the positive results of the approach - it effectively demonstrated learning and pattern detection capabilities - it was doomed to failure due to its scaling issue, which would permanently prevent the model from training with high amounts of data since the number of Problem-Device combinations increased significantly with the introduction of new lines to the causality matrix. However, this problem is mitigated with this approach, since by only using the *SpecificProblem* feature, the number of columns is drastically reduced, allowing not only a more extensive training but also, in the feature importance phase, the knowledge extraction about alarms' specific problem correlation.

5.2.1 *Feature Engineering*

As this approach is a permutation of the previous one referred to in this chapter, the feature Engineering steps are almost identical to the ones described in subsection 5.1.1, with the only difference in the "Pair Calculation" step and its transition to "Sliding Window Analysis". Whereas in the previous approach, all combinations (Problem, Device) existing in the dataset were pre-calculated, since they would be the columns of the causality matrix, this step is now unnecessary. Instead, it is necessary to identify all the unique existing problems since they will now be the columns of the causality matrix. Having the list of all unique *specificProblem*, its possible to proceed to the "Sliding Window Analysis" step, similarly to the previous approach, with the only difference being that now only the problems detected during the past 5 minutes are accounted for regardless of in which equipment they occurred. Hence, the number of columns is significantly reduced, thus reducing the causality matrix size, which also allows a much more extensive training. However, this will be further discussed in the Discussion subsection of this approach.

Time Deviation Sub-Approach

As mentioned at the beginning of this approach section, a sub-variant was implemented in a time separation perspective. Since the approach is now merely relying on the past problems as predictors for the model, an idea emerged during development: separate them according to their temporal remoteness. That is, instead of just using the Problems as columns, subdividing them into three different predictors:

- *Problem $<$ 30s* - the *specificProblem* at issue occurred up to 30 seconds before the alarm that led to the construction of the observation window under analysis;
- *Problem $\in$ [30-60]s* - the *specificProblem* at issue more than 30 seconds, but less than 1 minute, before the alarm that led to the construction of the observation window under analysis;
- *Problem $>$ 60s* - the *specificProblem* at issue occurred over 1 minute before the alarm that led to the construction of the observation window under analysis;

After the construction of the list with all different *specificProblem* and their subsequent division into three temporal spaces, the process is identical to the one previously described for the construction of the causality matrix, with the only difference that this will now have three times as many columns.

5.2.2 *Data Example*

Similar to the demonstration in the previous approach, this subsection will showcase a simple theoretical example of the application of the *Feature Engineering* phase of this approach for a better understanding of how it works. Assuming the dataset seen in Table 9 as the one provided by Alarm Manager

Id	Problem	Extra Features	Equipment
1	Energy	...	EqpA
2	Signal	...	EqpA
3	Energy	...	EqpB
4	Signal	...	EqpA
5	Disk Space	...	EqpC
6	Signal	...	EqpA

Table 9: Alarm Manager's simulation dataset to exemplify the second approach

applying the process described in this section, assuming that the time difference between the first and last alarm is less than 5 minutes, generates an intermediate causality matrix and an array of problems, as can be seen in Table 10.

Energy	Signal	Disk Space
0	0	0
1	0	0
1	1	0
2	1	0
2	2	0
2	2	1

Problem
Energy
Signal
Energy
Signal
Disk Space
Signal

Table 10: Intermediate causality matrix and its corresponding array of problems (second approach)

After the process is complete and the intermediate causality matrix is generated, a dataset can be created for each model. As an example, for the model related to the *Energy* problem, the dataset that can be seen in Table 11 would be generated for its training and testing, thus completing the causality matrix for this problem.

Energy	Signal	Disk Space	Target
0	0	0	1
1	0	0	0
1	1	0	1
2	1	0	0
2	2	0	0
2	2	1	0

Table 11: Causality Matrix for the *Energy* Model (second approach)

Time Deviation Sub-Approach

For the sub-variant of this approach with the concern of temporal remoteness and, assuming for the sake of this theoretical explanation, that the alarms in Table 9 are all equally 30s apart from each other, after applying the additional feature engineering steps, it would then reach the intermediate causality matrix that can be seen in Table 12.

Energy $<$ 30s	Energy $\subseteq$ [30s-60s]	Energy $>$ 30s	Signal $<$ 30s	Signal $\subseteq$ [30s-60s]	Signal $>$ 30s	Disk Space $<$ 30s	Disk Space $\subseteq$ [30s-60s]	Disk Space $>$ 30s	Problem
0	0	0	0	0	0	0	0	0	Energy
1	0	0	0	0	0	0	0	0	Signal
0	1	0	1	0	0	0	0	0	Energy
1	0	1	0	1	0	0	0	0	Signal
0	1	1	1	0	1	0	0	0	Disk Space
0	0	2	0	0	2	1	0	0	Signal

Table 12: Intermediate causality matrix and its corresponding array of problems (second approach with time deviation)

After this intermediate step, as mentioned in the past examples, it is possible to enter the final stage, where through a Boolean comparison, it is possible to generate the final causality matrix, as can be seen in Table 13 for the *Energy* model.

Energy $<$ 30s	Energy $\subseteq$ [30s-60s]	Energy $>$ 30s	Signal $<$ 30s	Signal $\subseteq$ [30s-60s]	Signal $>$ 30s	Disk Space $<$ 30s	Disk Space $\subseteq$ [30s-60s]	Disk Space $>$ 30s	Target
0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1
1	0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	0	0	0	0
0	0	2	0	0	2	1	0	0	0

Table 13: Causality Matrix for the *Energy* Model (second approach with time deviation)

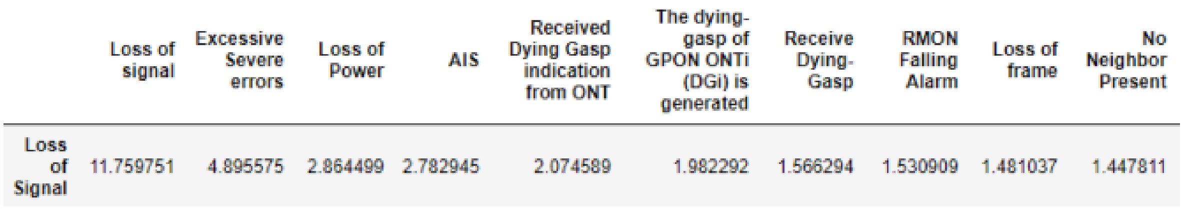
5.2.3 *Discussion*

Contrary to the last approach, which used the Cartesian product between *specificProblem* and *managedObjectInstance*, through the one described in this section, it becomes possible to train the Random Forest model with a significantly larger dataset, as the number of columns increases very slightly with the introduction of more rows, as shown in Table 14.

Rows	10 000	100 000	400 000
Columns	340	1481	3233

Table 14: Impact of the increase of rows processed on the number of columns (second approach)

Considering the success of the previous approach, which, due to the scaling issue, was disadvantaged for the training phase, it was expected that this variation in analysis, that despite only using *specificProblem* as a predictor, with access to a more extensive and varied dataset, would also be able to identify patterns between alarms and achieve a satisfactory success rate. In a test scenario of the same problem, "Loss of Signal" - the most frequent in the dataset, constituting 21% of it - the model showed its ability to learn and detect patterns effectively. Not only did it achieve a high success rate, as can be seen in Table 15, but it also had high percentage values at the feature importance phase, thus showing its capacity to identify the relationships among problems, as can be seen in Figure 18.



	Loss of signal	Excessive Severe errors	Loss of Power	AIS	Received Dying Gasp indication from ONT	The dying-gasp of GPON ONTi (DGi) is generated	Receive Dying-Gasp	RMON Falling Alarm	Loss of frame	No Neighbor Present
Loss of Signal	11.759751	4.895575	2.864499	2.782945	2.074589	1.982292	1.566294	1.530909	1.481037	1.447811

Figure 18: Top 10 feature importance of the *Loss of Signal* model

	Actual Positive	Actual Negative	Metrics	
Predicted Positive	11 958	275	Accuracy	ROC-AUC
Predicted Negative	3176	8261	85.42%	85.35%

Table 15: Confusion Matrix of the *Loss of Signal* model (second approach)

Having solved the scaling problem, the current approach demonstrating a good success rate and a new ability to train with large volumes of data, it became possible to transition from the top1 - Loss of Signal - to the top 7 most recurring problems, which represent about 58% of the dataset. At this stage, unexpectedly, it was detected that the problem with more significance in the dataset was also the one with the least successful model. Although initially off-putting, this makes perfect sense when considering the degree of repetition of the same failure, as it was so recurrent that the model had more difficulty in detecting which combination of failures caused it. When the models started to analyze the following problems, which despite belonging to the top7 are much less significant than the top1, each model started to reach very high success rates, as can be shown in Figure 19. These models have approached the maximum theoretical value of AUC - 1 - meaning the model has almost 100% capable of distinguishing between the two classes (the problem occurring or not).

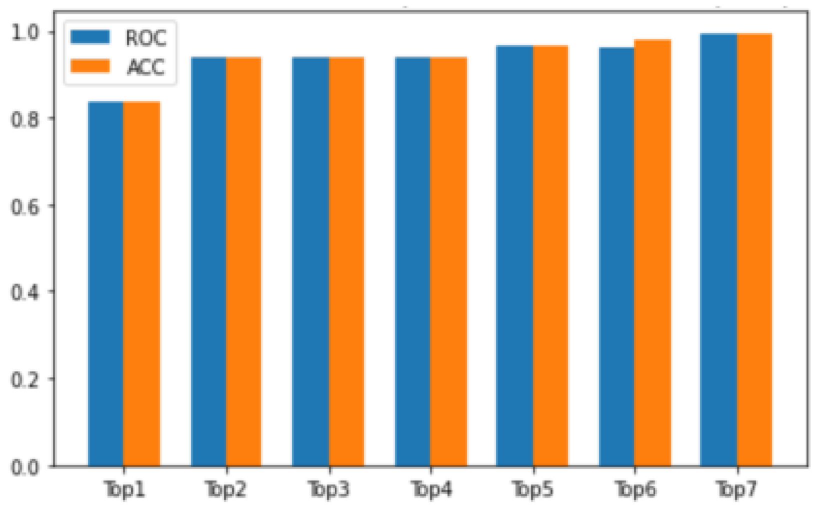


Figure 19: ROC-AUC and Accuracy metrics for the top7 models

At this point, the thesis goal seemed to have been achieved, as the model effectively demonstrated a considerable capacity to detect patterns among the various faults, and it would be possible to move on to a much more extensive training and testing phase in order to collect all possible information present in the dataset. However, after a new exchange of ideas with the Alarm Manager's development team, an idea for an even better model for the task in question emerged. Instead of just using the *specificProblem* as a predictor, do the cartesian product between it, the subsystem - which represents the supplier of the equipment where the failure occurred - and technology - which identifies the technology the equipment uses. Thus, a new approach emerges, which, after all that has already been achieved, promises to be the perfect solution to the problem proposed in this thesis and is therefore described in detail in the next section.

