UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Autodispatcher: a fully automated failure analyzer

Daniela Domingos Vieira

Mestrado em Ciência de Dados

Trabalho de Projeto orientado por:
Prof. Doutor Nuno Ricardo da Cruz Garcia
Prof. Doutor Luís Manuel Rodrigues Guimarãis

2023

Acknowledgments

I would like to commence by expressing my gratitude to Professor Nuno Garcia and to Professor Salvatore Signorello, my thesis advisors at the Faculty of Sciences, University of Lisbon. Throughout this journey, Professor Garcia and Professor Signorello have consistently demonstrated availability to address my inquiries and resolve any challenges I encountered, propelling me to strive for excellence in my work.

Additionally, I wish to extend my gratitude to my co-supervisor, Luís Guimarãis, from NOS. Despite shouldering a demanding workload, Luís has provided unwavering support throughout every phase of this thesis, always displaying empathy, accessibility, motivation, and appreciation. I extend my heartfelt thanks for your support and encouragement throughout this journey, as your contributions have played a pivotal role in the success of this thesis.

Last but not least, I want to express my deep gratitude to my entire family, in particular to my parents, who have always supported me and believed in me, and to my grandparents, who have always encouraged me to want more and do better. I cannot fail to mention my closest friends, who provided me with incredible support throughout the process. Your support was invaluable.

The realization of this thesis would have been impossible without the contributions of each and every one of you. I express my heartfelt gratitude to all mentioned individuals.

To my parents.

Resumo

Com o intuito de minimizar o impacto de possíveis falhas nos ativos da sua infraestrutura de telecomunicações em Portugal, a NOS está empenhada em aprimorar a eficiência dos atuais processos de decisão liderados por operadores humanos. Atualmente, uma equipa de especialistas seleciona manualmente as equipas de resposta para lidar com as falhas ocorridas, com base num conjunto pré-compilado de políticas para classificar os alarmes.

No entanto, esta abordagem apresenta duas principais desvantagens que afetam diretamente o tempo de resposta da empresa, após um alarme ser acionado. A primeira é o facto de nem todos os alarmes, após a sua ocorrência, serem imediatamente analisados pela equipa de especialistas, o que aumenta o tempo de resposta geral. Além disso, apenas os alarmes conhecidos são facilmente identificados pela equipa da empresa. Alarmes não classificados pela equipa são difíceis de examinar e podem, inclusive, levar à escolha da equipa de resposta errada para resolver o problema técnico no local.

Por isso, e de maneira a melhorar a capacidade da empresa na resolução das falhas técnicas, a NOS pretende aproveitar técnicas baseadas em Machine Learning (ML) para auxiliar a equipa de especialistas a selecionar a equipa de resposta mais adequada para lidar com um problema técnico ocorrido. O sistema é alimentado com uma grande quantidade de dados históricos de falhas e respostas correspondentes, permitindo-lhe aprender e identificar padrões complexos nos alarmes emitidos pela infraestrutura.

Para alcançar esse objetivo, um dos principais desafios reside na elevada dinâmica dos alarmes emitidos pela vasta e diversificada infraestrutura da empresa. Neste projeto, exploramos o design de um analisador de falhas totalmente automatizado, convergindo para quatro etapas de decisão, incluindo o método de similaridade de strings e uma rede neuronal, denominado **Autodispatcher**.

O **Autodispatcher**, funcionando como uma árvore de decisão, é capaz de analisar e processar eficientemente os alarmes emitidos pela infraestrutura em tempo real. Este analisador de falhas irá ser capaz de prever a equipa mais adequada para um determinado problema com base no nome e na natureza do alarme e tais previsões ajudarão a equipa de especialistas a identificar com mais facilidade as equipas de resposta capazes de lidar com novos tipos de alarmes. O Autodispatcher funcionará assim como auxiliar do processo humano de tomada de decisão e empregará, como primeira etapa, um conjunto de regras, as chamadas regras temporárias, que constituem uma lista de verificação para equipamentos específicos que estão sob observação constante a pedido de qualquer equipa, quer seja por certos equipamentos estar em desenvolvimento ou em rollout

(introdução de um novo equipamento). De seguida, o Autodispatcher empregará a segunda etapa composta pelas regras inferidas constituídas por regras baseadas no conhecimento e na experiência da equipa de especialistas, contendo apenas alarmes já conhecidos pela empresa. A lista destas regras consiste num conjunto de strings e na sua equipa respetiva já associada, e caso os alarmes contiverem qualquer uma dessas strings, o processo na árvore de decisão acaba e é devolvida a equipa correspondente à string contida no nome do alarme. Por exemplo, strings como "NOKIA", "ERICSSON" e "HUAWEI" estão presentes nestas regras, pois, após diversos testes por parte da equipa de especialistas, observou-se que todos os alarmes que contém estas strings estão relacionados à equipa de Conectividade Móvel. Adicionalmente, o Autodispatcher conterá também a análise de frequência de strings, que, antes da rede neuronal, calculará a menor distância entre cada alarme que chega à infraestrutura e um conjunto de substrings pré-geradas. Nesta etapa da árvore de decisão, o nome do alarme é comparado a um conjunto de substrings pré-geradas, previamente associadas à respetiva equipa de resposta. Como resultado, identifica-se a substring mais semelhante dentro deste conjunto bem como a sua equipa de resposta associada. Por fim, o último componente do **Autodispatcher** será então a rede neuronal e esta é treinada para identificar correlações entre diferentes tipos de alarmes e as equipas de resposta mais adequadas para lidar com cada situação. Isso permite que o sistema faça recomendações precisas e fiáveis à equipa de especialistas encarregada de selecionar as equipas de resposta. Tanto o treino e o teste do modelo foram realizados através da ferramenta TensorFlow, que é uma plataforma de código aberto para o design e implementação de modelos de *Machine Learning*. O TensorFlow permite assim a criação e o treino de redes neuronais usando dados de entrada para detectar padrões. Uma de suas maiores vantagens é a rapidez em gerar resultados a partir de um modelo treinado, não sendo necessário retreinar o modelo. Esta ferramenta foi escolhida para este projeto uma vez que, em primeiro lugar, como se trata de uma plataforma de código aberto, permite que qualquer pessoa o acesse livremente dentro da empresa, permitindo a qualquer pessoa instalar os trabalhos da plataforma e adaptá-la consoante as suas necessidades, desde adicionar novas funcionalidades, se necessário, ou otimizar o desempenho de um exemplo já existente. Em segundo lugar, dentro da empresa é a plataforma mais utilizada para Redes Neuronais. Existiram inclusive muitos workshops de treino no TensorFlow; portanto, como o projeto do Autodispatcher está para ser implementado na empresa, há mais pessoas para poder percebê-lo e/ou utilizá-lo e, posteriormente, potencializá-lo.

De maneira a garantir a eficiência e a eficácia do **Autodispatcher**, são realizadas atualizações regulares do modelo e retreinamento. Isso permite que o sistema se adapte às mudanças na infraestrutura e incorpore novos padrões de dados que possam surgir ao longo do tempo. Além disso, os operadores humanos desempenham um papel considerável na supervisão e validação das decisões do Autodispatcher, uma vez que têm a capacidade de rever e ajustar as recomendações do sistema, assumindo a decisão final.

O objetivo principal do **Autodispatcher** é, então, minimizar o tempo de resposta e otimizar a alocação de recursos. Ao automatizar o processo de seleção das equipas de resposta, a NOS pode melhorar significativamente a eficiência das suas operações. Isso não apenas reduz o impacto

das falhas na infraestrutura, mas também melhora a experiência global do cliente, garantindo resoluções atempadas para problemas técnicos.

Para alcançar o objetivo mencionado, foram realizados diversos testes a fim de determinar a melhor estrutura para a rede neuronal. Além disso, foram exploradas várias abordagens para mitigar potenciais problemas, como o overfitting. Essas abordagens incluíram a implementação de um conjunto de validação e a aplicação do método K-Fold Cross Validation. Após a realização de múltiplas avaliações, foi possível identificar que o modelo K-Fold Cross Validation apresentou um desempenho consistente e promissor em termos de precisão e capacidade de generalização durante as avaliações internas. Apesar de apresentar taxas de erro variáveis para diferentes classes de previsão, observou-se uma taxa de erro geralmente baixa para a maioria dessas classes. Portanto, recomenda-se a adoção do modelo baseado em K-Fold Cross Validation como a melhor escolha para o sistema Autodispatcher.

Em conclusão, o sistema **Autodispatcher** oferece uma solução abrangente para aprimorar o processo de classificação e resposta a alarmes na NOS. Ao utilizar técnicas de *Machine Learning* e integrar diversos tipos de regras, o sistema aprimora a capacidade da empresa de identificar rapidamente as equipas de resposta a falhas técnicas, reduzindo o tempo de resposta e aumentando a eficiência operacional. A eficiência é, assim, movida pela capacidade de aprendizagem contínua impulsionada por dados históricos e experiência humana acumulada. Desta forma, a parceria intrínseca entre as técnicas mencionadas anteriormente e a supervisão humana garante que as decisões do **Autodispatcher** sejam validadas e ajustadas quando necessário, mantendo a qualidade e a confiabilidade das operações.

Palavras-chave: Machine Learning, Redes Neuronais, Árvores de Decisão, Python, TensorFlow

Abstract

In order to minimize the impact of potential failures of assets within its telecommunications infrastructure in Portugal, NOS is interested in improving the efficiency of current decision processes which are led by human operators. Currently, an expert team manually selects response teams to deal with occurred failures, based on a pre-compiled set of policies to classify alarms.

However, NOS aims to leverage ML-based techniques to help the expert team select the most appropriate response team to deal with an occurred technical issue.

Towards that goal, one of the main challenges lies in the highly dynamic nature of the alarms issued by the company's large and diverse infrastructure. In this project, we explore the design of a fully automated failure analyzer based on neural networks, called *Autodispatcher*, to tackle this challenge.

The Autodispatcher framework will function as a decision tree, capable of efficiently analyzing and processing real-time alarms emitted by the infrastructure monitoring tools. It is designed to have four stages, including two stages which are the main focus of this work: a string frequency analysis method and a neural network approach for word analysis.

To ensure the efficiency and effectiveness of the Autodispatcher, regular model updates and retraining are conducted. This allows the system to adapt to changes in the infrastructure and incorporate new data patterns that may arise over time. Additionally, human operators play a considerable role in supervising and validating the Autodispatcher's decisions, because they have the ability to review and adjust the system's recommendations.

The main goal of the Autodispatcher is to minimize response time and optimize resource allocation. By automating the process of selecting response teams, NOS can significantly improve the efficiency of its operations. This reduces the impact of failures on the infrastructure and enhances the overall customer experience by ensuring timely resolutions to technical issues.

Keywords: Machine Learning, Neural Networks, Decision Tree, Python, TensorFlow

Contents

List of Figures

List of Tables

Chapter 1

Introduction

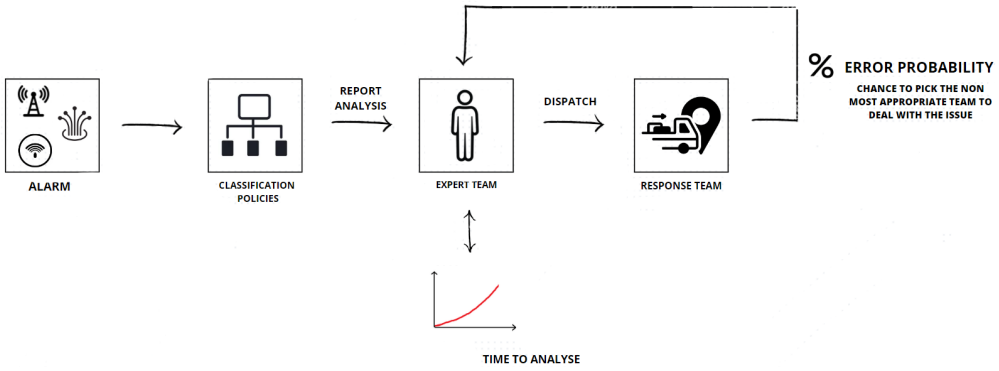
1.1 Motivation

NOS is a telecommunications company in Portugal and its main objective is to aim for maximum client satisfaction by ensuring efficiency and quality of services. NOS has an extensive portfolio of services, involving a considerable number of assets, such as cell phone towers, fiber optic retransmitters, routers, and many other technologies. To accomplish its goals, it is essential to monitor every single equipment to anticipate potential errors. The infrastructure can suffer sudden unexpected failures in any of its assets and, therefore, it is extremely important to reduce the response time to solve the related technical issue.

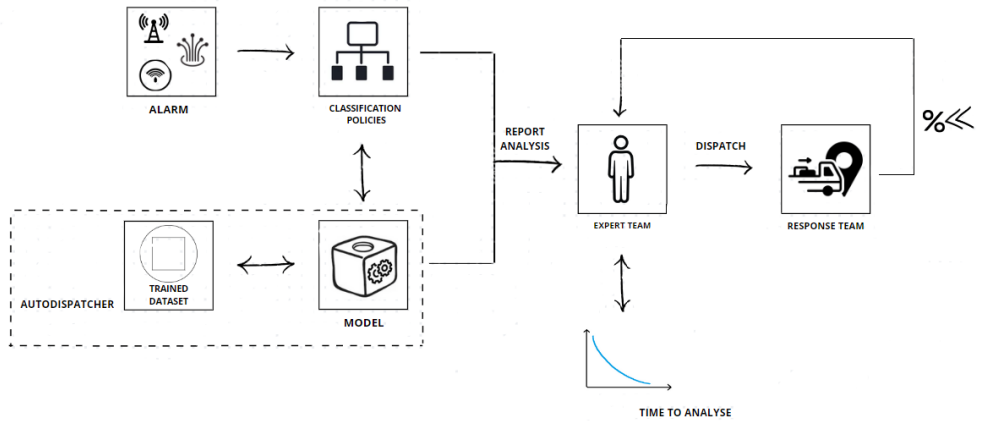
1.2 Problem Statement

Currently, as illustrated in Figure 1.1a, NOS has a dedicated team of people, experts in examining alarms issued by its infrastructure that manually analyze each alarm and identify the most suitable response team for solving the related technical failure. The alarm experts classify the alarms based on a pre-compiled set of classification policies, built by the company on a considerable number of past failure events over time.

The current approach has two main disadvantages, and both affect the company's response time. Firstly, not all alarms are immediately analyzed by the expert team when they occur, thus increasing the overall response time. Secondly, only known alarms are easily identified by the team of experts, while previously unclassified kinds of alarms are difficult to examine and may lead to the choice of the wrong response team to solve the technical issue on site. With regard to the first problem, additional time is often required to analyze and take decisions for previously unseen alarms. Yet, even known alarms usually require a non-negligible time to be examined and assigned to a team responsible for solving them. With regard to the second problem, as the analysis of the alarms is performed manually by a human operator, there is a certain probability that some alarms are overlooked and some others are erroneously classified, that is, the wrong team is sent to the site for solving the technical issue. Misclassification of previously unseen alarms can occur since the human operator does not hold any matching policies into the classification set and so



(a) Currently, the decision process is based on a manual analysis of pre-compiled classification policies, performed by a team of experts that classify the alarms to dispatch the response team most likely able to solve the occurred problem to the site.



(b) By introducing *Autodispatcher*, the decision process will be supported by an ML-based model that will automatically produce recommendations about response teams most likely able to solve occurred problems, reducing the analysis time and the probability of error of the expert team.

Figure 1.1: The current (a) and targeted (b) approach for dealing with alarms

must infer it only through the alarm description.

1.3 Objectives

This project aims to improve the company's ability to solve potential technical failures by introducing a new component, Autodispatcher, in the current decision process. Autodispatcher is an ML-based system component which aims to assist the expert team to classify previously unseen alarms, quickly classify familiar alarms, and overall increase the level of confidence in human decision-making. After deployment, Autodispatcher will analyze alarms in near real-time em-

ploying the set of rules it was trained to have and provide recommendations of the technical team (or teams) that should be selected to solve a technical failure, as illustrated in Figure 1.1b. If an alarm does not match any existing rule into that set, Autodispatcher can still infer the most likely team to address the issue.

The challenge of building an ML-model for a system like Autodispatcher lies in the amount and in the highly dynamicity of the alarms that the company's infrastructure may generate. Alarm names often change, and new types of alarms are regularly appearing, as a result of the company having a considerably dynamic pool of assets and constantly rolling out upgrades or new technology, like 5G technology.

The initial goal of the model is to relate two fields presents in a data sets built by the company: a short description of the alarm that occurred, called *summary*, and the team that solved the problem, called the *opening team*. The model should be able to predict the most suitable team for a certain problem based on the name and the nature of an alarm. Such predictions will help the expert team to more easily and precisely identify response teams able to deal with new kinds of alarms, i.e., assist the human decision-making process.

We opted for supervised learning techniques that can be retrained over any snapshot of the company's alarm reporting tool of its infrastructure, in a fast and understandable manner. In this way, given the name of a faulty component and the nature of the alarm, our model will be able to predict which team is the most appropriate to solve the related technical problem. As an outcome, a recommendation for a response team can significantly help tracking a problem at hand and thus reducing response time. Two supervised techniques are employed. The goal of the techniques is to relate the alarms fields **summary** and **openingteam**, which do not possess a direct relationship and, inclusively, are in distinct databases. The first technique developed is a model using neural networks [3] [22], through the TensorFlow software framework [26]. The second technique is a string distance approach. The latter is easily understandable by team members and, working by name familiarity, provides a certain degree of confidence to an unexperienced user. The techniques together with several other simple rules we will later explore provide a decision tree framework which will be Autodispatcher component.

1.4 Structure of the Document

This document is organized as follows: first, in Chapter 1 an introduction is made containing the motivation, the problem statement, and the objectives of this project. Chapter 2 introduces important concepts to consider for the project. In the following section, Chapter 3 describes the articles that were analyzed before and during the construction of Autodispatcher, more specifically during the implementation of the fourth block, the neural network. In Chapter 4 the data sets are presented, as well as the design, the workflow at a high level, including all the analysis and preprocessing, and the system components. Then, in Chapter 5 the results of our evaluation of the model are presented. And finally, the conclusions and the future work are described in Chapter 6.

Chapter 2

Background

In this chapter, we introduce the concepts that will be fundamental for the development of the work. We introduce the concept of neural networks in Section 2.1 and how they can assist us in making predictions for Autodispatcher. The section 2.2 mentions some types of loss functions and how their choice should vary depending on the problems to be solved and the model implemented. Next, in Chapter 2.3, we discuss different strategies to address overfitting, and some concepts to take into account for both multi-class classification, such as Autodispatcher, and for binary classification. We also provide a practical example involving four company teams to illustrate how metrics function within the company and how they are assessed based on their outcomes. We also introduce string similarity concepts in section 2.4, such as several types of distances between two strings, and how they can be used to build a prediction model.

2.1 Neural Networks

Neural networks [3] [22] are artificial intelligence techniques inspired by biological neural networks, based on networks of neurons in the human brain. They are used to identify patterns in large data sets, representing a significant aid in decision-making processes.

Similarly to the networks of neurons in a human brain, neural networks are composed of layers of interconnected nodes. They have an input layer that receives the input information, an output layer that produces the system's output, and several other hidden layers between these two. These hidden layers process and transform the input as it flows through the network, and the output from one layer is passed as input to the next layer.

The type of neural network used in this project is the Dense Neural Networks, which are also known as fully connected neural networks [12]. In this type of neural network, all nodes in a layer are connected to all the nodes in the previous layer. The output of each node is passed through an activation function[31] before being fed into the next layer. Initially, the input layer receives the input data which is usually a vector or a matrix. The input data is then passed through one or more hidden layers before reaching the output layer, which produces the final network output. Importantly, each hidden layer has its own sets of nodes and their activation functions.

The activation functions are mathematical functions applicable to the output of each neuron

in the neural network [31]. Activation functions allow non-linearity, that is, linear functions can only model linear relationships between the inputs and outputs, which limits the expressive power of neural networks. By introducing non-linearity, neural networks become able to model complex, non-linear relationships between inputs and outputs. Moreover, they play a crucial role in backpropagation as they allow loss function gradients for efficient calculation. Without them, the network output would be a linear combination of its inputs and the gradients would be constant, hindering the network training process.

There are many types of activation functions and the process of choosing them can have a significant impact on the performance of the network; therefore, their choice is a decisive step. For this project, two types of activation functions are used: RELU and Softmax.

The RELU activation function [31], in addition to its remarkable simplicity and efficiency, has the advantage of not saturating. Unlike other types of activation functions, RELU nodes can continue to learn and improve even for large inputs. Furthermore, this type of activation function ensures that gradients do not become too small during the backpropagation process, which would cause the network to prevent it from learning effectively.

Also, the SOFTMAX [31] activation function produces a discrete probability distribution over the classes, which means that the output values of the SOFTMAX layer can be interpreted as the probability of the input belonging to each of the classes. This becomes very useful in the case of this project since we want to assign an alarm name to any number of technical teams. Using a SOFTMAX creates a normalized output: all the outputs add up to 1, ensuring that the output can be interpreted as a discrete probability distribution over the involved classes. It is typically used in multi-class classification problems, presenting the great advantage of being able to deal with unbalanced classes. This way, it can continue to produce accurate predictions, even when the number of examples in each class is not equal.

There are other types of neural networks with a more complex topology, such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). If, on the one hand, CNNs [35] represent a type of neural network commonly used to classify images or computer vision tasks, on the other, RNNs [30] are used for tasks involving sequential data, such as time series, speech recognition, and language modeling.

Therefore, these neural networks are trained to process inputs through their layers. The training of neural networks consists of providing input and output layers. Throughout the training process, the model will compare the current output (output that the model predicted) with the correct output (output initially provided). Each time the neural network is trained, with some training data, an epoch is counted. Thus, the number of epochs is an important hyperparameter of this model since it defines the number of times the network was trained. At each epoch execution, the neural network leverages what it learned in previous epochs and consequently adjusts its layers.

Throughout the training process, inputs are transmitted over the network using direct propagation, and the network output is later compared to the desired output so that it is possible to calculate the loss.

The process of calculating the output of a neural network is known as direct propagation. Thus, during this process, the input data is fed into the first layer of the model and each subsequent layer processes the output of the previous layer until the final output is produced.

The loss, mentioned earlier, represents the difference between the network output and the expected output [14]. This calculation is called the Loss Function. This function is responsible for adjusting the model to the training data, thereby minimizing the aforementioned distance. It will be discussed in more detail further ahead, in section 2.2.

Nonetheless, after the network output and the expected output are compared, the inverse process is performed to the direct propagation called backpropagation [11]. This process calculates the loss function gradient relative to the network weights and bias starting at the output layer, moving backward until reaches the input layer. This gradient will later be used to update the two mentioned values in order to obtain the minimum Loss Function value.

In summary, the first step of the process is to calculate the error, that is, the distance between the network output and expected output through direct propagation. If the value obtained is minimal, the process ends, otherwise the hyperparameters are updated, through the backpropagation, and a new calculation of the distance is performed, until the minimum error is found. After this, the model is then ready to make predictions.

All these processes are repeated a few times until the loss function value is minimized to a satisfactory level, that is, the model manages the result closer to the target output. The weights and biases of each layer are adjusted throughout the training process to optimize the model output for a given set of inputs.

Weights and biases [13] are two important parameters of a model that will be adjusted during the training process to minimize the loss function. It is important to note that the model learns to make predictions owing to the adjustment of the parameters.

The weights represent the strength of the connections between neurons within the neural network, thus determining the influence of the network input on the output. Thus, the model adjusts these values to minimize the difference between the predicted outputs and desired outputs. It should be noted that when the weights of a neural network tend to be large, there is a possibility of overfitting the data because large weights allow the model to fit very well to the training data and fail; consequently, in its generalization, that is, fail in forecasts in unseen data.

Alternatively, bias represents a constant that is added to the equation of the calculation of the output to compensate for the result, showing how far the predictions were from the real values. This enables the model to make predictions even when the input values are zero. This value, like weights, was adjusted in the training phase to improve the accuracy of the predictions. Thus, through this parameter, it is possible to reduce the variance and consequently introduce flexibility to the model. In addition, note that a model with a high bias can eventually lead to underfitting of the target estimates. The model to possess such characteristics cannot capture the existing patterns, memorizing only, and, consequently, will lead the model to not correctly predict in unseen data, thus underfitting.

2.2 Loss functions

In this section, we describe some loss functions. The choice of the loss function depends on the type of problem being solved and the model being implemented.

The first loss function to be described is the Mean Squared Error (MSE) [24]. This function is used in regression problems and calculates the mean of the differences squared between the values predicted by the model and the expected values. The formula for calculating the MSE is given by:

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{\mathrm{n}} (y_{true_i} - y_{pred_i})^2$$

Where y_{true} represents the expected values, or the true values, y_{pred} the predictions made by the model and n denotes the total number of samples in the data set.

Moreover, there is also the Cross-Entropy Loss function [39]. This type of function measures the difference between the true class and predicted probabilities of all classes. In this way, each probability of the class predicted by the model is compared with the actual/desired class and then a score corresponding to the loss is calculated in order to penalize the probability based on its distance from the expected actual value. It is important to note that this penalty is logarithmic and will produce a high score for large differences close to 1, and a low score for small differences, tending to 0. The goal is again to minimize the loss, that is, get a cross-entropy loss value to tend to 0.

The formula for calculating the Cross-Entropy Loss is given by:

$$CrossEntropyLoss = -\sum_{i=1}^{\mathrm{n}} (y_{true_i} \cdot log(y_{pred_i}))$$

Here, y_{true} represents the true class (0 or 1), and y_{pred} the probability predicted by the positive class model (between 0 and 1).

This type of function can be used in both binary classification problems and multi-class classification problems.

Next, we will present the Binary Cross-Entropy Loss function and the Categorical Cross-Entropy Loss which are two specific cases of the Cross-Entropy Loss function.

In the Binary Cross-Entropy Loss function [27], as the name implies, it is used for binary classification problems, and calculates the cross entropy between the predicted values and desired/true values. Cross entropy is a measure that evaluates the degree of difference between two probability distributions. In this case, it will then measure the degree of difference between the probability distribution predicted by the model and the real probability distribution. The formula for calculating the Binary Cross-Entropy Loss is given by:

$$BinaryCrossEntropyLoss = -\frac{1}{n} \cdot \sum_{i=1}^{\mathrm{n}} (y_{true_i} \cdot log(y_{pred_i}) + (1 - y_{true_i}) \cdot log(1 - y_{pred_i}))$$

Similarly, y_{true} represents the true class of the sample, which can be 0 or 1, y_{pred} the probability predicted by the model that the sample belongs to class 1 (or positive class), $1 - y_{pred}$, the

probability predicted by the model that the sample belongs to class 0 (or negative class) and n the total number of samples in the data set. Note that the formula presented consists of the junction of two terms, one for the positive class, that is, $y_{true} = 1$, and another for the negative class, that is, $y_{true} = 0$.

If the model predicts a very high probability for the wrong class, the loss will be high. By contrast, if the model predicts a high probability for the correct class, the loss will be low. It is also important to note that the goal is always to minimize the value of the loss functions.

Another function is also the Categorical Cross-Entropy Loss function [4]. This function is used in classification problems with more than two classes, that is, in multi-class classification problems. It is important to note that the data under analysis must be categorical so that each instance belongs to a set of possible data and belongs to only one class.

Like the Binary Cross-Entropy, this function calculates the cross entropy between the predicted values and the actual values, i.e., measures the difference between the predicted probability distribution and the actual probability distribution. The predicted probability distribution is usually represented as a probability vector, where each element corresponds to the probability of the instance belonging to a given class. Furthermore, the real or true probability distribution is represented by a one-hot encoded vector, where the element corresponding to the true class is 1 and, all other elements are 0. The formula for this calculation is as follows:

$$CategoricalCrossEntropyLoss = -\sum_{i=1}^{\mathrm{n}}(y_{true_i} \cdot log(y_{pred_i}))$$

In this case, y_{true} represents a one-hot encoding vector that represents the true class (e.g., [0,1,0] for the second class), and y_{pred} is a probability vector predicted by the model for each class. In the case of probabilities, output values must be between 0 and 1, and their sum must be 1. Further, the loss is always non-negative and, equal to zero when the predicted probability distributions correspond perfectly.

Similarly, if the model predicts a very high probability for the wrong class, the loss is high, and if the model predicts a high probability for the correct class the loss is low.

The first loss functions presented are not good options for our project. On the one hand, MSE is used for regression problems assuming that the destination variable is continuous, which does not happen for categorical variables. The MSE also has the particularity of assigning equal weight for all errors, which could be a major problem when dealing with unbalanced classes. In contrast, the BCE function, as we saw earlier, is for binary classification problems, which is not the case for the model in question.

In TensorFlow, more specifically in Keras, a neural network library running on TensorFlow, we can also find the Sparse Categorical Cross Entropy [32]. This type of function, although it has the same responsibility as the Categorical Cross Entropy described above has some differences. Firstly, while the Categorical Cross Entropy function returns the average loss value over all samples, the Sparse Categorical Cross Entropy returns the sum of the loss values for all samples. Second, the Categorical Cross Entropy is formatted to receive as input the destination variable in a

one-hot encoded format, where each category is represented as a vector of binary sets. Conversely, Sparse Categorical Cross Entropy expects the target variable to be represented as integers, where each number corresponds to the index of the corresponding class.

Note that in one-hot encoding, a categorical feature with n different categories is represented as a binary vector of length n, where each vector has only one element set to one and all others are set to zero. Moreover, when a variable is represented by integers, each sample of the data set is associated with an integer value. This integer value represents the class label of that sample. This last type of encoding is often used when the order of the categories is irrelevant or when the number of categories is reduced.

Although there are several types of loss functions for text sorting, the Sparse Categorical Cross Entropy is the most suitable for Autodispatcher because it features the ability to handle problems of various classes, which is our scenario because the classes of the model will correspond to the various teams in the company. However, it also presents the ability to handle unbalanced classes, uncertainties, and sparse data. This last property is quite important because there can be quite a few in-text data. This means that many of the words in the data may be irrelevant to the sorting task. Thus, by assigning low weights for irrelevant words and high weights for relevant words, cross-entropy can handle this type of data.