Time Deviation Sub-Approach

Regarding the sub-variant with time offset concerns, it was found to have no impact, positive or negative, on the model's performance as it maintains the same values in the test metrics. However, it could be considered that it provided more information to the reader/results analyst by identifying the temporal distance between failures, which in itself is not sufficiently favorable for the scaling problem it generates by tripling the number of columns in the causality matrix.

5.3 CARTESIAN PRODUCT: SPECIFIC PROBLEM × TECHNOLOGY × SUBSYSTEM

Although initially, it may seem that this will restore the scaling issue since the cartesian product will be applied between 3 features, such does not happen. While the number of combinations of (Problem, Device) is colossal, since all devices can suffer from different problems, the number of permutations between the three features used in this approach is relatively small, since the number of different manufacturers or technologies used by the same device is meager.

5.3.1 *Feature Engineering*

As in subsection 5.2.1, this approach is a new variant of the one mentioned in subsection 5.1.1, so the whole preprocessing process is quite identical with only two differences:

- *Triples Calculation* - likewise the past approaches, before creating the observation windows that will lead to the causality matrix, it is necessary to calculate all the unique predictors. In this case, such can be achieved by applying the Cartesian product among the following features: *specificProblem*, *Technology* and *Subsystem*. As stated before, even though this product is between 3 different features, their relation is almost bijective; hence the number of combinations is so close to the number of unique *specificProblem*.
- *Model For Each Triple* - In contrast to the past approaches, where a model was problem-oriented, meaning a model was created for every unique problem in the dataset, and it would try to predict whether if the observation window in the analysis was of its type or not, now a model is created for every unique triple of (*specificProble*, *Technology*,*Subsystem*). Since the predictors now go into a much more specific degree of detail, taking into account not only the manufacturer but also the technology used by the equipment, and, fortunately, the number of combinations is quite similar to the unique problems, it becomes pretty advantageous to generate a model for each triple. Through this, each model can find even more specific relationships since the root for a problem in one manufacturer does not have to be identical to that of another manufacturer or even one using a different technology.

5.3.2 *Data Example*

Similar to the demonstration in the previous approach, this subsection will showcase a simple theoretical example of the application of the *Feature Engineering* phase of this approach for a better understanding of how it works. Assuming the dataset seen in Table 16 as the one provided by Alarm Manager

Id	Problem	Extra Features	Technology	Subsystem
1	Energy	...	Tech1	Sub1
2	Signal	...	Tech2	sub2
3	Energy	...	Tech1	Sub2
4	Signal	...	Tech3	Sub2
5	Disk Space	...	Tech3	Sub3
6	Signal	...	Tech2	Sub2

Table 16: Alarm Manager's simulation dataset to exemplify the third approach

applying the process described in this section, assuming that the time difference between the first and last alarm is less than 5 minutes, generates an intermediate causality matrix and an array of problems, as can be seen in Table 17.

Energy-Tech1-Sub1	Energy-Tech1-Sub2	Signal-Tech2-Sub2	Signal-Tech3-Sub2	Disk Space-Tech2-Sub1
0	0	0	0	0
1	0	0	0	0
1	0	1	0	0
1	1	1	0	0
1	1	1	1	0
1	1	1	1	1

Problem
Energy
Signal
Energy
Signal
Disk Space
Signal

Table 17: Intermediate causality matrix and its corresponding array of problems (third approach)

After the process is complete and the intermediate causality matrix is generated, a dataset can be created for each model. As an example, for the model related to the *Energy* problem, the dataset that can be seen in Table 18 would be generated for its training and testing, thus completing the causality matrix for this problem.

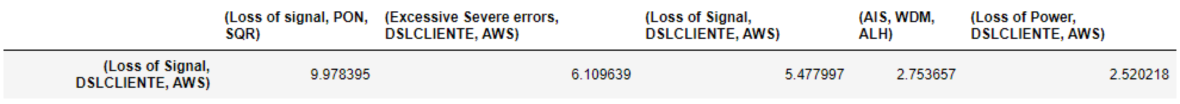
Energy-Tech1-Sub1	Energy-Tech1-Sub2	Signal-Tech2-Sub2	Signal-Tech3-Sub2	Disk Space-Tech2-Sub1	Target
0	0	0	0	0	1
1	0	0	0	0	1
1	0	1	0	0	1
1	1	1	0	0	1
1	1	1	1	0	1
1	1	1	1	1	1

Table 18: Causality Matrix for the *Energy* Model (third approach)

5.3.3 *Discussion*

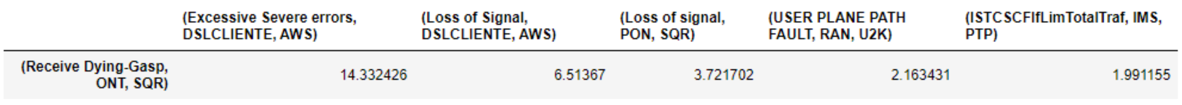
Since the previous approach was very successful and showed full learning capacity, through the improvement explained during this section to the approach, it was expected to get as good or even better results, since: i) the model has two more predictors than the previous version, thus having two more information variables for decision making; ii) the models are now oriented to triples (*specificProble*, *Technology*,*Subsystem*) instead of only to the problem, thus avoiding wrong generalizations, since the same problem in equipment from different manufactures or with different technologies may have different failure causes.

Applying the same test phase used on the previous approach quickly showed that the models could effectively at least maintain, but mostly improve their performance, both in a metric perspective of the models (ROC and ACC) and also in a feature importance phase where it was able to identify triples with relatively high impact on the origin of the triple under analysis, as can be seen in Figures 20 to 22.



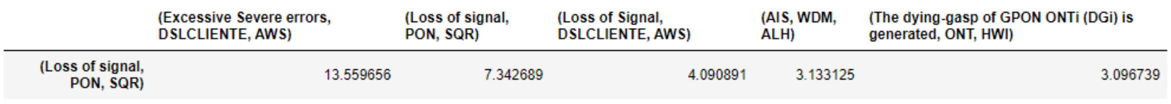
	(Loss of signal, PON, SQR)	(Excessive Severe errors, DSLCLIENTE, AWS)	(Loss of Signal, DSLCLIENTE, AWS)	(AIS, WDM, ALH)	(Loss of Power, DSLCLIENTE, AWS)
(Loss of Signal, DSLCLIENTE, AWS)	9.978395	6.109639	5.477997	2.753657	2.520218

Figure 20: Top5 feature importance of the model (Loss of Signal, DSLCLIENTE, AWS)



	(Excessive Severe errors, DSLCLIENTE, AWS)	(Loss of Signal, DSLCLIENTE, AWS)	(Loss of signal, PON, SQR)	(USER PLANE PATH FAULT, RAN, U2K)	(ISTCSCFIfLimTotalTraf, IMS, PTP)
(Receive Dying-Gasp, ONT, SQR)	14.332426	6.51367	3.721702	2.163431	1.991155

Figure 21: Top5 feature importance of the model (Receive Dying-Gastp, ONT, SQR)



	(Excessive Severe errors, DSLCLIENTE, AWS)	(Loss of signal, PON, SQR)	(Loss of Signal, DSLCLIENTE, AWS)	(AIS, WDM, ALH)	(The dying-gasp of GPON ONTi (DGi) is generated, ONT, HWI)
(Loss of signal, PON, SQR)	13.559656	7.342689	4.090891	3.133125	3.096739

Figure 22: Top5 feature importance of the model (Loss of signal, PON, SQR)

So, even before moving on to a much more extensive training and testing phase, the approach was tested for the top 15. The goal was to check whether it would still be able to find the patterns in the eight new problems, which were less significant in the dataset than those analyzed so far. As in subsection 5.2.3, the discovery that less common problems in the dataset (although still quite significant: Top15) are easier to predict and detect patterns was again verified. Given this empirically obtained result, it becomes evident that the reason why the less significant problems in the dataset are easier to predict is due to the specificity of their failure cause. While the problem, for example, "Loss of Signal", is caused by a myriad of different reasons, the less frequent ones have a consistent pattern of occurrence. This way, no matter how complex or not evident that pattern may be to a human, a machine learning algorithm quickly detects it and improves its predictive ability, thus making the new models reach the almost utopian metric values (ROC above 99.5%, being the maximum theoretical value 100%), as can be seen in Figure 23.

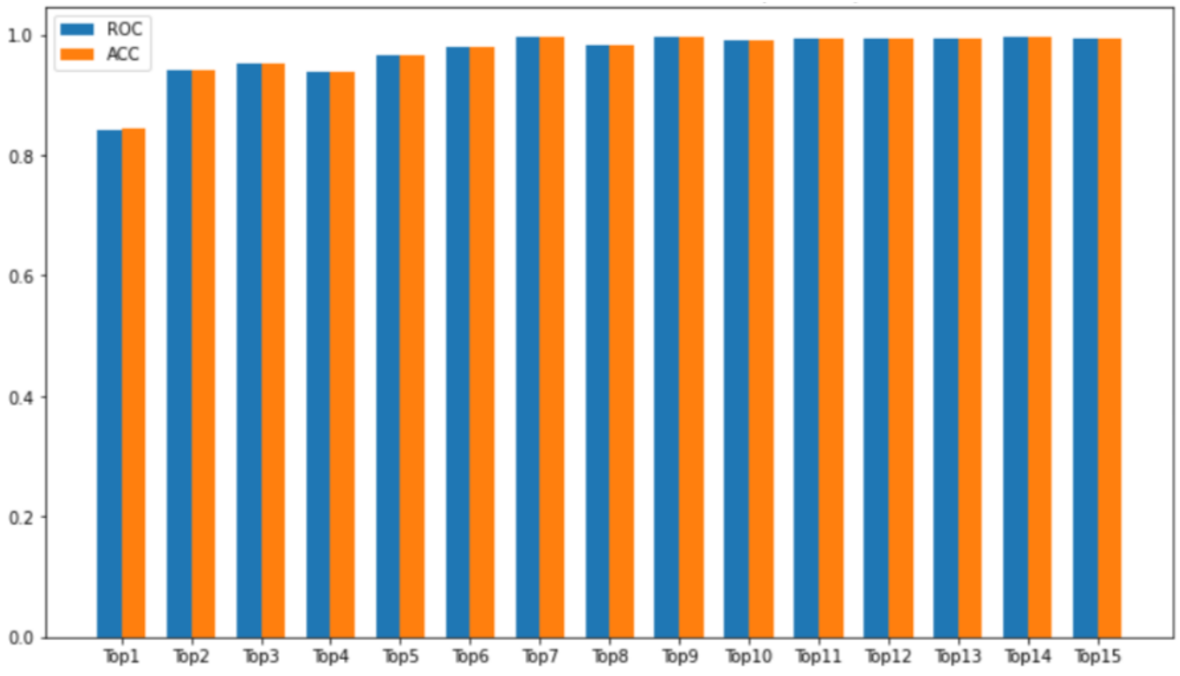


Figure 23: ROC-AUC and Accuracy metrics for the top15 models

At this point, it was considered that the approach had reached its maximum point of maturity and improvement since it became possible to conclude previously unknown relationships between alarms. Thus, the chapter regarding the approaches and their functioning is closed, and it is possible to move on to the results chapter, where all the training stages and methods will be exposed, alongside all the results obtained with the development of this thesis.