2.3 Training optimizations

However, although the objective is to increase the capacity of the model, minimizing the distance between the network output and the expected output, i.e., the training error, there is the possibility of, with this decrease, increasing the risk of the model memorizing data. This event is known as overfitting. Overfitting [6] is characterized by attempting to fully adjust training data by memorizing the patterns present in the data as well as noise fluctuations. It thus presents a good performance in data training, and an extremely low training error, unlike the error presented in the test phase. This event occurs with some regularity, being thus more likely to occur when the models have high variance or when they are very complex, failing to create part of the relationships between the variables, or when the training data is not cleaned having too much "noise" or when there is an incorrect adjustment of the hyperparameter in the training phase. When the model is complex, one of the techniques used to improve its performance is to train the model with more data, to balance the relationship between the number of parameters and the number of training samples.

There are some techniques to solve this problem of overfitting memorization. These techniques are then used to estimate model performance on unseen data to prevent overfitting and to ensure that the model generalizes well for new data.

One of the techniques used is called the validation set [10]. To do this, it is necessary to initially divide the data set into three subsets: training, validation, and testing. In the first set, that is, in the training set, the model is trained and then evaluated in the validation set. In this second set, the performance of the model is evaluated (during the training phase) and it is from it that

it is possible to obtain an estimate of the performance of the model in unseen data. In this way, it is in this set that it is possible to observe if the model performance is the intended one or if hyperparameter adjustments are necessary. Finally, the last set is reserved for the final evaluation of the model. Note that this technique can result in underfitting if the model is simple and fails to capture all the patterns. This is because the training set is smaller owing to the initial division into three sets.

Another technique used to limit overfitting is k-fold cross-validation [19]. This technique consists of dividing the training set into k equal subsets, called folds. For each existing fold, the model is trained in k-1 folds and tested on the remaining fold. The process is repeated k times, since there are k Folds, as shown in Figure 2.1.

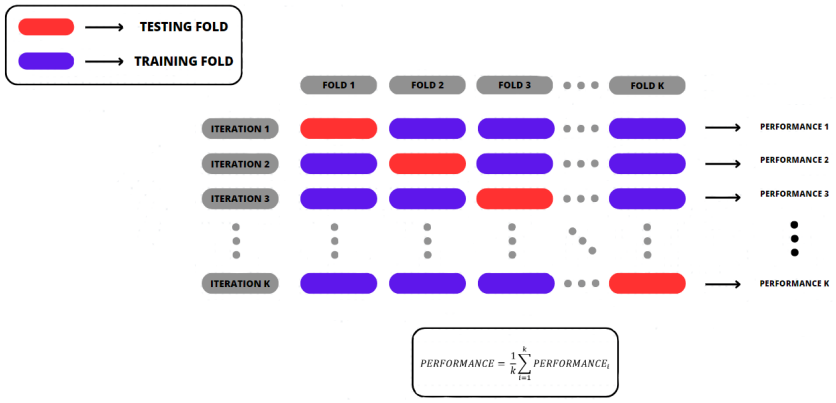


Figure 2.1: K-Fold Cross-Validation

Note that the test fold is cyclical, that is, in each iteration, the test fold is always different. Finally, the results of each fold are averaged to provide an overall performance metric.

In this way, this technique allows you to train and test the model k times on different subsets of training data and create an estimate of the model's performance on unseen data. One of the advantages of this technique is that it uses the available data in a very efficient way because each data point is used for training and testing at least once in one of the k iterations.

A key aspect of this technique is the choice of k. This k is initially used in order to divide the data set into k-equal subsets. Remember that in each iteration, the model is trained in k-1 Folds and evaluated in the rest.

Therefore, when k is a small value (e.g., 5), it means that the number of Folds is also a small number and, consequently, there will be a larger amount of data in each one. This will lead to the validation set being larger (and the training set being smaller), meaning that there will be a larger number of data that were not used to learn. This leads to variation as well as bias increase.

Contrary, when it comes to a higher k (e.g., 10), the number of Folds is also higher and, consequently, there will be a smaller amount of data in each one. This will lead to a smaller validation set (and a larger training set), leading to a decrease in variance (in the training set) and bias.

Also note that it is important to ensure that each fold contains a representative sample of

each class, especially when it comes to unbalanced classes, that is, classes that are not equally represented. Otherwise, the model will have difficulty accurately predicting the minority class, which has fewer examples than the others. This suggestion can be made using stratified k-fold cross-validation. During this approach, each fold contains approximately the same proportion of each class.

Another technique used to limit overfitting is the "Early Stopping" process [23], which can be used in conjunction with one of the techniques presented above. This form of regularization does not compromise the accuracy of the model and interrupts the process when parameter updates do not start to generate any improvements, that is, it provides information about how many iterations the model can perform without starting to adjust too much. In this way, this technique monitors the performance of the model for each season, stopping the training as soon as the performance of the model stops improving.

It is also possible to underfit [16] a model in the training process. In this scenario, the model presents a high error rate in the training phase and a low error rate in the test phase, thus failing to create relationships between the input and output variables. Unlike overfitting, underfitting is usually visible when the data are insufficient and can thus result in inaccurate predictions. This possibility can also be manifested when the model used is too simple for the data set, which can be the result of a model that needs more training or more input elements.

Similarly to other machine learning models, features and targets are two important concepts existing in neural networks. Features [25] are usually input variables to the machine learning model, for example, the input layer neurons in a neural network. Features are independent variables through which predictions are made by the model. Moreover, the targets are dependent variables and consist of the result that the model is trying to predict, that is, the expected output as a response to the input features.

Therefore, as previously mentioned and considering the concepts discussed so far, our model will be built over neural networks through TensorFlow, the software framework chosen to design and develop Autodispatcher in this project. TensorFlow [26] is an open-source platform for the design and deployment of machine learning models. Among other things, TensorFlow enables the creation and training of neural networks using input data to detect patterns. One of its greatest advantages is its speed in generating results from a trained model, not being necessary to retrain the model.

For the evaluation of Autodispatcher, we will consider standard evaluation metrics for multi-class classification models [20] since in the targeted scenario there exist several response teams alarms may be assigned to, namely, accuracy, precision, recall, and F1-score.

In fact, binary classification is not applicable because the teams (that will solve the problem) will correspond to the classes, and there are more than two.

In binary classification, there are only two classification classes, preferably one positive class and one negative class. In this type of classification, is used a matrix of confusion, with dimensions 2x2 (number of existing classes). There are standard evaluation metrics for classification models

[34]: accuracy, precision, recall, and F1-score.

Within these metrics, there are still concepts to consider within the case study, namely the definition of TP, FP, FN, and TN.

On the one hand, TP means true positives and, as the name implies, represents the proportion of positive cases that were correctly predicted by the model (and that, in this case, will represent cases where the model predicts correctly the team).

On the other hand, FP means false positives and represents the positive cases that were incorrectly predicted by the model (that is, cases where the model incorrectly predicts the team, that is, suggests a team that does not solve the problem).

Similarly, TN means true negatives and portrays the negative cases that were correctly predicted by the model. Finally, FN which are false negatives and treats negative cases that were correctly predicted by the model; in other words, positive cases that were not identified by the model.

The accuracy is the ratio of correctly predicted observations to the total number of observations. Normally, it's only used when the data sets are symmetrical, that is, if the data sets are balanced, which means that they have a similar class distribution, or better, if the value of false positives (FP) is identical to the value of the false negatives (FN). The higher the rate, the better the model under study.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$

Precision is the ratio between true positives (TP), that is, the proportion of correctly predicted positive observations in the universe of all positives detected by the model. It is value varies between 0 and 1 and the high precision is related to the low number of false positives.

$$Precision = \frac{TP}{TP+FP}$$

Recall is the ratio of correctly predicted positive observations in the universe of all positives, including those that had not yet been identified (FN).

$$Recall = \frac{TP}{TP+FN}$$

The F1-Score is the weighted average of precision and recall. This accuracy measurement is more used when there is an unbalanced class distribution in the data under study, that is, when the value of false positives is very different from the value of false negatives and is necessary when looking for a balance between precision and recall.

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

Usually, accuracy is observed when true positives and true negatives are more relevant for the classification problem, whereas the F1-score is used when false negatives and false positives matter more.

On the other hand, multi-class classification models consider a matrix of confusion whose dimension is k per k, where k represents the number of classes, that is, the number of response teams in our case. Unlike with binary classification, there are no positive and negative classes in this kind of model, so to calculate the standard evaluation metrics corresponding to each class, only whether the model predicts correctly should be considered.

In this project, the definitions of TP, TN, FP, and FN are not strictly applicable because, in the company, there are only two possible scenarios in relation to the forecasts made by the model: either the model hits the right team, or it doesn't hit and the team refuses the dispatch, starting the process again, wasting valuable time. This translates only into TP, true positives when the model suggests the correct team, and TN when suggesting the wrong team. Thus, the goal is to increase the number of TP and minimize the number of TN as much as possible.

One of the standard evaluation metrics that will be considered is accuracy [9]. The accuracy of the model represents the probability that the prediction performed by the model is correct across the entire data set. It is important to note that when we measure and observe accuracy in this type of classification problem, it is not possible to pinpoint classification classes where the model reports higher error rates. In fact, accuracy is generally computed by adding all cases that the model predicts correctly and dividing that value by all cases under study, rather than considering the classification results separately for each class.

Differently from the accuracy metric, the precision, the recall, and the F1-Score metrics are computed by considering each class separately. Thus, if we had more than two classes in our classification scenario, we would calculate, for example, the precision, of each class separately.

We will now illustrate the computation of these metrics in a multi-class classification problem with a small example, whose classes and data values are reported in Table 2.1.

Table 2.1: Example of a multi-class classification problem.

		Predicted				
		Core and Platforms	Mobile Connectivity	Outdoor Network	Fixed Connectivity	
Actual	Core and Platforms	**15**	2	1	1	19
	Mobile Connectivity	7	**15**	8	4	34
	Outdoor Network	2	3	**15**	2	22
	Fixed Connectivity	1	1	2	**15**	19
		25	21	26	22	94

The example depicts a multi-class classification problem where four classes are presented, representing the Opening Teams already described above: Core and Platforms, Mobile Connectivity, Outdoor Network, and Fixed Connectivity. If, on the one hand, the rows represent the actual outputs, that is, the expected outputs, on the other, the columns represent the outputs predicted by the model.

We will make a brief analysis only with the first two teams presented: Core and Platforms and Mobile Connectivity. Thus, the diagonal line represents the number of times the model correctly predicted the teams to which it was supposed. Further, number 2 represents the number of times that the Core and Platforms team was not identified by the model, and number 7, the number of times that the same team was incorrectly predicted by the model.

So, in this example, to calculate the accuracy we may use the following formula which considers all the classes together.

$$accuracy = \frac{15+15+15+15}{94} = \frac{60}{94} \approx 0,64$$

Then, to calculate a metric like precision, we must choose one specific class. For example, in the following computation, we chose the Core and Platforms class. Note that the calculation of precision of the selected class is made by dividing the number of cases that the model predicts correctly (15) by the total number of cases predicted by the model (25).

$$precision = \frac{15}{25} = 0,6$$

Similarly, to calculate the recall of the same class, we need to divide the number of cases that the model correctly predicts (15) by the total number of cases that actually exist (19).

$$recall = \frac{15}{19} \approx 0,79$$

Finally, the F1-Score is used to evaluate the performance of the model through the confusion matrix and can be interpreted as a weighted average between precision and recall. It can be computed based on previously obtained results.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{2 \cdot 0,6 \cdot 0,79}{0,6 + 0,79} = \frac{0,948}{1,39} \approx 0,68$$

The metrics above can be computed analogously for the other classes in this classification example.

After some careful consideration, NOS has decided to target the optimization of the training hyperparameters so that the minimum recall among the response teams would be as great as possible, that is, $MAX(MIN(Recall))$.

2.4 String Similarity Metrics

In order to find recurrent substring patterns in the data, it is important to introduce **string similarity metrics**.

There are many definitions for text distances. For the elaboration of Autodispatcher, it was necessary to study some of them to find recurring patterns in the names of the alarms to classify them correctly.

The first type of text distance studied was **Edit-Based Similarity** [28][21]. The algorithms that fit here aim to calculate the number of operations needed to transform a string into another, giving the same importance to all the characters of each string without, therefore, giving importance to the semantic meaning of the strings. The greater the number of operations, the greater the distance between the two strings presented and, consequently, the smaller the similarity between them. This type of approach is commonly used for small strings, making it efficient for longer texts or strings.

Within this type of text distance, several algorithms satisfy the conditions mentioned above, among them the **Hamming** and the **Levenshtein** Distances. The first-mentioned algorithm [21] calculates the distance overlapping the strings by observing the changes in the number between the two strings. In other words, the algorithm compares each character of a string with the corresponding character of another string, ranging from zero to the maximum length of the string. This type of implementation was performed to handle strings of the same length, and when such a feature is ignored, the algorithm takes into account the length of the longest string by adding one more value to the distance for each additional character in the longest string.

However, in the context of the distance from Hamming, there is another aspect to consider. As mentioned earlier, the Hamming distance can vary between zero and the length of the longest sequence shown. Therefore, it is possible to normalize this value by calculating the ratio between the distance and the length of the sequence. This result is called the normalized Hamming distance and represents the percentage in which the two sequences differ. The lower this value, the more similar the two sequences under analysis are.

Example: To calculate the Hamming Distance, the algorithm compares the corresponding characters in the two strings:

Position 1: "a" vs "a" - same
Position 2: "l" vs "l" - same
Position 3: "a" vs "b" - different
Position 4: "r" vs "u" - different
Position 5: "m" vs "m" - same

In this example, there are 2 positions where the characters of the two strings differ. Therefore, the Hamming distance between "alarm" and "album" is 2.

On the other hand, Levenshtein's algorithm [7][1] calculates the distance by determining the number of edits required to transform one string into another. These transformations can be of several types: insertion (adding a new character), exclusion (excluding a character present in the string), and substitution (replacing an existing character with another). When performing these three operations, the algorithm seeks to modify the first string to match the second. As with the Hamming distance, the greater the number of operations, the less similarity between the two strings.

In addition, it is possible to normalize the Levenshtein distance by obtaining the normalized Levenshtein distance. This value represents the index of similarity between the two strings, ranging from 0 to 1.

Example: For the two words *alam* and *alarm*, it is obvious that there is a missing character "r". Thus, to transform the word *alam* to *alarm* it is necessary to insert that character. The distance, in this case, is 1 because there is only one edit needed.

But, on the other hand, for the two words *ela* and *alarm*, we must first replace "e" with "a", then add a missing "r", followed by an "m" at the end. As a result, the distance is 3 because there are three operations applied.

Then, another type of text distance studied was **Token-Based Similarity** [28][21]. The algorithms that adopt this type of approach analyze the text as a set of tokens (bags of words), thus taking into account the semantic meaning of words and processing long texts. However, token-based similarity functions have the limitation of considering only the exact matching of two tokens, neglecting the fuzzy matching between them. In the case of Autodispatcher, it would be a major inconvenience since, if for some reason there were some kind of typo, they would make incompatible token pairs when they referred to the same token.

One of the algorithms that satisfy the conditions of Token-Based Similarity is the **Jaccard Index** [21]. The Jaccard Index measures the similarity between sets and is defined by the formula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| + |A \cap B|}$$

The numerator represents the number of tokens common to the two sets (intersection), and the denominator represents the total number of unique tokens. Again, the resulting value can range from 0 to 1.

Example: For this algorithm, consider the following sets composed of several strings:

A1 = ['alarm', 'urgency', 'team', 'realtime']
B1 = ['severity', 'impact', 'summary', 'realtime']

As can be seen, only 'realtime' belongs to both sets. As for the number of distinct tokens, there are four of the first set plus three of the second, for a total of 7. Therefore, through the formula presented earlier, it is obtained that the similarity of Jaccard is equal to $\frac{1}{7} = 0.142857$.

In addition, and recalling the case of typos, if the sets are considered:

A2 = ['Alarm', 'urgency', 'team', 'realtime']
B2 = ['severity', 'impact', 'summary', 'real-time']

It is observed that there is no equal token in both sets since realtime is, for this algorithm, different from real-time. Therefore, Jaccard's similarity is zero.

Finally, the last type of distance text studied was **Sequence-Based Similarity** [5]. In this type of approach, algorithms measure the similarity of text sequences, taking into account the order and structure of words, but, like Edit-Based, do not give importance to semantic meaning.

Within this type, there are two considerable algorithms: the **Longest Common Subsequence Similarity** (LCS) and the **Longest Common Substring Similarity** (LCSubstring) [5][33].

The LCS, as the name implies, is the longest subsequence common to all sequences in a set of sequences. The characteristic of this algorithm is that the sequences do not need to occupy consecutive positions within the original sequences.

On the other hand, LCSubstring represents the longest common substring between strings, and unlike LCS, it needs to occupy consecutive positions within the original strings.

Example: A simple example to portray the Longest Common Subsequence and the Longest Common Substring is to consider only the following strings: *test* and *text*.

If, on the one hand, the Longest Common Subsequence is *tet* (since consecutive characters are not required), on the other, the Longest Common Substring is just *te*.

Chapter 3

Related Work

In [36], Zhang et al. introduce an approach to sentiment analysis on Twitter using two variants of convolutional neural networks (CNN): one based on words and one based on characters.

Word-based CNN operates by considering the semantics of words as units of analysis. On the other hand, character-based CNN takes a more granular approach, including all characters in its analysis and requiring no knowledge of the semantic or syntactic structure of the language it classifies. This character-based CNN process begins with the transformation of characters into numbers using the corresponding UTF-8 decimal codes. This technique provides a significant advantage in targeting sentences at the character level, even when using different languages, such as Chinese and English, in a single text. In addition, the size of the character alphabet is dynamic, differing from traditional approaches that rely on a predefined alphabet.

The results obtained in the analysis of the eight data sets are shown, and it is possible to observe that, in two languages, the word-based CNN presented a slightly higher performance compared to the character-based CNN. However, this trend was reversed in the other six languages analyzed, where character-based CNN demonstrated a superior ability to classify feelings.

This finding underscores the importance of the flexibility of the character-based approach, which dynamically adapts to the particularities of different languages. The hybrid approach combining word- and character-based CNNs offers a new perspective for analyzing feelings in a multilingual context, as in the case study of the social network Twitter.

Additionally, and in consideration of the research introduced previously, in [2], present an approach to automating problem dispatch using machine learning. Specifically, the study compares the performance of a CNN at the character level and at the word level in order to understand which of the two is most effective as input to the model.

When analyzing the results, it is apparent that the character-level CNN slightly surpassed the word-level CNN in terms of accuracy. This discovery raises a fundamental question, already addressed in the previous study presented [36]: Why do characters prove to be a more appropriate entry for a CNN than words in this context? One of the suggested reasons for this difference in performance thus lies in CNN's ability at the character level to capture different forms of words, which rely on distinct grammatical rules in different languages.

In summary, this study, now under analysis, demonstrates that, given the specific characteristics of the data set and the complexity of the languages involved, the character-level approach is more effective in the task of automatic dispatch of questions using machine learning. This highlights the importance of selecting the correct representation of input data in natural language processing tasks, taking into account linguistic particularities.

On the other hand, Zhong et al. in [38] addresses the issue of the complaint channels used to report quality issues. These complaints often require quick classification and resolution. The study proposes an approach based on CNN, where they incorporate a deep learning method to automatically classify short texts contained in complaints.

The classification process described is carried out in two distinct steps. The first consists of data pre-processing and, as the name implies, consists of pre-processing the raw texts, performing word segmentation, and eliminating stop words. This allows the text of the complaint to be expressed in the form of valid sentences and prepared for subsequent analysis. The second step consists of the CNN-based deep learning model. The study acquires the semantic characteristics of texts in the form of text vectors. This is accomplished by incorporating words, resulting in each text being represented as a dense matrix of real values. The extraction of resources from texts related to construction quality problems is performed through convolution kernels. These semantic characteristics are then mapped to individual labels using a multilayer perceptron classifier (MLP). Finally, the sorting task is completed with the generation of labeled text.

The study recognizes the complexity of determining the number of hidden layers and the size of neurons involved in neuronal networks. Zhong et al. [38] suggest taking several factors into account, including the input and output layers, the activation function, and the neuronal network architecture. However, they admit that there is no standard method to determine the number of hidden layers and the size of neurons. For the case under review, research indicated that a single hidden layer is sufficient to achieve reasonable sorting results for short texts.

In summary, the research highlights a significant conclusion: CNNs prove to be highly effective in capturing semantic information contained in short texts, exhibiting remarkable adaptability to this task.

Similar to the previous experiment mentioned earlier [38], in [37], Zhang et al. present an experimental analysis of CNNs for sentence classification. It aims to provide useful guidance for researchers and practitioners who want to implement and use CNNs in real-world phrase classification scenarios.

The study investigates a wide range of architecture options and hyperparameters of CNNs with the purpose of understanding the behavior of these networks. One of the main approaches employed by CNNs is the conversion of the tokens that make up each phrase into vectors, resulting in an array that serves as input to the network.

One relevant finding is that choosing the input word vector has a significant impact on CNN

performance. For the classification of sentences, it was observed that the direct use of one-hot vectors is not the most effective approach.

In addition, the article recommends considering different activation functions, with emphasis on Relu and tanh, which stand out as the best options for obtaining satisfactory results.

Ultimately, this study provides valuable insights for those seeking to apply CNNs to sentence classification tasks, highlighting the importance of careful experimentation and adaptation of settings to the specific needs and characteristics of each scenario.

Chapter 4

Autodispatcher Design

This chapter provides a comprehensive overview of the Autodispatcher system by detailing its design. It starts with a description of the data sets used in Autodispatcher development, the development of Autodispatcher in Section 4.1. This description lays the groundwork for understanding the importance of the company's data in the subsequent analysis of the various system components.

The Autodispatcher system was designed to establish a hierarchical decision-making process for the classification of the alarms raised by the NOS company's infrastructure. Autodispatcher employs several stages within a decision tree that embeds different types of classification rules, including human-defined and automatically inferred rules, string frequency analysis, and machine learning-based classification. The different stages of the system are detailed in Section 4.2. The integration of different classification rules enabled by the use of Autodispatcher allows for the continuous classification of incoming new alarms in the company's infrastructure.

Finally, Subsection 4.3 illustrates the importance of the Autodispatcher system's components and their interactions within the organizational structure of the company. The illustration of the interactions is important to better understand the impact of this system on the work environment and the organizational processes at NOS Portugal.

In summary, this chapter provides an in-depth view of the Autodispatcher system, including its design, implementation, and integration into the company's operational workflow. Through this detailed analysis, the reader can gain a deep understanding of the system's components as well as of their interactions, and, as a consequence, potentially assess the benefits and challenges with the adoption of this system into the operational workflow of the company.

4.1 Data sets

To start with the design of Autodispatcher, it was necessary to analyze two existing data sets, used by the company to store information about alarms issued by its infrastructure. The first data set, $helix.alarm\_events$, stores a detailed description of the alarms in a single table. Among other fields, this data set includes the severity of the alarm, the $troubleticket\_ttid$ field that corresponds to the alarm's unique identification code, the name of the equipment that triggered the alarm, the type of alarm, the area and the district where the event occurred, the device type, the priority of

the alarm, the device vendor. The data set table is filled automatically with a new record each time an alarm is generated. Hence, updates to this table can be considered to happen in near real-time.

The second data set, $npd.inc\_generalinfo$, contains 96 fields in a single table. This data set contains additional fields corresponding to how the alarm was handled. It includes for example, fields like the $openingteam$, the $ticket\_key$ (basically, an alarm id), the company name (management, within NOS, responsible for the equipment), the summary, impact, urgency level, and priority of the alarm. Differently from $helix.alarm\_events$, this data set is filled manually by a human operator by observing the first data set.

More in detail, whenever an alarm is triggered, some company's software detects it and adds a corresponding entry to the $helix.alarm\_events$ data set. Shortly afterward, a human operator analyzes the event and inserts a new record of it into the $npd.inc\_generalinfo$ data set. When the related event is eventually resolved (for instance, the response team successfully repaired a malfunctioning component), a new human-generated entry is added to $npd.inc\_generalinfo$.

Table 4.1: Fields characterizing alarms, selected from two original data sets at NOS Portugal.

(a) Alarm fields selected from $helix.alarm\_events$ data set.

Field	Description	Type
troubleticket_ttid	alarm identification code (id)	VARCHAR
severity	the importance/intensity of the alarm: critical, major, minor or warning	VARCHAR
type	name of the incoming notification corresponding to the alarm type	VARCHAR
alarmtype	brief description of the alarm type	VARCHAR
rawalarmtype	type of raw alarm that generated the alarm	VARCHAR
baseprobablealarmcause	possible alarm cause	VARCHAR
area	parish and county where the alarm occurred	VARCHAR
district	county and district where the alarm occurred	VARCHAR
devicetype	type of device that generated the alarm	VARCHAR
vendor	vendor of the device that generated the alarm	VARCHAR
priority	alarm priority	NUMERIC

(b) Alarm fields selected from $npd.inc\_generalinfo$ data set.

Field	Description	Type
ticket_key	alarm id	VARCHAR
openingteam	name of the team that solved the problem	VARCHAR
company	name of the company the alarm belongs to	VARCHAR
summary	description of the alarm occurrence	VARCHAR
impact	alarm impact generated	NUMERIC
urgency	alarm urgency level	NUMERIC
priority	alarm priority	NUMERIC
systemorigin	system origin of each alarm	VARCHAR

Since the data sets of the company have a considerable size, with more than 10000 records and 200 alarm fields in total, it was decided, as an initial exploratory approach, to identify and work with a small subset of their data, aiming to identify a minimal set of fields to encode the nature of each alarm in the data sets. A smaller data set allowed us to easier experimentation with several of the functional blocks of Autodispatcher presented in the next sections of this chapter. Towards such a goal, we have performed a join of the two tables, $helix.alarm\_events$ and $npd.inc\_generalinfo$, selecting only the records where $troubleticket\_ttid = ticket\_key$. With this joint operation, we have considered only nineteen fields from the original tables. These fields, shown in Table 4.1, were selected because they contain more descriptive information about the

alarms and because they have an easier format to analyze, i.e., they have very few fields with blank values in the existing records. It is worth noting that this join was possible since all tickets had already been validated previously by a set of human operators. Specifically, when the alarms are received by the company's infrastructure and recorded in the $helix.alarm\_events$ table, a human operator analyzes the alarm and, if the operator knows what it is, creates a ticket in the $npd.inc\_generalinfo$. In this way, all tickets are validated, ensuring that all alarms that are found in this second table, are previously classified. We acknowledge that validation errors can still happen due to human errors, but we believe that the number of errors is going to be relatively small considering the expert team applied to validate them. Besides, we believe that the small percentage of error would not be relevant to affect the overall classification process.

Within the newly created data set, we have included alarm fields that are relevant to the classification process we aimed to implement. Namely, the data set includes the field $summary$, which corresponds to a more detailed description of the alarm occurrence, and the field $openingteam$ which contains the name of the response team to be dispatched for the resolution of the technical problem that generated the event. Again, the ultimate goal of the first version of the Autodispatcher will be to predict the $openingteam$ to most likely solve a problem based on any $summary$ given as input.

By manually inspecting the summary field, the new data set allowed us to immediately analyze in more detail some alarms and quickly draw some important conclusions. Namely, it was possible to observe that all the summaries that started with "NOKIA", "ERICSSON", "HUAWEI" and "BRA1" always belonged to the opening team called "Mobile Connectivity" and, in the same way, that the summaries that started with "SMOP" always belonged to the "Core and Platforms" team. These preliminary conclusions were very useful since they allowed us to further reduce the number of summaries to be analyzed later.

To further simplify the preliminary design of Autodispatcher, we have also created another smaller data set, consisting of only four fields: the $openingteam$ and the $summary$ (fields taken from the data set created previously) and two new fields called $openingteam\_num$ and $openingteam\_vec$. The first field created, $openingteam\_num$, consists of digits from zero to three, depending on the $openingteam$. If the $openingteam$ is "Core and Platforms", the field holds the value zero; if the $openingteam$ is "Mobile Connectivity", the field holds the value one; if the $openingteam$ is "Outdoor Network", the fields hold the value two; and, if the $openingteam$ is "Fixed Connectivity" the field holds the value three. The other new field, $openingteam\_vec$, contains a vector, whose values indicate the selected to $openingteam\_num$. If the value is zero, then the corresponding vector will be [1,0,0,0]. Similarly, if $openingteam\_num$ = 1 then the vector will be [0,1,0,0], if $openingteam\_num$ = 2 then $openingteam\_vec$ = [0,0,1,0] and, finally, if $openingteam\_num$ = 3, $openingteam\_vec$ = [0,0,0,1].

Finally, based on the analysis of the two smaller data sets, we have decided to remove the alarm summaries that started with "NOKIA", "ERICSSON", "HUAWEI", "BRA1" and "SMOP" to simplify the analysis of the remaining summaries. This reduced the total count from 231 different

alarm summaries to 195. This new set of summaries, which we called d_red, contained the following fields: $openingteam$, $openingteam\_num$, $summary$ and $openingteam\_vec$. Finally, we transformed each $summary$ of the data set d_red into ASCII code and added this information into a new column, called sum_encod, to the data set. Initially, this data set created will be used to train and test the neural network later, which constitutes the fourth and final component of the Autodispatcher decision tree.

4.2 System Design

The Autodispatcher tool builds upon a hierarchical set of decision blocks that automatically classify incoming alarms. The hierarchical classification process is illustrated at a high level in Figure 4.1. It must be reported that this configuration, that is, the order and kind of decision blocks, has been agreed upon by the company, and it is meant to always return one single team at the end.