5.4 SUMMARY

This chapter explains in detail the approach used in this project, considering three variants regarding different predictors. Thus, the potential of each is exposed, accompanied by a theoretical example for a better understanding and a result obtained to ascertain the usefulness of the approach.

At the end of this chapter, the specific variant of the approach to be used is defined, and it becomes necessary to enter a more extensive and intensive testing phase.

6 RESULTS DISCUSSION

After developing and refining the approach, an expansion of the training and test case becomes necessary. Whereas previously, it was only tested up to the top15 most significant problems in the most recurrent area of the country, having the approach optimized allows expanding the training and testing phase. Thus, it will be applied to all problems in all areas to extrapolate as much knowledge as possible from the data provided for the project.

This chapter will cover not only the technical specifications of the machine used during the training and testing phase (since in a big data and machine learning area, hardware resources are fundamental for a fluid development), the automation process that allowed a more accurate and consistent training across all zones and problems and, finally, the results and knowledge obtained after all the continuous training of the models.

6.1 HARDWARE

One of the main limiting factors in big data processing and machine learning is the hardware resources [46], as it not only needs to access many data simultaneously but also needs to apply computationally heavy learning algorithms. Additionally, there is a considerable gap between the minimum requirements and the recommended ones, i.e., even if it is a heavy process, most of the currently available machines can run it, but the processing times can be so high that makes the development stage utterly impossible. On the contrary, there are machines available to rent that have immense computational power and would almost immediately do the training stage for this project. However, they are costly to rent, and due to their power, in the development stage, it would not make sense to rent as most of the paid time would not be on training but refining the approach, thus wasting the money spent on the machine. So it is necessary to find a middle ground that allows a stable development and that allows running the models several times without waiting endless hours for results, which always delays the efficient development of the project.

Thus, the machine used for the development of the project was my personal computer, which has the technical specifications that can be seen in Table 19.

Component	Name	Speed/Capacity/Quantity
CPU	Intel i7-6700k	4 64-bit cores @ 4.2GHz
GPU	Nvidia Geforce GTX 1070	8GB GDDR5 @ 1683 Mhz
Motherboard	Asrock Fatal1ty Z170 Gaming K6	Socket 1151
RAM	Kingston HyperX Fury DDR4	2 x 16GB @ 3000MHz CL15
RAM	GeIL SUPER LUCE	2 x 8GB @ 3000MHz CL15
Storage	Western Digital Blue	1TB, SATA3
Storage	Crucial 2.5"	256GB, SSD

Table 19: Specification of the system used during the development stage

These are the machine's components worthy of mentioning, as others have no significance for the project, such as the power supply. Of these components, it is worth highlighting that:

- *CPU* - as the random forest models are trained on CPU, this component performance has a huge impact on the training time. The processor used here is quad-core with hyper-threading, meaning despite having four fixed cores, it has eight execution threads, thus allowing multiple parallel processing. Additionally, it is the "K" version, so the processor frequency is "unlocked", allowing it to overclock during higher load scenarios;

- *GPU* - in the initial phase in which models were run over GPU, this was the most crucial component, but with the change to *sklearn*, it is no longer used. In the machine learning area, the essential factors in a GPU are the high number of *CUDA* cores and the dedicated memory available, since when models are trained on GPU, the memory they have access to is not the RAM, but the graphics dedicated VRAM.

- *RAM* - after the CPU, this is the component with the most impact as it is where all the information relating to and necessary for training is stored. As described in the approach chapter, the causality matrix is of very high dimension, not only for having lots of rows but also lots of columns, thus requiring immense amounts of RAM for storage. At an early stage of the project, the machine was equipped with only 16GB of RAM, but it was quickly detected that it was necessary to upgrade for more extensive training. So, an additional kit of 32GB of RAM was purchased, making a total of 48GB of RAM (sufficient for training with more extended periods, thus more significant causality matrices).

- *Storage* - this component only needs to be able to store all the data provided by a TO and the results generated from the processing. Although the SSD has a significantly higher performance than the HDD [47], this aspect has no impact on the project's performance, as the disk time for reads and writes are minimal compared to the rest of the learning and testing process.

6.2 AUTOMATION PROCESS

During the development phase, as mentioned in Chapter 5, the training was limited to a single zone (the one with the highest number of problems in the country), from only a single problem to the top15 after the successful evolution of the implemented approach. However, there are thousands of different problems in the dataset, as stated in section 3.4 and 55 different zones in the country. So the approach needs to be extended and tested in all possible scenarios to extract as much information as possible from the system. While when running for just one zone, it is possible to manually do it with different balancing and then do a weighted analysis of the results to change whatever is needed; this becomes quite complicated when needed to expand the length of the training phase massively.

Therefore, an "automation" of the training process was necessary and implemented, allowing that instead of manually running the models and then applying the necessary test changes, this occurs automatically. Thus allowing the model to run non-stop until it reaches the stop case condition. To better understand the automation process and why it is fundamental, it will be described step by step:

- *Creation of Files by Zone* - since now it is desired to expand the approach to all zones of the country in order to detect if patterns are consistent between zones or if their location has an impact on the relationships detected, the datasets for each zone must be created. Thus, it was necessary to go back to using the native Alarm Manager files in order to: i) aggregate all the files corresponding to the interval from May 1st to May 16th (same spatial interval used during the testing of the approaches); ii) separate the corresponding dataframe by the lowest granularity zone "localCode2D", in order to generate 55 datasets, each corresponding to a unique zone of the country;

- *Cycles* - as expected, automating the training process involves applying the entire learning process to all problems, in this case, all triples (*specificProble*, *Technology*,*Subsystem*) of the dataset. The simplest and most efficient way to do this is through a cycle. However, it was necessary to use four nested cycles in order to automate the entire process, being those as follows:
  - *Zone Files Cycle* - the first and most broad cycle, which encompasses the entire pre-processing and training process, is responsible for the zone files. This cycle iterates through all the different zones in order to train models for all unique triples (*specificProblem*, *Technology*,*Subsystem*) in every zone, thus allowing, as previously mentioned, to determine whether the location has an impact on the relationship between problems. After removing unnecessary entries from the dataset, as the files generated for the zones do not take into account the approach in practice, a new, "clean" version of the dataset is generated, and it becomes possible to calculate all the different existing triples, which is fundamental for the next cycle implemented. Additionally, the causality matrix for the zone in the analysis is calculated during this step.
  - *Triples Cycle* - while previously the number of problems to be analyzed was pre-defined, the approach is now considered mature enough to remove this limitation. Thus, the second cycle iterates through all the different trios (*specificProble*, *Technology*,*Subsystem*), allowing models to be trained

for, in a best-case scenario, all the problems. However, it is considered useless for models with past mediocre results to be trained as it will not be possible to extract any useful information from them. So, a new guard was added to the cycle: the model must obtain more than 60% ROC: since 50%s is the value at which the model identifies that it has 0 ability to separate the two classes and 100% is the theoretical maximum value (the model has full ability to distinguish the two target classes) 60% seemed good as a middle ground and an acceptable minimum threshold value. Additionally, the target variable array (list of 0's and 1's determining whether the problem in analysis occurred or not) is calculated during this step, thus allowing the next step to have all information necessary for training.

– *Balancing Cycle* - Since when running manually, different balancing was consistently used in order to see the impact on the quality of the model, it was considered valuable to keep this concern during the automation process. So, this third cycle makes three iterations to simulate the following balancing scenario: 50-50 (yes-no), 40-60 and 30-70. This allows the analysis of balancing has for training a model and identifying which scenario allows for the best learning ability. As can be seen in Table 20, the 50-50 balancing scenario is the one that consistently shows the best overall results. The difference in metrics is mitigated by analyzing more problems because, as we will see in the next section, after a certain point, the problems are so consistent in their occurrence process that the models get 100%, absolute theoretical maximum no matter the balancing applied. Additionally, the model's metrics for the different balancings are saved during this step to detect when the first cycle should stop: 60% ROC limit achieved.

	Top1	Top5	Top15	Top50
Balancing 50/50	85.31%	94.09%	97.35%	99.08%
Balancing 40/60	79.49%	91.40%	96.01%	98.59%
Balancing 30/70	72.94%	88.92%	94.79%	98.18%

Table 20: Class balancing impact on models performance

– *Saving the Results* - this last cycle is responsible for saving the model results obtained at the end of the second cycle, that is, at the end of the training with three different balancings. Through this, a new excel file is generated for each zone, which will be composed of multiple tables, where each one corresponds to one of the triples (*specificProble*, *Technology*,*Subsystem*) analyzed, and its 14 columns are: the ten triples with the most impact on the one under analysis, the ROC value obtained by the model during training, the Accuracy value obtained, the number of positive inputs used during training and the number of negative inputs. These last four metrics are recorded alongside the model results to describe better the quality/veracity of the information drawn.