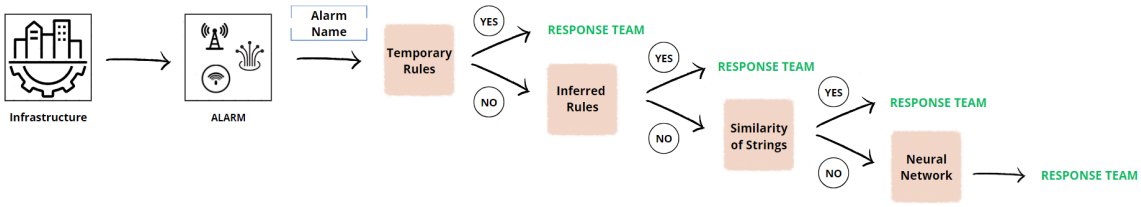


Figure 4.1: The hierarchical classification process of Autodispatcher at a high level.

The classification process implemented by Autodispatcher implements a decision tree. This tree is composed of different types of rules, including rules defined by the team of experts at the company, so-called temporary rules, inferred rules, string frequency analysis, and, finally, the application of a model based on a neural network. This combination allows, when an alarm is received by the company's system, it to be classified instantly, running through this decision tree.

Figure 4.2 and Figure 4.3 illustrate two examples of distinct alarms and their progression through the decision tree to determine the corresponding response team. In the first example, shown in Figure 4.2, we observe that the alarm *SETUBAL_CHARAFE_LTE_MST208B1* does not fall into the initial two blocks of the tree, namely the temporary rules or the inferred rules. However, it shares a common sequence of strings, leading to the termination of the alarm's traversal through the tree. As a result, the output identifies the response team most likely to address the alarm-related issue.

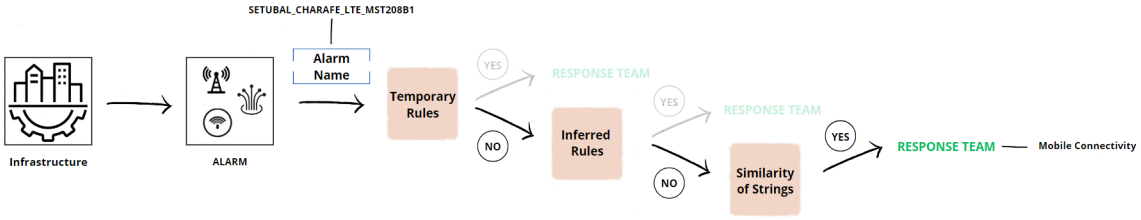


Figure 4.2: The path of an alarm through the first three blocks of the decision tree.

In contrast, Figure 4.3 illustrates an alarm example where the name of the alarm traverses through the first three blocks of the decision tree, finally halting at the neural network stage. Remarkably, the name of this alarm did not fall under the temporary or inferred rules, nor did it match any string similarity. Consequently, the alarm exhaustively journeyed through the entire tree to determine the response team most likely to address the problem, ultimately concluding at the last block, which indicated the involvement of the external network team.

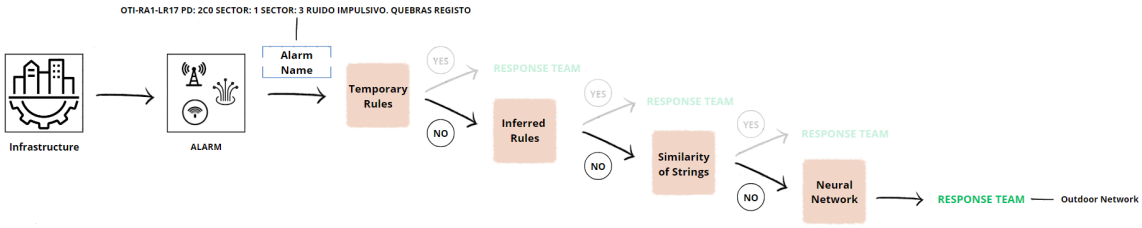


Figure 4.3: The path of an alarm through the four blocks of the decision tree.

As the whole alarm classification process is lightweight and runs fast, it is possible to use Autodispatcher in real-time, so that the human operator responsible for the decision to choose the most likely response team to solve the problem can leverage this tool to classify an incoming alarm. Autodispatcher provides the capability to create a backlog of alarms that are deemed less critical. This backlog serves the purpose of analyzing the potential impact on response teams if they have to handle all the recorded alarm incidents. By automating the handling of non-urgent alarms, the burden on human operators is reduced. This, in turn, allows them to concentrate their attention on truly important matters.

4.2.1 Temporary rules for alarm analysis

The first step consists of a check-up list for specific equipment that requires rigorous attention upon the request of any team. At a given time, certain equipment might be in a developmental or rollout phase, thus not falling under the regular oversight of the designated team. Although this step cannot be automated, the team responsible for addressing work orders can easily modify the current team's responsibility through a simple configuration file edit. Subsequently, in this block of the decision tree, the function evaluates whether the provided ci_name (or alarm name) corresponds to any of the established rules in the database. Both the rules and the database are developed by human operators in the NOS's internal infrastructure. Upon finding a match, this decision block assigns the appropriate response team.

4.2.2 Inferred rules for alarm analysis

Another set of human-defined rules is checked when an alarm name is not present in the temporary rule sets, which records well-known equipment and related alarms. These inferred rules are built manually by company experts to encode potential match patterns between alarm descriptions and response teams which have occurred over time. The list consists of a set of strings and their associated response team if the alarm contains any of those strings. For example, strings such as

"NOKIA", "ERICSSON" and "HUAWEI" are present in these rules because, after several tests, it was observed that all alarms containing these strings in their respective alarm descriptions are related to the Mobile Connectivity response team.

This second classification block works as follows. If an alarm name (ci_name) contains some string present in the inferred rules set, this block will return the response team associated with that string.

Based on test trials in the company, this approach of identifying possible correlations between certain strings in alarm names and previous classification results, turned to further refine the classification process previously formed by only the fixed human rules.

4.2.3 String frequency analysis

One of the main challenges in the design of Autodispatcher is represented by having company assets (e.g. equipment) and response team names constantly changing. Therefore, to cope with this dynamicity, we have devised and included in Autodispatcher a specific decision block. This block is based on a string distance-based approach that tries to reassure that previously known associations between known alarms and response teams will be preserved if only minor changes to their names occur. This step will become the third stage of the decision tree, but since the project is meant to be modular, it could easily be set as any other stage. An explanation follows.

One of the most common changes in alarm information over time is a change in the asset name. More often than not, this change is minor. The change in nomenclature can occur due to several reasons. The introduction of some novel assets to the infrastructure of the company (cell phone tower network, TV, internet fiber optics, etc.) may require further changes to existing names so that all existing and new components fall under. Therefore, we worked on a string frequency analysis method so that minor name changes can still point to the same team as before.

For this part of the work, a new data set composed of several columns from $helix.alarm\_events$ is used, among them: *equipmentname*, *equipmentname2*, and *additionaltext*. The field $equipmentname$ contains the alarm name and the other columns contain additional information about the alarm.

So, in the previous chapter 2, we discussed different types of distance measures used to determine the similarity between texts. One particular type we explored was **Sequence-Based Similarity**, which focuses on analyzing the order and structure of words within the texts. And, within Sequence-Based Similarity, two notable algorithms stand out: *Longest Common Subsequence Similarity (LCS)* and *Longest Common Substring Similarity (LCSubstring)*.

Now, in the current chapter, we delve into the practical application of these algorithms. Specifically, in step 3 of our hierarchical function, we aim to find the most similar string. For this purpose, the most appropriate algorithm to employ is the *LCSubstring* algorithm. Unlike *LCS*, *LCSubstring* considers consecutive positions within the original strings, which is crucial in our context.

This choice becomes particularly relevant when dealing with alarm names since using the *LCS* algorithm could result in an inadequate match since it doesn't require consecutive positions. The absence of a single letter could lead to a completely different alarm.

By leveraging the capabilities of the *LCSubstring* algorithm in step 3, we can confidently find the most similar string, ensuring the accuracy and effectiveness of our system in handling alarms.

Therefore, our objective is to calculate substrings of the *equipmentname* field that are repeated frequently. We have applied some criteria to the selection of the substrings, to retain only those whose information was relevant to the choice of a potential response team. Hence, we implemented a filtering process to ensure accuracy and relevance. Substrings that appeared fewer than five times were excluded, as the alarm information within the company typically consists of marks with a minimum length of five characters (e.g., "NOKIA"). Additionally, we disregarded substrings with a length of less than four, as these shorter substrings lack precision in conveying meaningful information. And, lastly, we eliminated any substrings that contained some kind of Stop Words, further refining the data set.

The Stop Words list is a list created for, whenever an alarm may contain substrings present in it, discarding substrings whose information is irrelevant to the team's choice. It consists of the names of Districts (Distrito in Portuguese and in the database), Municipalities (Concelho, idem), and Parishes (Freguesia, idem). Most alarms include geographical information like the district where they occur, but this does not provide information on the nature of the faulty asset in the company. This happens because the naming of each item is the responsibility of the team that reports it at first, but naming conventions change over time with the introduction of new technologies (e.g., 5G cells).

After obtaining the list of all possible substrings of this column, the strings in the summary field are tested to see if they contain any substring (a field present in $npd.inc\_generalinfo$) and, if found, how they relate to the teams. In this way, and for better analysis, a data set was created composed of the substrings already described by the corresponding teams and their probabilities. These probabilities were calculated based on the number of times the string appeared throughout the data set as well as the number of times each team appeared for that string.

If an alarm is not classified in the previous two decision blocks of Autodispatcher, this third stage of the hierarchical function will work as follows: if the ci_name does not meet the previous two requirements, i.e., if it does not belong to any set of rules previously described, it will be moved to this third stage. In this third step, the smallest distance between the ci_name, given as input, and the substrings previously created is calculated; that is, it will be observed which is the largest common sequence between the input and a string present in the list in question. After that, the function will return the strings with more similarity to the input, as well as the probabilities of the respective teams. This similarity was restricted because only the results will appear where the distance between the ci_name and the string is less than three. This number was chosen because, after some tests, it was concluded that it is the minimum number of characters that would be able to return reliable information so that comparisons could be made correctly and, subsequently, possible assignments to the teams could be made.

As already mentioned, this approach aims to identify the most similar strings even when the algorithm was not trained with the new name changes. To illustrate the practical application of

this approach, two examples are presented below in the form of tables 4.2 and 4.3. These examples demonstrate how the algorithm calculates substrings and determines the similarities between strings. By examining these examples, it is possible to gain a better understanding of how the algorithm operates and how it aids in the decision-making process to accurately assign alarms to teams. For example, there is the alarm *SETUBAL_CHARAFE_LTE_MST208B1* and the *PORTO_CURRAIS_LTE_MPT351B2*, both present in the table 4.2. As can be seen, the largest subsequence common between them is in fact *_LTE_M*, which, according to rules already practiced and decided by the company led to the Mobile Connectivity team.

Table 4.2: Three alarm instances that show the application of the string method

STRING	SETUBAL_CHARAFE_LTE_MST208B1	PORTO_CURRAIS_LTE_MPT351B2	CACEM_NORTH_WEST_GSM_110S2
SETUBAL_CHARAFE_LTE_MST208B1	SETUBAL_CHARAFE_LTE_MST208B1	_LTE_M	-
PORTO_CURRAIS_LTE_MPT351B2	_LTE_M	PORTO_CURRAIS_LTE_MPT351B2	-
CACEM_NORTH_WEST_GSM_110S2	-	-	CACEM_NORTH_WEST_GSM_110S2

Table 4.3: Three more alarm instances that show the application of the string method

STRING	CACEM_NORTH_WEST_GSM_110S2	AGUIAR_DA_BEIRA_LTE_NGR079B2	LISBOA_CGD_JOAO_XXI _GSM_UMTS_LTE_335U6
CACEM_NORTH_WEST_GSM_110S2	CACEM_NORTH_WEST_GSM_110S2	-	_GSM_
AGUIAR_DA_BEIRA_LTE_NGR079B2	-	AGUIAR_DA_BEIRA_LTE_NGR079B2	_LTE_
LISBOA_CGD_JOAO_XXI _GSM_UMTS_LTE_335U6	_GSM_	_LTE_	LISBOA_CGD_JOAO_XXI _GSM_UMTS_LTE_335U6

4.2.4 Neural Networks for alarm analysis

The final decision block of the Autodispatcher classification process has been implemented through a neural network. More in detail, the neural network model of Autodispatcher was built through the TensorFlow (TF) framework. TF was chosen for two main reasons. First, it is an open-source platform with free access to it. It is also well documented, enabling easy installations and customization (optimizing existing examples and/or adding new features) of such a framework. Second, it is the most used platform for Neural Networks within the company. NOS hosted many training workshops on TensorFlow in the recent past, so its employees are well-versed in this technology and, as a consequence, more willing to use and improve tools developed through it.

To start with, we opted for a neural network with four layers, an input layer, an output layer, and two hidden layers. The input layer corresponds to the twenty-character-long description of the asset generating the alarm. The output layer is an N element layer, where N is the number of distinct response teams in our data set. The choice of such an output layer is meant to compute confidence levels for each distinct team, instead of generating a single team as an output. Each node of the output layer reports a score indicating how likely the current alarm belongs to the respective response team, as illustrated in Figure 4.4. The number of hidden layers for this first design was determined by a few experimental trials which are reported in more detail in Appendix

A, in Figure A.1. We opted for a neural network with two hidden layers of 64 elements each, because classification results obtained using a neural network with a single hidden layer were below our target expectations for this model.

Team	Score
0	0.02
1	0.00
2	0.98
3	0.00
⋮	⋮
N	0.00

Figure 4.4: Example of a neural network output consisting of the N distinct teams and the score determined by the model

After the initial design, to improve the efficiency of the model, we performed an exploration of the hyperparameter space. First, in the summary field, we increase the number of characters given as input. With an increase of 20 characters to 25, we observed a slight improvement in the results.

Second, we also varied the number of training epochs. In the first design only ten training epochs are used. In the first design, we used only ten training epochs. Afterward, we increased the number of training epochs to sixty for the next model we experimented with. This choice was driven by the size and complexity of our data set, and by some initial tests we performed. Those tests have been reported in Appendix A, see Figure A.2. The combination that presented the best results was the one consisting of twenty-five characters and two hidden layers with 64 nodes each.

To create a model, the data was initially divided into two sets: the **training set** and the **test set**. This division was achieved using the $train\_test\_split$ function provided by the TensorFlow (TF) framework. The training set, which accounted for 80% of the data, consisted of 304 alarms. The purpose of this set was to train the model.

Each input summary's twenty-five characters were converted into their respective ASCII codes during the training process. This resulted in a vector that served as the input to the neural network's input layer. For alarm names shorter than 25 characters, white spaces were added and converted to their ASCII equivalents as well.

Embeddings

The Autodispatcher project is programmed to be modular and easily configurable by another user inside the company, to customize it to their own needs. However, the default configuration consists of the four blocks of decision in a strict order, to be used in an automated manner, i.e., to generate tickets automatically in a pilot program. As it stands, the first two blocks represent rules established by the company, detailed in Subsections 4.2.1 and 4.2.2. The third block is formed by the string similarity technique, as described in Subsection 4.2.3. The last block consists of the neural network trained in TensorFlow.