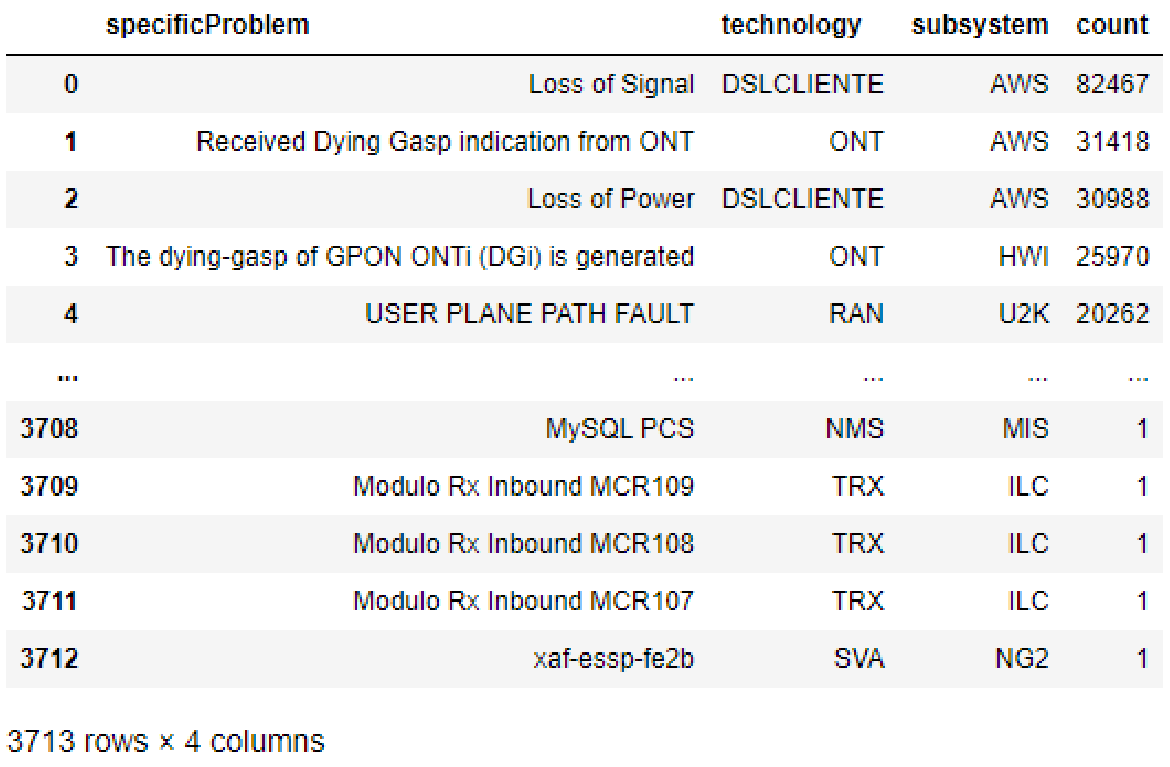
Finally, having the automation process implemented, it was possible to move on to a much more intensive training phase, from which all the conclusions and patterns obtained in this work were drawn, thus being exposed in the following section.

6.3 RESULTS

In order to better understand the results obtained during the development of the project, these will be exposed according to their granularity, thus allowing a more critical and precise interpretation. Initially, the results obtained when the approach was applied to a single problem (the most significant one in the dataset) will be shown, followed by the results of applying it to an entire zone (the one with the most alarm occurrences in the country) and, finally, the result of iteratively applying the approach to all the zones in the country, analyzing all the different problems.

6.3.1 *One Problem*

For the problem that will be showcased, the dataset used is relative from May 1st to May 16th, of the most significant area in the country, which, after all the dataset processing, has approximately 670k entries; however, due to hardware limitations (RAM capacity), the first 400k will be used. Therefore, before applying the approach to the problem, it is necessary to calculate, as indicated in the approaches chapter, all the triples existing in the dataset to be used as they will be the columns of the causality matrix. Thus, in the dataset used, there are 3713 different trios, as shown in Figure 24. Additionally, for a better sense of the distribution of the trios on the 400k entries dataset, the number of occurrences of each was saved.

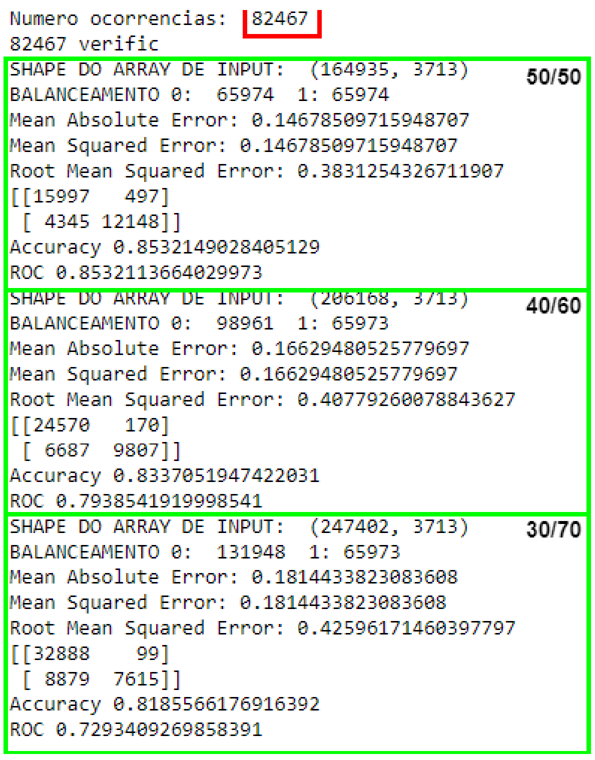


	specificProblem	technology	subsystem	count
0	Loss of Signal	DSLCLIENTE	AWS	82467
1	Received Dying Gasp indication from ONT	ONT	AWS	31418
2	Loss of Power	DSLCLIENTE	AWS	30988
3	The dying-gasp of GPON ONTi (DGi) is generated	ONT	HWI	25970
4	USER PLANE PATH FAULT	RAN	U2K	20262
...	...	...	...	...
3708	MySQL PCS	NMS	MIS	1
3709	Modulo Rx Inbound MCR109	TRX	ILC	1
3710	Modulo Rx Inbound MCR108	TRX	ILC	1
3711	Modulo Rx Inbound MCR107	TRX	ILC	1
3712	xaf-essp-fe2b	SVA	NG2	1

3713 rows × 4 columns

Figure 24: All unique trios present in the zone with most alarms in the country

At this point, it is already possible to apply the approach to be used to the dataset for the most frequent trio, in this case being the ("Loss of Signal", "DSLCLIENTE", "AWS"). During the algorithm execution, since it will be applied three times to each trio, in order to test different balances, the program will print relevant information about the model for better monitoring of the processing, as shown in Figure 25.

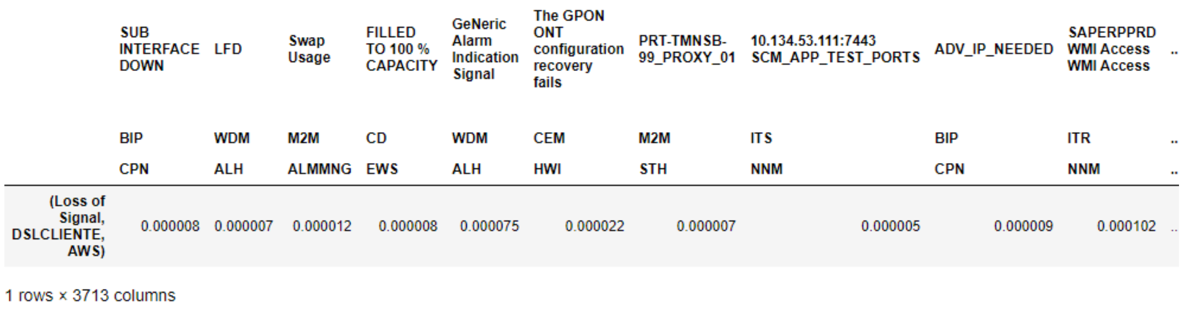


```
Numero ocorrencias:  82467
82467 verific
SHAPE DO ARRAY DE INPUT:  (164935, 3713)
BALANCEAMENTO 0:  65974  1: 65974
Mean Absolute Error: 0.14678509715948707
Mean Squared Error: 0.14678509715948707
Root Mean Squared Error: 0.3831254326711907
[[15997   497]
 [ 4345 12148]]
Accuracy 0.8532149028405129
ROC 0.8532113664029973
SHAPE DO ARRAY DE INPUT:  (206168, 3713)
BALANCEAMENTO 0:  98961  1: 65973
Mean Absolute Error: 0.16629480525779697
Mean Squared Error: 0.16629480525779697
Root Mean Squared Error: 0.40779260078843627
[[24570   170]
 [ 6687  9807]]
Accuracy 0.8337051947422031
ROC 0.7938541919998541
SHAPE DO ARRAY DE INPUT:  (247402, 3713)
BALANCEAMENTO 0:  131948  1: 65973
Mean Absolute Error: 0.1814433823083608
Mean Squared Error: 0.1814433823083608
Root Mean Squared Error: 0.42596171460397797
[[32888    99]
 [ 8879  7615]]
Accuracy 0.8185566176916392
ROC 0.7293409269858391
```

Figure 25: Information printed during the training phase of a problem

Some boxes were added to the above figure to highlight the most critical details printed. Initially, inside the red box is the number of occurrences of the trio under analysis in the dataset, which is relevant information since if this value is too low, the model will not be able to learn and detect patterns. Next, relevant metrics are printed about the model for each of the three different balancing scenarios, presented in the following order: 50-50 (True-False), 40-60 and 30-70, each highlighted by a green box on the figure. The first line indicates the dimensions of the causality matrix to train and test the model, followed by the exact distribution of the target class for an accurate view of the balance in the training phase. Thereafter, 3 model metrics are printed regarding its errors (the smaller they are, the better), followed by the confusion matrix obtained by the model during the testing phase - 80% of the data is used for training and 20% for testing - and finally, the ROC and accuracy values obtained by the model are printed, these being are the most relevant metrics of the model.

After the end of the training and testing process of the three balances, the model with the best performance - usually the 50-50, but coded to choose the one with the best metrics - is saved, thus allowing it to move to a feature importance phase to detect the impact of each triple on the triple under analysis. Through this step, it is then possible to generate a dataframe, as can be seen in Figure 26, whose columns are identical to those of the causality matrix, but instead of storing how many times that triple occurred, the direct impact of that triple forecasting the one under analysis is store.

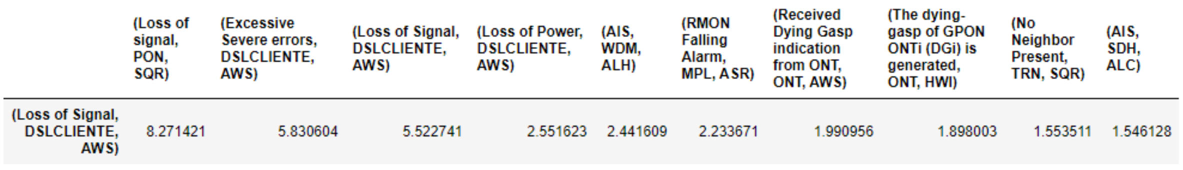


	SUB INTERFACE DOWN	LFD	Swap Usage	FILLED TO 100 % CAPACITY	GeNeric Alarm Indication Signal	The GPON ONT configuration recovery fails	PRT-TMNSB-99_PROXY_01	10.134.53.111:7443 SCM_APP_TEST_PORTS	ADV_IP_NEEDED	SAPERPPRD WMI Access WMI Access	..
	BIP	WDM	M2M	CD	WDM	CEM	M2M	ITS	BIP	ITR	..
	CPN	ALH	ALMMNG	EWS	ALH	HWI	STH	NNM	CPN	NNM	..
(Loss of Signal, DSLCLIENTE, AWS)	0.000008	0.000007	0.000012	0.000008	0.000075	0.000022	0.000007	0.000005	0.000009	0.000102	..

1 rows × 3713 columns

Figure 26: Result of applying Feature Importance to the Model (Loss of Signal, DSLCLIENTE, AWS)

However, as seen in the above figure, several columns have values without any significance (values lower than 0.01%). Since the project aims to discover the relationship between problems, an analysis of all the predictors is useless, so it is necessary to filter the most important ones, which impact the trio under analysis. Thus, instead of saving the dataframe with the impact of all the predictors, a new one is generated, as shown in Figure 27, which identifies the ten most important features and their corresponding significance. As can be seen, all the predictors now displayed have a higher impact and allow to extract information about the system, such as: since the "Loss of Signal" problem has itself as the predictor with the highest impact, it is immediately apparent that this is a recursive/hierarchical problem (if a piece of equipment loses access to the network, a hierarchically dependent one will also lose it) and that it is more problematic if the subsystem and technology the equipment uses are "PON-SQR" instead of "DSLCLIENTE-AWS", even tho the trio under analysis has "DSLCLIENTE-AWS" as its subsystem and technology.



	(Loss of signal, PON, SQR)	(Excessive Severe errors, DSLCLIENTE, AWS)	(Loss of Signal, DSLCLIENTE, AWS)	(Loss of Power, DSLCLIENTE, AWS)	(AIS, WDM, ALH)	(RMON Falling Alarm, MPL, ASR)	(Received Dying Gasp indication from ONT, ONT, AWS)	(The dying-gasp of GPON ONTi (DGi) is generated, ONT, HWI)	(No Neighbor Present, TRN, SQR)	(AIS, SDH, ALC)
(Loss of Signal, DSLCLIENTE, AWS)	8.271421	5.830604	5.522741	2.551623	2.441609	2.233671	1.990956	1.898003	1.553511	1.546128

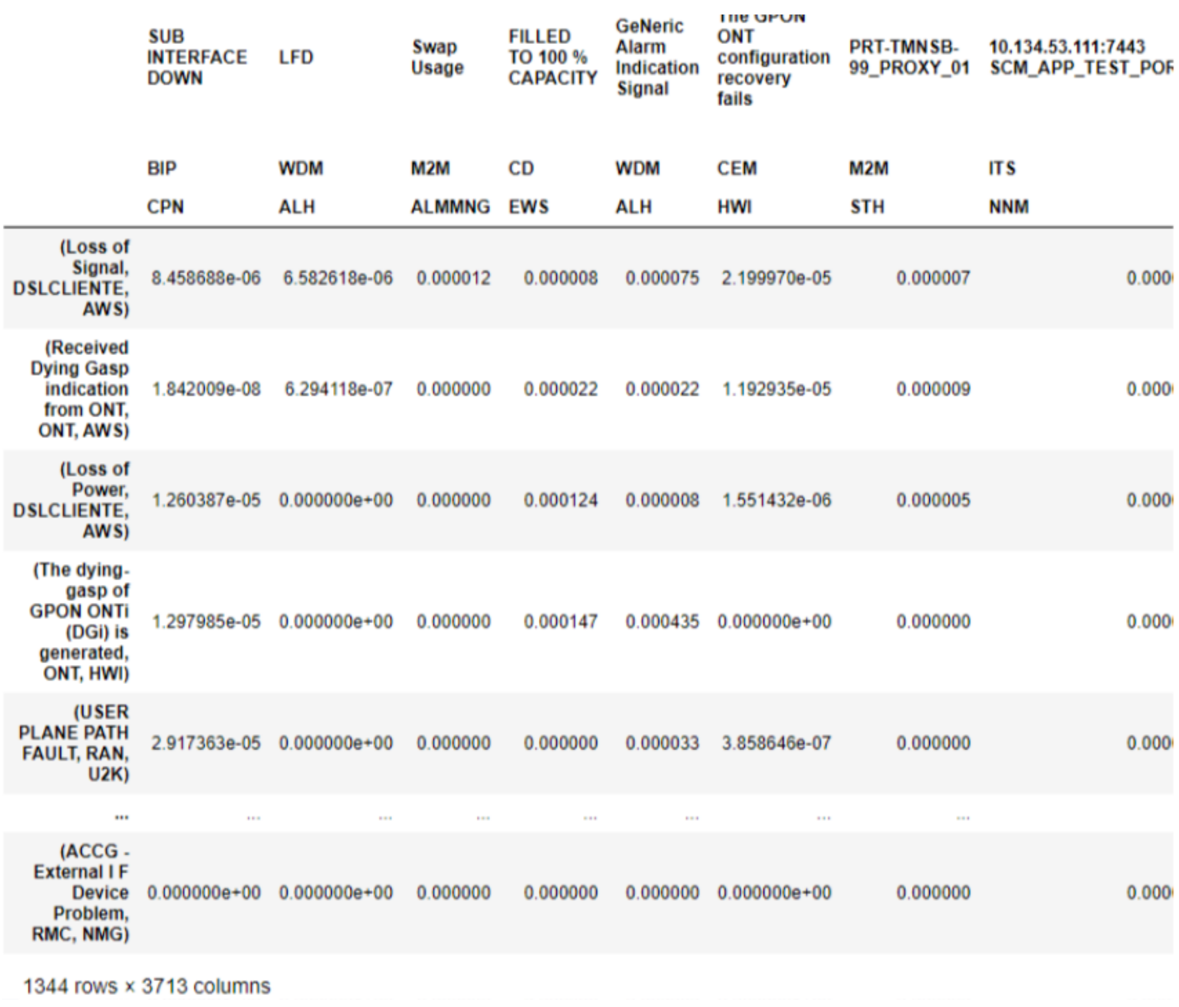
Figure 27: Top10 feature importance of the model (Loss of Signal, DSLCLIENTE, AWS)

These results were obtained by applying the approach explained in section 5.3 on the machine with the specifications referenced in Table 19, having an average execution time (average between 5 distinct executions) of 37 minutes.

6.3.2 *One Zone*

Once understood the approach used and the results obtained for a single problem in a single zone, the training phase can be expanded: all unique problems existing in a single zone, being this the one with more occurrences. As previously mentioned, in a utopian scenario, the model would run for all 3713 different triples; however, several occurred so few times during the observed time that the model cannot detect patterns and learn relations effectively. Thus, the new guard referred to in 6.2 was added, so the process only continues to be applied as long as the best version of the model (within the three balance variations) has a ROC value greater than 60%.

With this new guard and applying the previously described approach, it is possible to train 1344 different trios; that is, 1344 models are generated with ROC values higher than 60%. Although it may initially seem minor, considering that there are 3713 unique trios, 1344 processed correspond to 98.92% of the 400 thousand entries of the dataset. Despite being numerous, the remaining unexamined trios have very few occurrences; more precisely, the maximum value of detected occurrences in these trios is 5. Therefore it is impossible to train such models, as not only the value is already low, but also 20% are separated for testing the model. After the process is done, analogous to Figure 26, a dataframe is generated with all the trios processed and the impact of each feature in its model, as shown in Figure 28.



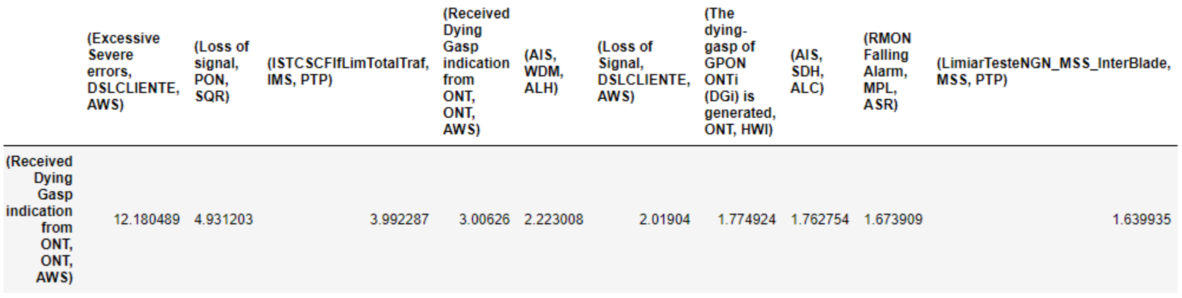
	SUB INTERFACE DOWN	LFD	Swap Usage	FILLED TO 100 % CAPACITY	GeNeric Alarm Indication Signal	The GPON ONT configuration recovery fails	PRT-TMNSB-99_PROXY_01	10.134.53.111:7443 SCM_APP_TEST_POF
	BIP	WDM	M2M	CD	WDM	CEM	M2M	ITS
	CPN	ALH	ALMMNG	EWS	ALH	HWI	STH	NNM
(Loss of Signal, DSLCLIENTE, AWS)	8.458688e-06	6.582618e-06	0.000012	0.000008	0.000075	2.199970e-05	0.000007	0.000
(Received Dying Gasp indication from ONT, ONT, AWS)	1.842009e-08	6.294118e-07	0.000000	0.000022	0.000022	1.192935e-05	0.000009	0.000
(Loss of Power, DSLCLIENTE, AWS)	1.260387e-05	0.000000e+00	0.000000	0.000124	0.000008	1.551432e-06	0.000005	0.000
(The dying-gasp of GPON ONTi (DGi) is generated, ONT, HWI)	1.297985e-05	0.000000e+00	0.000000	0.000147	0.000435	0.000000e+00	0.000000	0.000
(USER PLANE PATH FAULT, RAN, U2K)	2.917363e-05	0.000000e+00	0.000000	0.000000	0.000033	3.858646e-07	0.000000	0.000
...	...	...	...	...	...	...	...	
(ACCG - External I F Device Problem, RMC, NMG)	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000