To understand the reason behind the transformation of text data into ASCII code, it is essential to analyze in detail the implementation of the third block, which is based on the "bag-of-words" model. This model represents text as an unordered collection of words, ignoring their meaning and order. It is often used in document classification methods, where the frequency of each word is used as a resource, as demonstrated in the construction of the third block of the decision tree, as explained in Subsection 4.2.3.

However, replicating the same process in the fourth and final stage of the decision tree would result in identical results in both blocks, making the last block redundant in Autodispatcher. Therefore, to ensure the independence of the neural network training about the previous steps, we chose to use an alternative method, that is, the transformation of text data into ASCII codes. This approach proved to be advantageous as it is widely applicable and has already demonstrated success in related projects within the company. Thus, this choice is supported by both the company and the project itself.

It is also important to note that, when transforming the input data into ASCII code, it is possible to normalize them. This provides a great advantage since alarm names have varying lengths. To resolve this issue, it is possible to add white spaces, ensuring that all alarms are the same length. During the implementation of these steps, another advantage was observed: pre-processing is faster when comparing text data with data already converted into ASCII code. This speed is especially beneficial when dealing with real-time data, which will be the case with Autodispatcher when implementing it.

Returning to the topic, after the model was trained using the training set, the performance and fitness of the model were evaluated using the test set. The test set consisted of 77 alarms, which accounted for the remaining 20% of the data. This evaluation process helped determine how well the trained model performed on unseen data and provided insights into its effectiveness.

To design a model, particularly neural networks, TensorFlow requires specifying three input parameters: optimizer, loss, and success metric [8]. These three parameters together contribute to the training and evaluation process of the model, enabling the optimization of its performance and assessment of its effectiveness.

The optimizer parameter is meant to update the model based on the data it analyses. It adjusts the model's internal parameters to minimize the loss function and improve its performance. Although several types of optimizers are available on TensorFlow, the "Adam" optimizer was chosen because it is known to automatically adapt the learning rate based on the magnitude of the gradients, allowing for faster convergence. Besides, Adam adapts well to large data sets, being able to deal with large-scale training scenarios without requiring excessive memory, and is less sensitive in the choice of model hyperparameters compared to other optimizers, it is the case with SGD (Stochastic Gradient Descent) [15][29].

The loss parameter is used to measure the accuracy or discrepancy of the model's predictions during the training phase. It quantifies how well the model minimizes the difference between

predicted and actual values. In the case of Autodispatcher, we have chosen the Sparse Categorical Cross Entropy, as previously explained in Section 2.

The success metric parameter is employed to monitor and evaluate the performance of the model in both the training and testing stages. It provides a quantitative measure of how well the model achieves the expected objectives. In this case, we have chosen accuracy as a success metric, to determine the percentage of summaries that were classified correctly.

During training, the model was evaluated with the training set data alone. The training of the model with the training set data outputs reports the accuracy and loss function of the model. At the end of training, we obtained an accuracy of the model of approximately 62% and a loss rate of 3.05.

After training, the model was tested with the test set, which included data points the model had never seen. Two test variables were created to help in the analysis of the model's performance: $test\_list$ (list of summaries) and $test\_targets$ ($openingteam\_num$). In this phase, the model's predictions were compared with the data in $test\_targets$ and we observed that the model made correct predictions in 69% of the cases.

Following the neural network's results, a considerable step was undertaken to enhance their accuracy: the introduction of a validation set between the training set and the test set. This addition aimed to obtain more precise and reliable outcomes. The validation results showed great promise, indicating the effectiveness of this approach.

It is worth highlighting that to uncover the ideal configuration, a comprehensive exploration of different layer combinations was conducted, systematically testing and refining the model until the most favorable setup was discovered. This iterative process allowed for fine-tuning the model and optimizing its performance to attain the best possible results.

Jointly with the implementation of a validation test, we implemented the k-fold technique to avoid the potential overfitting of the data. This technique was particularly important since the classes in our data set were not optimally balanced. This technique involved dividing the data into k equal subsets, known as folds. In each iteration, the model was trained on k-1 folds and tested on the remaining fold. By repeating this process for all the folds, we ensured a comprehensive evaluation of the model's performance.

To address the imbalanced classes, we utilized the oversampling technique, whereby entries from the sub-represented classes were duplicated. This approach aimed to achieve equal frequencies for all classes across all sets of the model. By adopting this strategy, along with choosing 5 folds, we were able to achieve remarkable results that had not been previously observed during the model's construction.

The neural network constitutes the last decision block of Autodispatcher. An alarm name ci_name will reach this stage if the tool is not able to classify the alarm through any of the previous decision blocks. If an alarm reaches this decision block, the neural network calculates the most likely team to solve the problem, based on the given input, and presents results as probabilities for each response team, as shown in the example below (Table 4.4).

Table 4.4: Example of an output where the possible response teams for an analyzed alarm are indicated by the neural network.

Response Teams	Probability
Core and Platforms	0.02
Mobile Connectivity	0.00
Outdoor Network	0.98
Fixed Connectivity	0.00

This example is only intended to illustrate the potential output of this decision block. The neural network was fed with a fake alarm name '___' only to visualize the layout described so far. As the table shows, the response team indicated that the most likely to deal with this alarm indicated by the neural network was the Outdoor Network one, with a 98% certainty.

4.3 System Components and Interactions

Within NOS Portugal, the Autodispatcher toll will play a considerable role in managing alarms and assigning teams to solve asset-related problems. When an alarm is triggered, the Autodispatcher will immediately take action and create a ticket in a public database, visible to all teams, where entries will indicate the most appropriate team to deal with the situation.

Autodispatcher's primary goal is to optimize the process of responding to incidents by streamlining the efficient allocation of teams and ensuring that each alarm is routed quickly to the right team. It is important to note that Autodispatcher will not replace human intervention but it will act as a complementary tool, hopefully improving the allocation of available resources.

In the event of an incorrect team assignment by Autodispatcher, the system is designed to correct the error. The alarm is returned for review, and the team responsible for the information table related to the alarms makes the necessary corrections. These fixes are then used to train and improve the Autodispatcher model, gradually reducing the likelihood of similar errors occurring in the future. This feedback loop ensures continuous improvement and it enhances the Autodispatcher's accuracy and reliability over time.

Autodispatcher has been implemented to deploy it throughout the company, considering the existence of several distinct teams that will want to audit the developed code. Individual teams will also use Autodispatcher and create their custom versions. This flexibility will allow each team to tailor and adjust Autodispatcher to their specific needs, taking full advantage of the benefits of the automated system to accelerate task assignment within their area of expertise.

Furthermore, the company practices a policy of open code and the code is available to every team. Autodispatcher may be used not only to tackle the allocation of teams to alarms, but it could also be adapted for teams looking to solve challenges such as customer complaints, commercial contracts, or any other multiclass classification challenge where there is a constant shift in the responsible team or person required to handle an event with haste.

Autodispatcher could be changed to play an essential role as an important component of the

system, interacting dynamically with teams (or persons) and the alarm information table (or data sources with a similar structure). By streamlining the dispatch process, Autodispatcher enables response teams to act quickly and efficiently, ensuring the effective resolution of identified issues effectively. In addition, human feedback and ongoing model training fuel a cycle of constant improvement, improving Autodispatcher accuracy and reliability over time.

In the ever-evolving industrial environment, these Autodispatcher interactions with teams and the alarm information table result in an adaptable and reliable system, able to meet emerging challenges and assist in maintaining the efficiency and operational safety of the company.

Chapter 5

Results and Evaluation

In this chapter, the results obtained through the implementation of Autodispatcher will be presented. A thorough evaluation of the performance and effectiveness of the model will be shown, analyzing its impact and its ability to optimize the team allocation workflow. The test environment and the metrics used to assess system performance will also be described.

Subsequently, the results obtained during the test period will be presented, including the ability of Autodispatcher to be changed and customized by any interested party at the company.

It is important to note that the Neural Network results presented are obtained using a different data set than the ones described in previous chapters. However, the data set now used shares the same characteristics (same schema) and underwent the same pre-processing steps as the data set used in Chapter 4.1. The new data set now consists of approximately 1000000 alarms and was utilized to enhance the performance of the models, and their generalization capabilities, and mitigate potential overfitting or underfitting issues.

The performance metrics for the string similarity model are more straightforward than the array of neural network training metrics. Since a direct relationship between features and targets does not exist in the training data set, we will employ the data set where that relation is established by the experts of the alarm allocation team for evaluation. The analysis will consist of simple *hit or miss* tests.

Finally, we will conclude the chapter with a general evaluation of Autodispatcher, summarizing its contributions and discussing its practical implications.

It is important to highlight that throughout the entire development process of the Autodispatcher, exclusively the Python programming language was used. Within this programming language, specific libraries were utilized, namely TensorFlow, NumPy, Pandas, and Scikit-Learn. The outcomes of the diverse models presented in Subsection 5.2 were executed in parallel, using the same data sets, with the objective of understanding which model showed the most favorable results.

5.1 String distance evaluation

The evaluation for the string distance method of Autodispatcher is not as extensive as the existing well-developed evaluation methods for Neural Networks models.

5.1.1 Evaluation

As mentioned earlier, the purpose of string distance is to find the most similar string in a data set based on an alarm name provided as input. This data set consists of the Strings field, the corresponding team, and the percentage to which the team is related to the alarm. The evaluation addresses the accuracy of the function in identifying the most appropriate sequence and, consequently, the corresponding team. The evaluation was conducted using a new data set consisting of 11270 alarm names, and for each alarm name in the data set, the function was applied to find the most similar string, within the data set of the strings found and described in Subsection 2.4, observe the team that the function returned, and, in the end, verify that this same team was equal to the team coming in the new data set.

The alarm name similarity function showed an accuracy of 81%. This indicates that in 81% of cases, the function was able to correctly identify the team corresponding to the alarm in question, even for alarm names that were not present in the original data set. This can be attributed to the distance-based string approach, which allows you to identify similar patterns in alarm names regardless of specific variations.

It is important to note that, given the nature of alarm names, which may change over time, the accuracy rate achieved is considered positive.

5.2 Neural Network performance

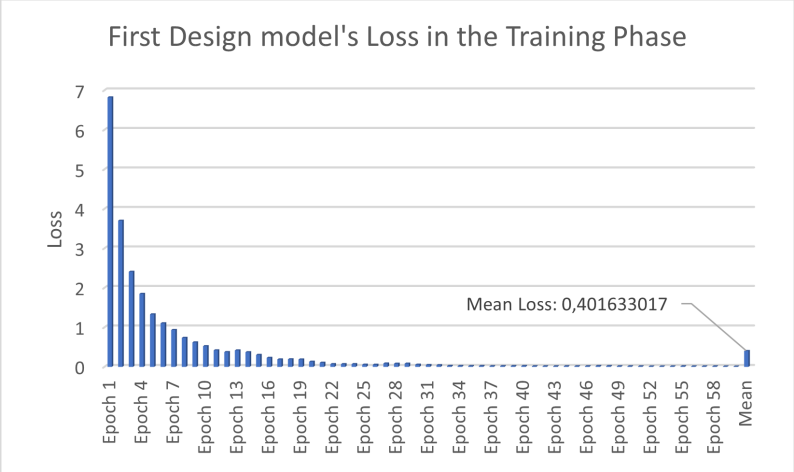
5.2.1 First Design

The first model design featured a concealed layer comprising 64 nodes. The model underwent training during 60 iterations, with the "Adam" optimizer and the Sparse Categorical Cross Entropy loss function used. As mentioned in Section 4.2.4, the training data set was constituted of 304 input alarms, while the testing data set encompassed 77 alarms. The goal of the model was primarily an exploratory exercise to get familiarized with the TensorFlow framework. However, this approach did not yield favorable outcomes. The following model was redesigned using two hidden layers as a new baseline, The following model was redesigned using two hidden layers as a new baseline, with the same characteristics that the first one described at the beginning of this Section 5.2.1.

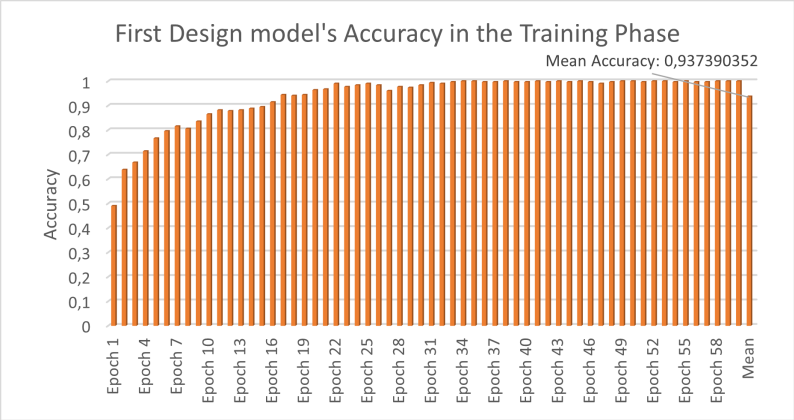
During the training phase, the first design model, with two hidden layers, demonstrated promising performance, with an average loss of 0.201 and an average accuracy of 0.937 approximately, Figure 5.1. These metrics reflect the model's ability to learn from the provided data and make accurate predictions.

It is important to recognize that the mean loss represents the mean error obtained by the model throughout the training phase, signifying the mean discrepancy between the network's output and

the anticipated outcome. Likewise, the mean accuracy denotes the mean accuracy across the 60 epochs, as observed throughout the model's training phase. These metrics serve to illustrate that the model assimilated knowledge from the training data and exhibited considerable performance.



(a) Results of the loss in the training phase of the first model



(b) Results of the accuracy in the training phase of the first model

Figure 5.1: Results of the training phase of the first model

In the subsequent test phase, the first design model achieved an average accuracy of 0.941 when comparing its predicted values with the actual values ($test\_targets$). This indicates that the model accurately predicted the desired outcomes in 94% of the cases. The model exhibited a 30% error rate in predicting team 0 (Core and Platforms), an 8% error rate in predicting team 1 (Mobile Connectivity), a 1% error rate in predicting team 2 (Outdoor Network), and a 26% error rate in predicting team 3 (Fixed Connectivity), Figure 5.2.

Overall, the results of the first design model are quite good, with high accuracy during both the training and testing phases. However, the significant variation in error rates for different teams suggests that the model has experienced some difficulty accurately predicting outcomes for some classes. This could be attributed to an imbalanced class distribution in the training data or inherent difficulty in distinguishing some classes.