1344 rows × 3713 columns

Figure 28: Result of applying Feature Importance to the 1344 processed models

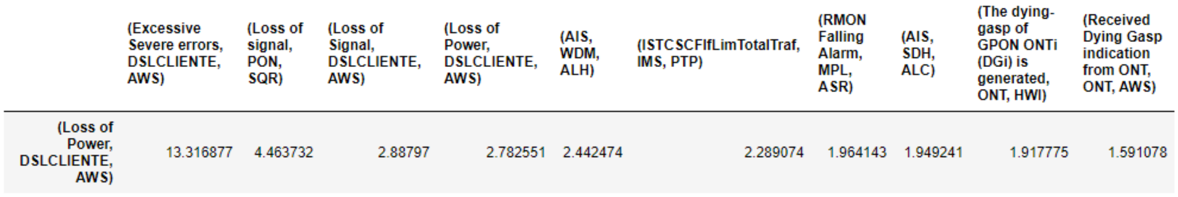
Once again, as shown in the figure above, most columns have little values, showing almost 0 correlation with triple under analysis. To this end, the filtering process is applied once again, keeping for each trio only the top

10 features with the most impact on the model, as can be seen in Figs. 29 to 31. Analyzing these results shows that although the three triples are distinct, the root cause of failure among the 3 is the same, thus pinpointing the impact that such triple has on the entire system.



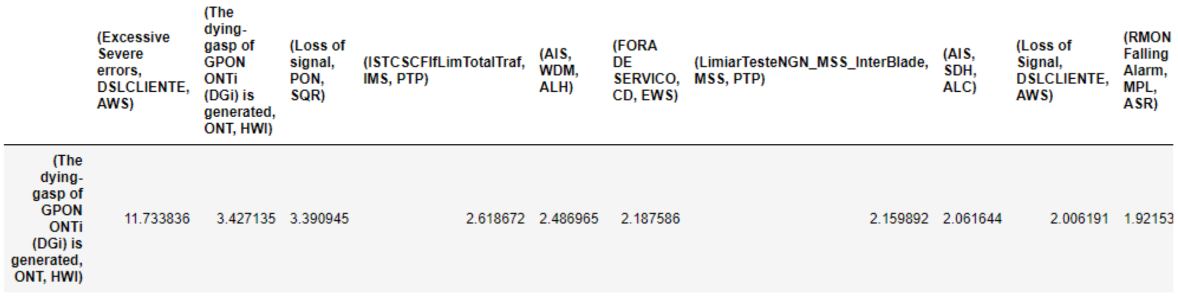
	(Excessive Severe errors, DSLCLIENTE, AWS)	(Loss of signal, PON, SQR)	(ISTCSCFIfLimTotalTraf, IMS, PTP)	(Received Dying Gasp indication from ONT, ONT, AWS)	(AIS, WDM, ALH)	(Loss of Signal, DSLCLIENTE, AWS)	(The dying-gasp of GPON ONTi (DGi) is generated, ONT, HWI)	(AIS, SDH, ALC)	(RMON Falling Alarm, MPL, ASR)	(LimiarTesteNGN_MSS_InterBlade, MSS, PTP)
(Received Dying Gasp indication from ONT, ONT, AWS)	12.180489	4.931203	3.992287	3.00626	2.223008	2.01904	1.774924	1.762754	1.673909	1.639935

Figure 29: Top10 feature importance of the model (Received Dying Gasp indication from ONT, ONT, AWS)



	(Excessive Severe errors, DSLCLIENTE, AWS)	(Loss of signal, PON, SQR)	(Loss of Signal, DSLCLIENTE, AWS)	(Loss of Power, DSLCLIENTE, AWS)	(AIS, WDM, ALH)	(ISTCSCFIfLimTotalTraf, IMS, PTP)	(RMON Falling Alarm, MPL, ASR)	(AIS, SDH, ALC)	(The dying-gasp of GPON ONTi (DGi) is generated, ONT, HWI)	(Received Dying Gasp indication from ONT, ONT, AWS)
(Loss of Power, DSLCLIENTE, AWS)	13.316877	4.463732	2.88797	2.782551	2.442474	2.289074	1.964143	1.949241	1.917775	1.591078

Figure 30: Top10 feature importance of the model (Loss of Power, DSLCLIENTE, AWS)



	(Excessive Severe errors, DSLCLIENTE, AWS)	(The dying-gasp of GPON ONTi (DGi) is generated, ONT, HWI)	(Loss of signal, PON, SQR)	(ISTCSCFIfLimTotalTraf, IMS, PTP)	(AIS, WDM, ALH)	(FORA DE SERVICO, CD, EWS)	(LimiarTesteNGN_MSS_InterBlade, MSS, PTP)	(AIS, SDH, ALC)	(Loss of Signal, DSLCLIENTE, AWS)	(RMON Falling Alarm, MPL, ASR)
(The dying-gasp of GPON ONTi (DGi) is generated, ONT, HWI)	11.733836	3.427135	3.390945	2.618672	2.486965	2.187586	2.159892	2.061644	2.006191	1.92153

Figure 31: Top10 feature importance of the model (The dying-gasp of GPON ONTi(DGi) is generated, ONT, HWI)

Analogous to the diagram in Figure 23, a graph with the ROC and accuracy values of each model was developed to assess the quality concerning the metrics of the models. However, due to the very high number of processed triples, it becomes impossible to correctly perceive them in a bar chart. Thus, this information will be divided into two graphs for better analysis: Figure 32 shows the ROC and ACC of the top20 models and Figure 33 shows the accuracy, in orange 33a, and the ROC, in blue 33b, of the 1344 models. Despite of visually appearing as a continuous line, the graph is composed by points, where X represents the trio processed and Y the ROC and ACC).

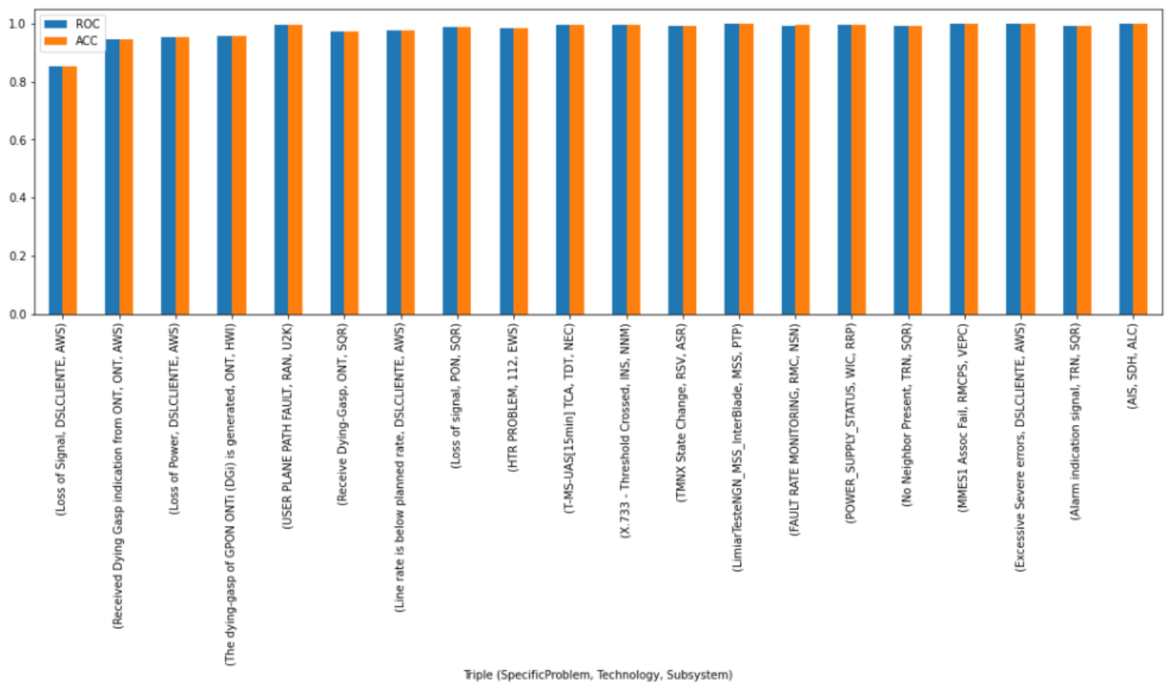
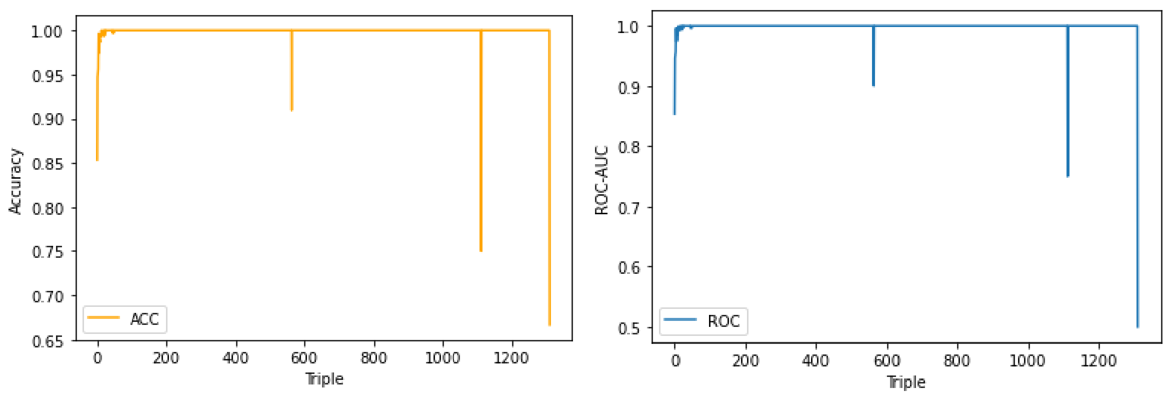


Figure 32: ROC-AUC and Accuracy metrics for the top20 models



(a) Accuracy of the 1344 processed models

(b) ROC-AUC of the 1344 processed models

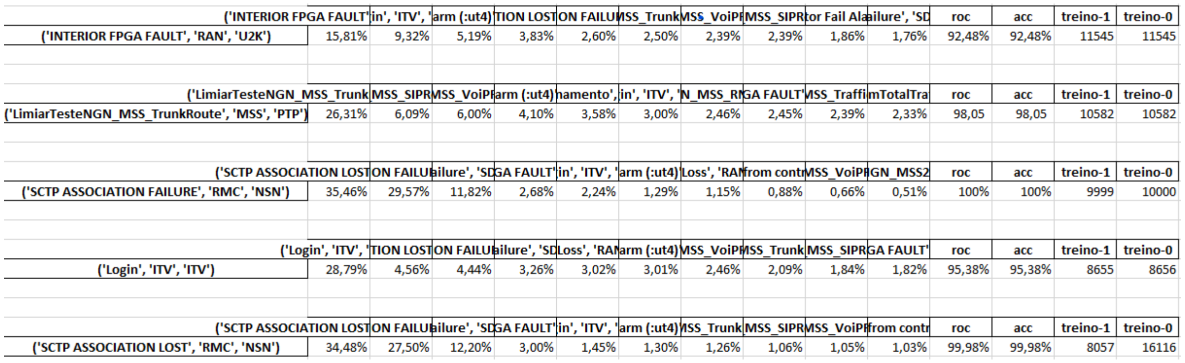
Figure 33: Metrics obtained for every model processed

These results were obtained by applying the approach explained in section 5.3 on the machine with the specifications referenced in Table 19, having an average execution time (average between 5 distinct executions) of 1h36 minutes.

6.3.3 *All Problems on All Zones*

After the success of analyzing a single trio and the subsequent expansion to all the different trios of a single zone, it is once again possible to expand the approach test scenario. So, the approach will now be applied to all 55 zones of the country, thus being the broadest test scenario possible. Moreover, this expansion allows, in case of success, to maximize the extrapolation of system-wide knowledge and relationships.

Since now the training and testing process is applied cyclically between all zones, a subsequent manual analysis of the results becomes unfeasible (also because between iterations, all variables have to be freed and/or reset, thus avoiding cycle stops due to lack of RAM).Thus, at the end of each iteration of the zones cycle, the filtered results (top 10) of each trio are saved for excell, as can be seen in Figure 34, together with the ROC and ACC obtained, in order to understand the significance of the results, and the number of positive and negative inputs used for training, in order to understand the degree of "depth" of the training phase and the veracity of the results. Thus, at the end of the process, the folder with the 54 files of the zones grows to 110 files, being the 54 new ones the results for each individual zone.



											roc	acc	treino-1	treino-0
	('INTERIOR FPGA FAULT'	in', 'ITV', '	arm (:ut4)	TION LOST	ON FAILU	MSS_Trunk	MSS_VoiP	MSS_SIPR	tor Fail Ala	ailure', 'SD	roc	acc	treino-1	treino-0
('INTERIOR FPGA FAULT', 'RAN', 'U2K')	15,81%	9,32%	5,19%	3,83%	2,60%	2,50%	2,39%	2,39%	1,86%	1,76%	92,48%	92,48%	11545	11545
	('LimiarTesteNGN_MSS_Trunk	MSS_SIPR	MSS_VoiP	arm (:ut4)	hamento',	in', 'ITV', '	N_MSS_R	GA FAULT'	MSS_Traffi	mTotalTra	roc	acc	treino-1	treino-0
('LimiarTesteNGN_MSS_TrunkRoute', 'MSS', 'PTP')	26,31%	6,09%	6,00%	4,10%	3,58%	3,00%	2,46%	2,45%	2,39%	2,33%	98,05	98,05	10582	10582
	('SCTP ASSOCIATION LOST	ON FAILU	ailure', 'SD	GA FAULT'	in', 'ITV', '	arm (:ut4)	Loss', 'RAN	from contr	MSS_VoiP	GN_MSS2	roc	acc	treino-1	treino-0
('SCTP ASSOCIATION FAILURE', 'RMC', 'NSN')	35,46%	29,57%	11,82%	2,68%	2,24%	1,29%	1,15%	0,88%	0,66%	0,51%	100%	100%	9999	10000
	('Login', 'ITV', '	TION LOST	ON FAILU	ailure', 'SD	Loss', 'RAN	arm (:ut4)	MSS_VoiP	MSS_Trunk	MSS_SIPR	GA FAULT'	roc	acc	treino-1	treino-0
('Login', 'ITV', 'ITV')	28,79%	4,56%	4,44%	3,26%	3,02%	3,01%	2,46%	2,09%	1,84%	1,82%	95,38%	95,38%	8655	8656
	('SCTP ASSOCIATION LOST	ON FAILU	ailure', 'SD	GA FAULT'	in', 'ITV', '	arm (:ut4)	MSS_Trunk	MSS_SIPR	MSS_VoiP	from contr	roc	acc	treino-1	treino-0
('SCTP ASSOCIATION LOST', 'RMC', 'NSN')	34,48%	27,50%	12,20%	3,00%	1,45%	1,30%	1,26%	1,06%	1,05%	1,03%	99,98%	99,98%	8057	16116

Figure 34: Example of the excel file generated after the processing of a zone

To better analyze the approach's success in expanding to all country zones, a graph was developed with the number of trios processed per zone, as shown in Figure 35 in orange. However, since not all zones have the same number of problems within the same time frame, as shown in Figure 10, it was necessary to add a second metric in blue to the graph to understand what percentage of the dataset was processed. As shown in the above figure, almost the entire dataset of the country was correctly processed: 48 out of 54 zones with more than 95% alarms processed and 41 with more than 99%. Overall, 95% of the entire dataset for all zone intervals between May 1st and May 16th was correctly processed. Thus, it is possible to conclude that the approach was a success, and it was possible to extrapolate the maximum amount of information from the data provided by the TO, thus having fulfilled the objective proposed in the thesis.

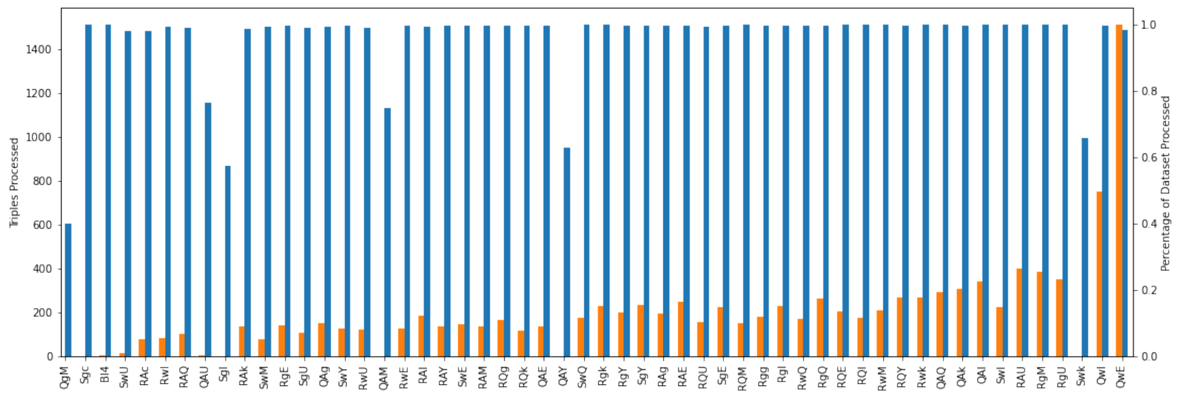


Figure 35: Number of triples processed per zone (orange) and the respective percentage they represent in the total dataset (blue)

6.4 SUMMARY

In this chapter, the technical specifications of the machine used during the development of this project and the automation process developed were exposed to apply the final approach to all areas and problems in the country, thus maximizing the knowledge possible to extrapolate from the system.

Additionally, the results for each of the three different levels of granularity are also exposed: one problem, one zone, and all the problems on all the zones; in order to understand in more detail the results obtained and what percentage of the data it was possible to process.

7

CONCLUSIONS AND FUTURE WORK

This research project arose from the possibility of applying ML techniques to Altice Labs' Alarm Manager tool. The main goal was to extrapolate the relationship between different types of alarms in order to have a better understanding of the impact of a failure on the entire system, thus easing and helping the process of manual implementation of RCA rules.

During this dissertation, the whole development process of the project was presented, from the treatment and pre-processing of data received by a TO, to the creation and implementation of an approach capable of absorbing such data variability and effectively learning the existing relationships between alarm instances, to the results obtained during the whole refinement process. Thus, it becomes possible for the reader to closely follow the entire developing process and the decisions taken throughout the project.

In more detail, the first step involved evaluating the state of the art techniques used in similar problems since, through the analysis and interpretation of the solutions exposed in the literature, it was possible to better understand the problem and how to tackle it. Then, the creation of a new structured dataset was essential since the data provided for the project development was taken directly from the Alarm Manager tool database, containing many incongruences and irrelevant information to the project at hand. So, all unnecessary information was dropped, removing outdated and deprecated fields, and through the dataset purification, all initial entropy was removed, becoming possible to pass to a development stage.

After implementing and testing different approaches, one was found to have a higher learning capability, this being based on the proposal on [26], which tackles a simplified RCA scenario showcasing the possibility of the development of a model without using the specific fields of each alarm, which allows a better perception of the causality effect among alarms, based only on their occurrence. However, numerous adjustments and enhancements had to be made in order to fit the Altice Labs Alarm Manager scenario best, solving the issues that originated from this being a broader and real-life problem with big data. The new proposed approach allowed to maximize the valuable knowledge possible to draw from the results, which is essential for a later phase of manual creation of correlation rules.

Finally, the results obtained during the intensive testing phase, applying the final approach in all zones for all problems, were exposed. These results are crucial to the project, not only for being its primary goal but also for confirming previously known relationships (hence proving the truthfulness of the results) and ascertain new ones (providing new helpful information to the system administrators to create more specific RCA rules). Thus, the implemented ML approach was considered a success, as it allowed not only to achieve the goal outlined initially, but it also opened a window to an even more exciting and complex scenario, which will be followed after the end of this project.