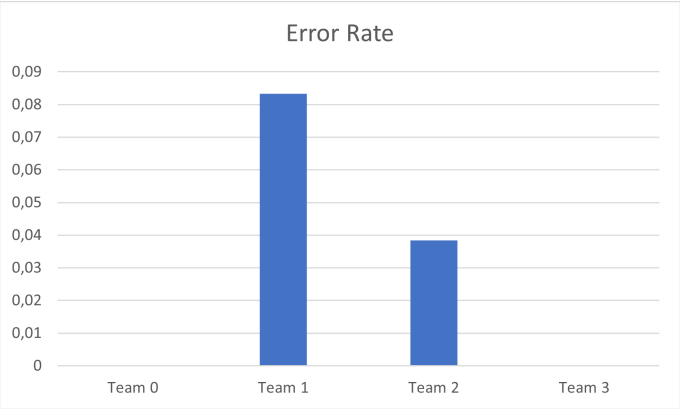


Figure 5.2: Error rate of the first model

5.2.2 Validation Set

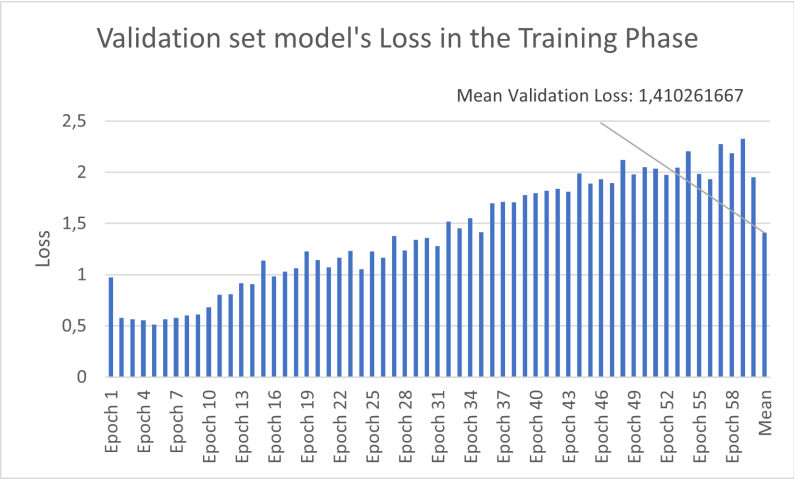
The second model was constructed with a similar architecture to the first model, employing two hidden layers, each comprising 64 elements. The data set for this model was divided into three subsets: a training set, a validation set, and a test set, with a proportional size of the entire data set of 70%, 15%, and 15%, respectively.

During the training phase, the second model achieved a mean validation loss of 2.833 and a mean validation accuracy of 0.857, Figure 5.3. It is important to acknowledge that the mean validation loss denotes the mean error obtained by the model during the training phase, that is, the mean of the difference between the network output and the expected output. Similarly, the mean validation accuracy signifies the mean accuracy over the 60 epochs, also observed during the training phase of the model. These metrics indicate that the model learned from the training data and performed reasonably well on the validation set, capturing the underlying patterns in the data.

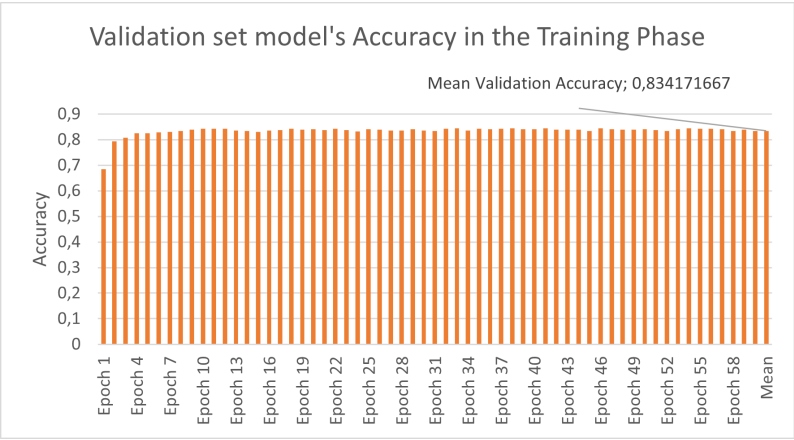
Unlike the training phase, the subsequent test phase evaluates the performance of the model on data that have not yet been seen, measuring its average accuracy, which in this case was 0.857, when comparing its predicted values with the actual values ($test\_targets$). This indicates that the model achieved an 83% success rate in accurately predicting the desired outcomes. Specifically, it had a 22% error rate in predicting team 0 (Core and Platforms), a 7% error rate in predicting team 1 (Mobile Connectivity), a 14% error rate in predicting team 2 (Outdoor Network), and a 14% error rate in predicting team 3 (Fixed connectivity), Figure 5.4.

The overall performance of the second model is reasonably good, with high accuracy during both the training and testing phases. This indicates that the model can generalize its learned patterns effectively and perform reliably on unseen data during the testing phase. The consistent accuracy suggests that the model is not overfitting to the training data and is capable of making accurate predictions on new, unseen data. These results indicate a robust and reliable model, i.e., one that performs consistently across different phases, which is a positive outcome.

However, the variation in error rates for different teams suggests that the model may have



(a) Results of the loss in the training phase of the validation set model



(b) Results of the accuracy in the training phase of the validation set model

Figure 5.3: Results of the training phase of the validation set model

faced challenges in accurately predicting outcomes for some classes. This could be attributed to the inherent complexity of distinguishing between specific classes.

5.2.3 K-Fold Cross Validation

Since the results so far have not been entirely satisfactory, a decision has been made to develop a more robust model, striving to achieve the best possible accuracy. To accomplish this objective, another architecture was created utilizing K-fold cross-validation with five folds. This approach allowed us to assess the model's performance across multiple subsets of the data set.

It's worth highlighting that the sole modification, in comparison to the previous approach detailed in Section 5.2.2, pertains exclusively to the validation method. Notably, both the data set and the test set have remained unchanged.

During the training phase, the K-Fold Cross Validation model demonstrated a mean accuracy of 0.933, indicating its ability to learn and generalize patterns effectively. Additionally, the model achieved a mean loss of 0.193, indicating its capacity to minimize errors and optimize predictions,

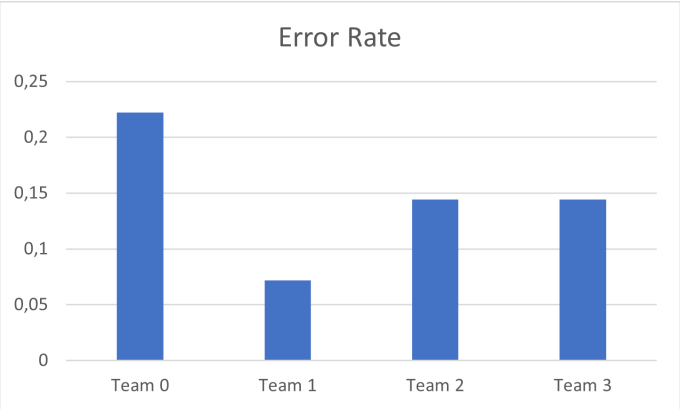


Figure 5.4: Error rate of the validation set model

Figure 5.5.

Moving on to the test phase, the K-Fold Cross Validation model continued to showcase its competency, attaining a mean accuracy of 0.938. This result indicates that the model successfully generalized its learning to new, unseen data with a high degree of accuracy.

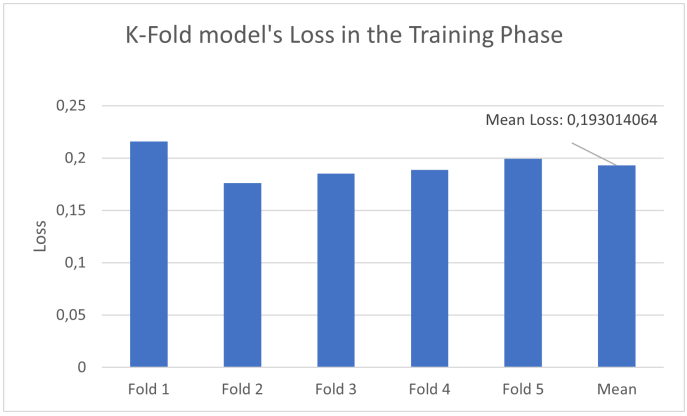
Furthermore, it is essential to consider the error rates for each team within the data set. Team 0 (Core and Platforms) exhibited a relatively low error rate of 0.07, suggesting that the model accurately predicted outcomes for this specific team. Similarly, team 1 (Mobile Connectivity) and team 2 (Outdoor Network) experienced minimal error rates of 0.024 and 0.03, respectively, indicating the model's proficiency in predicting their respective attributes. However, it is worth noting that team 3 (Fixed connectivity) had a higher error rate of 0.11, indicating the model struggled to accurately predict outcomes related to this team, Figure 5.6.

Overall, the results of the K-Fold Cross Validation model are promising, with high accuracy achieved during both the training and testing phases. The model demonstrates effective learning and generalization capabilities, as well as the ability to minimize errors.

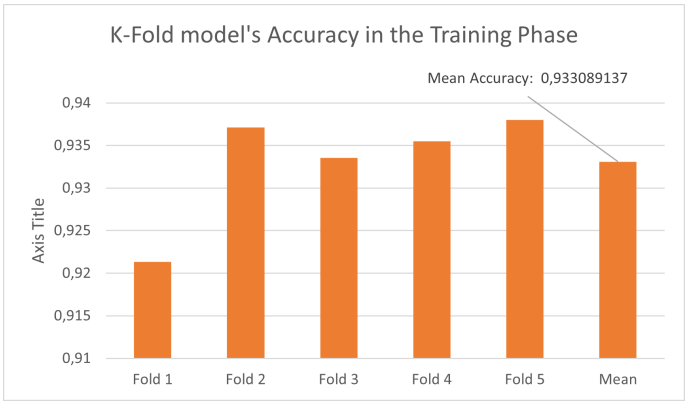
5.3 Full Autodispatcher output analysis

After the breakdown of the two main modules of Autodispatcher, the string similarity approach, and the Neural Network approach, we now turn our attention to the entire assembled product.

As was described in Figure 4.1, the default configuration of Autodispatcher consists of four steps, as per the company's requirement. Again, the first step is a check-up list for specific equipment that is under very close scrutiny at the request of any one team. At a given time, a particular equipment might be in a development or rollout phase, and not being overseen by the team that usually oversees this kind of equipment. For example, a cell phone tower is being retrofitted with a battery, to have backup power. The company may wish to use the team responsible for power supply as a default team for the time the retrofit is taking place, instead of the original Mobile Connectivity team. There is no way of automating this step, but a team responsible for addressing work orders can easily change what team is responsible for the time being, as it is a simple con-



(a) Results of the loss in the training phase of the K-Fold model



(b) Results of the accuracy in the training phase of the K-Fold model

Figure 5.5: Results of the training phase of the K-Fold model

figuration file edit. An alarm for example is *ALMEIRIM_GSM_79US3*, as can be seen in Figure 5.7. The visual representation indicates that when the name in question is provided as input, the hierarchical function instantly halts at stage 1. At this stage, the function effectively identifies the name, which is present in the check-up list designated by the company for specific equipment.

The second step of Autodispatcher is a variation of the first. Instead of a unique name for the equipment under special investigation, it is a family of names falling under a certain rule. As an example, equipment started with "NOKIA" will automatically be addressed to the "Conetividade Móvel" (Mobile Connectivity) team, skipping subsequent steps, as depicted in Figure 5.8 below.

The third step of Autodispatcher is triggered when the alarm name, given as input, fails to meet the previous requirements or rules. In this stage, the function computes the smallest distance between the input and pre-generated substrings. The function then returns the most similar strings, along with the probabilities associated with their respective teams, as evident from the three distinct examples illustrated in Appendix B.1.

The fourth and final phase of the Autodispatcher involves the utilization of the neural network, specifically designed to handle alarms that do not meet the requirements established in the preceding three stages. Within this phase, the model, which has undergone pre-training and validation,

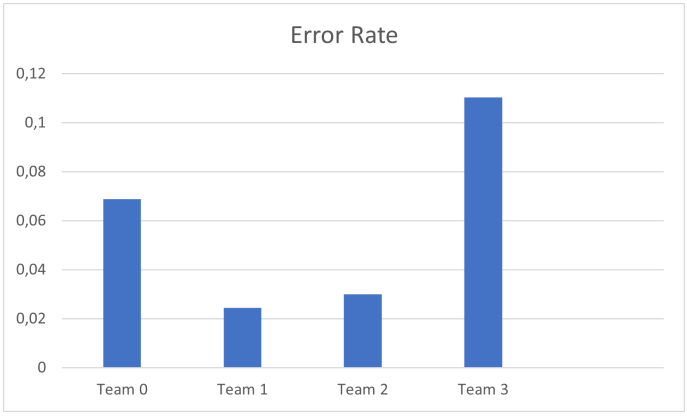


Figure 5.6: Error rate of the K-Fold model

```
hierarchical_function('ALMEIRIM_GSM_79US3', df_fullname, df_rule)
```

	String	equipa
0	ALMEIRIM_GSM_79US3	Conetividade Móvel

Figure 5.7: Illustrative instance of an alarm name that triggers the immediate termination of the hierarchical function at stage 1

generates suggestions for the teams along with their respective probabilities. Figure B.2 provides a visual representation, showing cases where the input alarm does not meet the criteria established in the previous three stages, and therefore the neural network suggests a team. In these three examples in Appendix B.2, Figure B.2, it can be observed that the model can give its prediction with a high rate of certainty, hitting the correct team.

On the contrary, in Appendix B.2, Figure B.3, highlights scenarios where the neural network correctly identifies the team but the certainty rates displayed by the network are relatively modest.

In addition, it can also be observed cases in which the alarm has been renamed. As can be seen in Figure B.4a, the neural network misses its prediction and suggests the wrong team. After a new system-wide training session, the alarm name still does not stop at stage 3, which in this case is because the alarm name contains some Stop Words and the only identifiable pattern is *_MST*, which not appear more than five times, the string method is encoded to continue to ignore. Therefore, it follows for the neural network that after re-training successfully generates the correct prediction, Appendix B.2, Figure B.4b.

During the writing of this thesis, another possible scenario was raised to display as many examples as possible. The hypothesis was questioned to test the credibility of the model in cases where the Autodispatcher errs in the classification of the alarm in the third block of the hierarchical function, but if it were tested exclusively in the fourth stage of Autodispatcher, this stage, constituted by the model built based on neural networks, was already correct in the classification of the same.

```
hierarchical_function('NOKIA', df_fullname, df_rule)
```

	Substring	equipa
0	NOKIA_	Conetividade Móvel

Figure 5.8: Exemplary case of an alarm name that prompts the hierarchical function to terminate at stage 2

It is worth noting that the described scenario is highly improbable. The third block of the Autodispatcher is constructed using alarm names that have previously occurred and analyzed by our team of experts. These alarm names are already associated with the appropriate response teams, as confirmed by said experts. As a result, any substrings extracted from generated strings and presented as output in block 3 will consistently align with previously generated strings and their respective assigned teams.

However, if there is any case where this happens, that is, if the third block of the decision tree returns the wrong team, the team of experts who are responsible for validating the suggestion of Autodispatcher will observe what happened, verify the incorrect assignment, and, finally, incorporate the alarm name in question into the data set within the first block. This measure ensures that such errors are rectified and avoided in the future.

It should be noted that, until the completion of the completion of this document, no cases were found where the third block returned an incorrect team assignment.

5.4 Summary

In this study, we developed and analyzed three different models using TensorFlow for a specific prediction task. Each model had a different architectural design and utilized various training and testing approaches. The goal was to identify the best model based on its performance and characteristics.

For the Neural Network models, it is clear that the K-Fold Cross Validation model outperformed the other two models in terms of accuracy and generalization. This model showed the highest mean accuracy during testing, indicating its ability to predict outcomes with a high degree of accuracy on unseen data. Although it also had varying error rates for different teams, it showcased a relatively low error rate for most of the prediction classes. Therefore, the K-Fold Cross Validation model is the recommended choice for the prediction task for the Autodispatcher.