7.1 MAIN CONTRIBUTIONS

The project developed and described throughout this dissertation focused on the analysis and exploration of actual relations between alarms of real equipment from a TO and controlled by Alarm Manager. It is also essential to take into account that, in this TO and at this moment, Alarm Manager processes about 4 million daily alarms with a forecast of expansion to more than 10 million in a few years so. For this reason, its manual processing is entirely impossible. Thus any alarms' hierarchy information that can be extracted and empirically proven is exceptionally beneficial because it supports the system administrators in the creation of proper and previously unknown RCA rules.

The development of this project was considered a success, as it was not only possible to discover the relationship between thousands of problems in 55 different areas, but it was also able to take into consideration the technology used by the equipment and its supplier, which allowed for a new and more thorough perspective of the hierarchy and relationship between failures. Additionally, it was possible to elaborate excel reports for each one of the 55 different zones, including all the relations between all the trios of (*specificProblem*, *Technology*, *Subsystem*), the impact percentage value of its top10 at feature importance level, alongside the metrics obtained by the model, for a detailed insight into the veracity of the results.

In this way, this project outcome has real impact, because now, through the application of machine learning techniques on the causality matrix (thus implying that all discoveries are through patterns detected by the failure history), immense previously unknown relationships were discovered, allowing the generation of new RCA rules and reducing the manual processing currently performed by system administrators.

7.2 FUTURE WORK

As in all projects, having reached an end, it becomes possible to look back and question and/or identify aspects in which the work could be improved or continued. In this particular case, and for all the reasons previously mentioned throughout the dissertation, I believe that the ideal scenario for the proposed challenge was reached. However, during the testing phase of the approaches considered, specifically, the one that uses the Cartesian product between the problem and the equipment in which it occurred, a glimpse of a possible different path to follow with a more ambitious objective emerged.

While in the scope of this master thesis, the proposed goal was the detection of correlations between alarms, the approach in Section 5.1 revealed the possibility of, instead of using the ML models results for a posterior manual RCA rule definition phase, the ML models would automatically be able to identify for each problem which was the precedent that triggered it and, in a cascade of failures scenario, identifying which is the real source failure that has to be solved. Therefore, it can be considered that, through this successful expansion, an administrator would never have to manually define rules again, as ML would automatically do this process which furthermore represents the highest possible aggregation scenario, considering the outstanding capabilities of detecting patterns provided by ML techniques.

After the delivery of this project and, since my work will continue at Altice Labs, this new path will be explored to develop a fundamental tool that would run on top of the Alarm Manager and would be able to receive an instance of an alarm sent by a device and add, in real-time, which precedent alarm triggered it. This tool would be highly beneficial since, in its successful scenario, when series of faults occur, all stemming from the same cause, instead of the system administrator having to deal with them all sequentially, he would see them all aggregated on the alarm instance source. Thus, only one instance would have to be fixed to solve numerous problems, which would otherwise be much more time consuming for the administrator, hence providing a massive reduction in dimensionality, maximizing the systems' efficiency.

BIBLIOGRAPHY

[1] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, “Machine learning on big data: Opportunities and challenges,” *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[2] T. O. Ayodele, “Types of machine learning algorithms,” *New advances in machine learning*, vol. 3, pp. 19–48, 2010.

[3] F. Provost and R. Kohavi, “Glossary of terms,” *Journal of Machine Learning*, vol. 30, no. 2-3, pp. 271–274, 1998.

[4] R. Caruana and A. Niculescu-Mizil, “An empirical comparison of supervised learning algorithms,” in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161–168.

[5] S. B. Kotsiantis, I. Zaharakis, P. Pintelas *et al.*, “Supervised machine learning: A review of classification techniques,” *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[6] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep clustering for unsupervised learning of visual features,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *Unsupervised Learning*. New York, NY: Springer New York, 2009, pp. 485–585.

[8] O. Chapelle, B. Scholkopf, and A. Zien, Eds., “<emphasis emphasistype="bold">semi-supervised learning</emphasis> (chapelle, o. et al., eds.; 2006) [book reviews],” *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[10] Simhyeonju and Hyunjulie, “Machine learning studying roadmap,” 2018. [Online]. Available: https://medium.com/hyunjulie/machine-learning-studying-roadmap-8596b6571f8a

[11] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, “Random forests and decision trees,” *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 272, 2012.

[12] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, “An introduction to decision tree modeling,” *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 18, no. 6, pp. 275–285, 2004.

[13] R. C. Barros, M. P. Basgalupp, A. C. De Carvalho, and A. A. Freitas, “A survey of evolutionary algorithms for decision-tree induction,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 3, pp. 291–312, 2011.

[14] M. El Habib Daho, N. Settouti, M. El Amine Lazouni, and M. El Amine Chikh, “Weighted vote for trees aggregation in random forest,” in *2014 International Conference on Multimedia Computing and Systems (ICMCS)*, 2014, pp. 438–443.

[15] A. Adadi and M. Berrada, “Peeking inside the black-box: a survey on explainable artificial intelligence (xai),” *IEEE access*, vol. 6, pp. 52 138–52 160, 2018.

[16] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, “Xai—explainable artificial intelligence,” *Science Robotics*, vol. 4, no. 37, 2019.

[17] W. Samek and K.-R. Müller, “Towards explainable artificial intelligence,” in *Explainable AI: interpreting, explaining and visualizing deep learning*. Springer, 2019, pp. 5–22.

[18] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, “From local explanations to global understanding with explainable ai for trees,” *Nature machine intelligence*, vol. 2, no. 1, pp. 56–67, 2020.

[19] W. Du and Z. Zhan, “Building decision tree classifier on private data,” 2002.

[20] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[21] J. Huang and C. Ling, “Using auc and accuracy in evaluating learning algorithms,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[22] G. Pal, X. Hong, Z. Wang, H. Wu, G. Li, and K. Atkinson, “Lifelong machine learning and root cause analysis for large-scale cancer patient data,” *Journal of Big Data*, vol. 6, no. 1, pp. 1–29, 2019.

[23] A. Lokrantz, E. Gustavsson, and M. Jirstrand, “Root cause analysis of failures and quality deviations in manufacturing using machine learning,” *Procedia CIRP*, vol. 72, pp. 1057–1062, 2018.

[24] M. Giollo, A. Lam, D. Gkorou, X. L. Liu, and R. van Haren, “Machine learning for fab automated diagnostics,” in *33rd European Mask and Lithography Conference*, vol. 10446. International Society for Optics and Photonics, 2017, p. 104460O.

[25] S. Azimi and C. Pahl, “Root cause analysis and remediation for quality and value improvement in machine learning driven information models.” in *ICEIS (1)*, 2020, pp. 656–665.

[26] J. M. N. Gonzalez, J. A. Jimenez, J. C. D. Lopez *et al.*, “Root cause analysis of network failures using machine learning and summarization techniques,” *IEEE Communications Magazine*, vol. 55, no. 9, pp. 126–131, 2017.

[27] R. Costa, N. Cachulo, and P. Cortez, “An intelligent alarm management system for large-scale telecommunication companies,” in *Portuguese Conference on Artificial Intelligence*. Springer, 2009, pp. 386–399.

[28] C. Kim, V. B. Mendiratta, and M. Thottan, "Unsupervised anomaly detection and root cause analysis in mobile networks," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2020, pp. 176–183.

[29] K. Zhang, M. Kalander, M. Zhou, X. Zhang, and J. Ye, "An influence-based approach for root cause alarm discovery in telecom networks," in *International Conference on Service-Oriented Computing*. Springer, 2020, pp. 124–136.

[30] P. Fournier-Viger, G. He, M. Zhou, M. Nouioua, and J. Liu, "Discovering alarm correlation rules for network fault management," in *International Conference on Service-Oriented Computing*. Springer, 2020, pp. 228–239.

[31] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proceedings of the 2015 Internet Measurement Conference*, 2015, pp. 211–224.

[32] L. Maccari and A. Passerini, "A big data and machine learning approach for network monitoring and security," *Security and Privacy*, vol. 2, no. 1, p. e53, 2019.

[33] R. Harper and P. Tee, "The application of neural networks to predicting the root cause of service failures," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 953–958.

[34] R. M. A. Velásquez and J. V. M. Lara, "Root cause analysis improved with machine learning for failure analysis in power transformers," *Engineering Failure Analysis*, vol. 115, p. 104684, 2020.

[35] M. Dondo, P. Mason, N. Japkowicz, and R. Smith, "Network event correlation using unsupervised machine learning algorithms," Defence Research and Development Canada Ottawa (Ontario), Tech. Rep., 2006.

[36] A. Labs, "Alarm manager about altice labs nossis one fault management solution," 2021, available online. [Online]. Available: https://www.alticelabs.com/content/products/BR_AlarmManager_ALB_EN.pdf

[37] ——, "Alarm manager manual de utilização v9.9," June 2020, internal Document Accessed in October 2020.

[38] J. Brownlee, "Machine learning mastery with python," *Machine Learning Mastery Pty Ltd*, vol. 527, pp. 100–120, 2016.

[39] P. Documentation, "Pandas dataframe append method - api reference," Accessed in 2020, available online. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.append.html

[40] W. McKinney and P. Team, "Pandas-powerful python data analysis toolkit," *Pandas—Powerful Python Data Anal Toolkit*, vol. 1625, 2015.

[41] C. A. R. Hoare, “Quicksort,” *The Computer Journal*, vol. 5, no. 1, pp. 10–16, 01 1962. [Online]. Available: https://doi.org/10.1093/comjnl/5.1.10

[42] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O’Reilly Media, Inc.", 2012.

[43] D. Bhamare and P. Suryawanshi, “Review on reliable pattern recognition with machine learning techniques,” *Fuzzy Information and Engineering*, vol. 10, no. 3, pp. 362–377, 2018.

[44] A. Richter, “Random forest on gpus: 2000x faster than apache spark - towards data science,” 2020, available online. [Online]. Available: https://towardsdatascience.com/random-forest-on-gpus-2000x-faster-than-apache-spark-9561f13b00ae

[45] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, “Explainable ai for trees: From local explanations to global understanding,” *arXiv preprint arXiv:1905.04610*, 2019.

[46] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, “Hardware for machine learning: Challenges and opportunities,” in *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2017, pp. 1–8.

[47] V. Kasavajhala, “Solid state drive vs. hard disk drive price and performance study,” *Proc. Dell Tech. White Paper*, pp. 8–9, 2011.