Chapter 6

Conclusions and Future Work

The goal of this work is to improve efficiency and the response time to serious alarms generated by the telecommunications infrastructure of NOS through the implementation of Autodispatcher. At the writing of this work, it is in testing.

The Autodispatcher system aims to improve the efficiency and response time of NOS by introducing a new component that assists in classifying alarms and recommending the most suitable response team to solve technical failures. The current manual approach to analyzing alarms has limitations, including delays in alarm analysis and the potential for misclassification.

To address these challenges, the Autodispatcher system utilizes supervised learning techniques, including a neural network model and a string distance approach. These techniques are trained on the company's alarm reporting tool data sets and are capable of predicting the most appropriate team based on the alarm name and nature. By providing recommendations for response teams, the Autodispatcher system can assist in reducing response time and improving decision-making.

The development of the Autodispatcher system involves the integration of various types of rules, such as temporary rules, inferred rules, string frequency analysis, and the neural network model. The first type of rule consists of a checklist for specific equipment that is under observation for some reason. On the other hand, the inferred rules are rules created by a team of experts based on their knowledge, containing only alarms known to the company. The third stage of Autodispatcher consists of string frequency analysis, whose objective is to calculate the shortest distance between each alarm that reaches the company's infrastructure and a set of pre-generated substrings previously associated with a response team. The last block of this tool is thus the neural network, and this was made to identify correlations between the different types of alarms and the most appropriate response teams to deal with each problem that reaches the infrastructure, making their recommendations.

These rules form a hierarchical decision-making structure within the system, allowing for continuous and instant classification of alarms as they traverse the decision tree. The system's components and their interactions within the organizational structure are crucial for understanding the impact of the Autodispatcher tool on the work environment and organizational processes.

After a comprehensive analysis of the three distinct approaches to the Autodispatcher system,

as outlined in section 5.2, the K-Fold Cross-Validation model consistently demonstrated a high level of performance, both in terms of accuracy and generalizability during internal evaluations. With its accuracy and strong generalization capabilities, the K-Fold cross-validation model proves its capacity to accurately predict outcomes when dealing with unseen data. While there may be some variability in error rates across different forecast categories, the overall performance of the K-Fold cross-validation model shows great promise. Consequently, we highly recommend it as the optimal choice for forecasting within the Autodispatcher system.

Overall, the Autodispatcher system provides a comprehensive solution to enhance the classification and response process for alarms in NOS. By leveraging machine learning techniques and integrating multiple rule types, it improves the company's ability to quickly and accurately identify response teams for technical failures, ultimately reducing response time and enhancing efficiency.

In the final stretch of the writing of this thesis, the idea of developing a mobile application that connects to Autodispatcher emerged after its implementation in the company. This application would aim to optimize the mobility and efficiency of employees and technicians by allowing them to receive alarm notifications and update the state of incidents in real time through their mobile devices. This would result in a faster and more effective resolution of incidents.

With this implementation, technicians would be promptly informed about incidents, receiving notifications whenever an alarm was assigned to them. This would ensure even faster response times, allowing technicians to start working on alarm resolution immediately, taking into account their location and/or proximity to workstations.

The mobile application would also offer users the possibility to update the status of incidents in real time, providing information about progress, and obstacles encountered, and even indicating if the designated team is the correct one. If, when a technician arrives on site, he realizes that it is the wrong team, he could communicate this important information to the team of experts and other colleagues involved in the process of sending the team and/or solving the problem. This feature would promote collaboration, by ensuring that everyone is on the same page at the same time.

In short, implementing the described mobile application has the potential to further reduce response times in solving the problem generated by the alarm, helping it to be treated effectively anytime and anywhere. This would result in a more agile and responsive incident management process.

Bibliography

[1] Karin Beijering, Charlotte Gooskens, and Wilbert Heeringa. Predicting intelligibility and perceived linguistic distance by means of the levenshtein algorithm. *Linguistics in the Netherlands*, 25(1):13–24, 2008.

[2] Fredrik Bengtsson and Adam Combler. Automatic dispatching of issues using machine learning, 2019.

[3] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[4] Chien-Hua Chen, Po-Hsiang Lin, Jer-Guang Hsieh, Shu-Ling Cheng, and Jyh-Horng Jeng. Robust multi-class classification using linearly scored categorical cross-entropy. In *2020 3rd IEEE International Conference on Knowledge Innovation and Invention (ICKII)*, pages 200–203. IEEE, 2020.

[5] Yi-Ching Chen and Kun-Mao Chao. On the generalized constrained longest common subsequence problems. *Journal of Combinatorial Optimization*, 21(3):383–392, 2011.

[6] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[7] Alexandru Ene and Andrei Ene. An application of levenshtein algorithm in vocabulary learning. In *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–4. IEEE, 2017.

[8] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[10] Isabelle Guyon et al. A scaling law for the validation-set training-set size ratio. *AT&T Bell Laboratories*, 1(11), 1997.

[11] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[13] Euge Inzaugarat. Understanding neural networks: What, how and why? *Medium library*, 2018.

[14] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Will Koehrsen. Overfitting vs. underfitting: A complete example. *Towards Data Science*, pages 1–12, 2018.

[17] Venkat Krishnan. Iris data tensorflow neural network. `https://www.kaggle.com/code/venkatkrishnan/iris-data-tensorflow-neural-network`, 2023.

[18] Haochen Li and Zhiqiang Zhan. Machine learning methodology for enhancing automated process in it incident management. In *2012 IEEE 11th International Symposium on Network Computing and Applications*, pages 191–194. IEEE, 2012.

[19] Bruce G Marcot and Anca M Hanea. What is an optimal value of k in k-fold cross-validation in discrete bayesian network analysis? *Computational Statistics*, 36(3):2009–2031, 2021.

[20] Joydwip Mohajon. Confusion matrix for your multi-class machine learning model. *Towards data science*, 2020.

[21] Felix Naumann. Similarity measures. *Hasso Plattner Institut*, 2013.

[22] Phil Picton. *What is a Neural Network?*, pages 1–12. Macmillan Education UK, London, 1994.

[23] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[24] Jun Qi, Jun Du, Sabato Marco Siniscalchi, Xiaoli Ma, and Chin-Hui Lee. On mean absolute error for deep neural network based vector-to-vector regression. *IEEE Signal Processing Letters*, 27:1485–1489, 2020.

[25] Tianshuo Qiu, Xin Shi, Jiafu Wang, Yongfeng Li, Shaobo Qu, Qiang Cheng, Tiejun Cui, and Sai Sui. Deep learning: a rapid and efficient route to automatic metasurface design. *Advanced Science*, 6(12):1900128, 2019.

[26] Ladislav Rampasek and Anna Goldenberg. Tensorflow: biology's gateway to deep learning? *Cell systems*, 2(1):12–14, 2016.

[27] Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(10), 2020.

[28] Rui Santos, Patricia Murrieta-Flores, and Bruno Martins. Learning to combine multiple string similarity metrics for effective toponym matching. *International journal of digital earth*, 11(9):913–938, 2018.

[29] Aaswad Sawant, Mayur Bhandari, Ravikumar Yadav, Rohan Yele, and Mrs Sneha Bendale. Brain cancer detection from mri: A machine learning approach (tensorflow). *Brain*, 5(04):2089–2094, 2018.

[30] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

[31] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[32] Manu Siddhartha and Avik Santra. Covidlite: A depth-wise separable deep neural network with white balance and clahe for detection of covid-19. *arXiv preprint arXiv:2006.13873*, 2020.

[33] Vianney Kengne Tchendji, Hermann Bogning Tepiele, Mathias Akong Onabid, Jean Frédéric Myoupo, and Jerry Lacmou Zeutouo. A coarse-grained multicomputer parallel algorithm for the sequential substring constrained longest common subsequence problem. *Parallel Computing*, 111:102927, 2022.

[34] Ž Vujović et al. Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications*, 12(6):599–606, 2021.

[35] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[36] Shiwei Zhang, Xiuzhen Zhang, and Jeffrey Chan. A word-character convolutional neural network for language-agnostic twitter sentiment analysis. In *Proceedings of the 22nd Australasian Document Computing Symposium*, pages 1–4, 2017.

[37] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[38] Botao Zhong, Xuejiao Xing, Peter Love, Xu Wang, and Hanbin Luo. Convolutional neural network: Deep learning-based classification of building quality problems. *Advanced Engineering Informatics*, 40:46–57, 2019.

[39] Yangfan Zhou, Xin Wang, Mingchuan Zhang, Junlong Zhu, Ruijuan Zheng, and Qingtao Wu. Mpce: a maximum probability based cross entropy loss function for neural network classification. *IEEE Access*, 7:146331–146341, 2019.

Appendix A

Tests for Autodispatcher design

```
for ind in range(4):
    tt = test_targets==ind
    print(ind, 1-np.count_nonzero(test_targets[tt]==pred[tt])/len(test_targets[tt]))
```

```
0 0.6666666666666667
1 0.6666666666666667
2 0.019230769230769273
3 0.9230769230769231
```

(a) Results of the model with one hidden layer and 64 elements

```
for ind in range(4):
    tt = test_targets==ind
    print(ind, 1-np.count_nonzero(test_targets[tt]==pred[tt])/len(test_targets[tt]))
```

```
0 0.6666666666666667
1 0.5555555555555556
2 0.0
3 0.15384615384615385
```

(b) Results of the model with two hidden layers and 64 elements each

```
for ind in range(4):
    tt = test_targets==ind
    print(ind, 1-np.count_nonzero(test_targets[tt]==pred[tt])/len(test_targets[tt]))
```

```
0 0.6666666666666667
1 0.7777777777777778
2 0.019230769230769273
3 0.7692307692307692
```

(c) Results of the model with one hidden layer and with 128 elements

Figure A.1: First tests for the first Autodispatcher design, varying the number of hidden layers and their elements

It is important to note that, in this first Figure A.1, all the possible designs consisted of 10 epochs and 20 characters as input.

```
for ind in range(4):
    tt = test_targets==ind
    print(ind, 1-np.count_nonzero(test_targets[tt]==pred[tt])/len(test_targets[tt]))
```

```
0 0.19999999999999996
1 0.1428571428571429
2 0.05555555555555558
3 0.2727272727272727
```

(a) Results of the model with one hidden layer, with 64 elements and 60 epochs

```
for ind in range(4):
    tt = test_targets==ind
    print(ind, 1-np.count_nonzero(test_targets[tt]==pred[tt])/len(test_targets[tt]))
```

```
0 0.0
1 0.19999999999999996
2 0.0
3 0.07692307692307687
```

(b) Results of the model with two hidden layers, with 64 elements each, 60 epochs and 20 characters

```
for ind in range(4):
    tt = test_targets==ind
    print(ind, 1-np.count_nonzero(test_targets[tt]==pred[tt])/len(test_targets[tt]))
```

```
0 0.0
1 0.0
2 0.0
3 0.2666666666666667
```

(c) Results of the model with two hidden layers, with 64 elements each, 60 epochs and 25 characters

Figure A.2: Tests for the first Autodispatcher design, varying the number of hidden layers and their elements

Appendix B

Outputs of exemplary alarms

B.1 Instances of alarms that stopped during the third stage of the Autodispatcher

```
hierarchical_function('OLA_GSM_34', df_fullname, df_rule)
```

	String	Equipa	Percentagem
509	_GSM_3	Conetividade Móvel	1.0
511	E_GSM_3	Conetividade Móvel	1.0

(a) Instances where the alarm encompasses a widely used acronym in the company, such as *GSM*

```
hierarchical_function('CMP1C', df_fullname, df_rule)
```

	String	Equipa	Percentagem
1280	CMP1CMTS00	Core e Plataformas	1.0
1282	CMP1CMTS007	Core e Plataformas	1.0

(b) Segments of particular alarm names that are already recognized, such as *CMP1C*

```
hierarchical_function('SETUBAL_CHARAFE_LTE_MST208B1', df_fullname, df_rule)
```

	String	Equipa	Percentagem
1368	_LTE_M	Conetividade Móvel	1.0

(c) An exemplification of an alarm that incorporates the stop word *SETUBAL*

Figure B.1: Illustrative scenarios of alarm names that cause the hierarchical function to halt at stage 3.

B.2 Occurrences of alarms that halt during the fourth stage of the Autodispatcher

```
hierarchical_function('OTI-RA1-  LR17 PD: 2C0 Sector: 1 Sector: 3  RUIDO IMPULSIVO. QUEBRAS REGISTO', df_fullname, df_rule)

No similar string found in a dataset, the model will suggest a team.
Core e Plataformas: 0.00
Conetividade Móvel: 0.00
Rede Exterior: 1.00
Conetividade Fixa: 0.00
```

(a) An exemplification is presented wherein the model confidently provides a suggestion with utmost certainty.

```
hierarchical_function('OTI-RA2 VFR25-3C7 COM DS SNR BAIXO NA FREQ 642.00MHZ', df_fullname, df_rule)

No similar string found in a dataset, the model will suggest a team.
Core e Plataformas: 0.00
Conetividade Móvel: 0.00
Rede Exterior: 1.00
Conetividade Fixa: 0.00
```

(b) An illustration is presented wherein the model exhibits unwavering confidence as it delivers a suggestion with utmost certainty.

```
hierarchical_function('COMMUNICATION LINK IS DOWN # NLSPL1ERIP1 # A n Rtr_Cisco device, named NLSPL1ERIP1, has detected a', df_fullname, df_rule)

No similar string found in a dataset, the model will suggest a team.
Core e Plataformas: 0.00
Conetividade Móvel: 0.00
Rede Exterior: 0.02
Conetividade Fixa: 0.98
```

(c) An example where the model demonstrates a notably high certainty rate when dealing with the alarm in question.

Figure B.2: Examples where the model exhibited a high certainty rate in correctly identifying the right team.

```
hierarchical_function('Canal (192) TCV INTERNACIONAL com macroblocos', df_fullname, df_rule)

No similar string found in a dataset, the model will suggest a team.
Core e Plataformas: 0.22
Conetividade Móvel: 0.00
Rede Exterior: 0.10
Conetividade Fixa: 0.68
```

(a) An illustrative example where the neural network successfully identifies the correct team, despite its certainty rate not reaching a considerable level.

```
hierarchical_function('ARM1BQAM002 Com alarme de  Backup device is in use', df_fullname, df_rule)

No similar string found in a dataset, the model will suggest a team.
Core e Plataformas: 0.22
Conetividade Móvel: 0.04
Rede Exterior: 0.12
Conetividade Fixa: 0.62
```

(b) An instance where the neural network adeptly identifies the appropriate team, even though its certainty rate falls short of a substantial level.

```
hierarchical_function('Incidente para a acompanhar os trabalhos da colt NLSPL1ERIP1', df_fullname, df_rule)

No similar string found in a dataset, the model will suggest a team.
Core e Plataformas: 0.00
Conetividade Móvel: 0.50
Rede Exterior: 0.37
Conetividade Fixa: 0.13
```

(c) Illustrative scenario where the neural network presents distributed probabilities, yet achieves positive results by correctly identifying the appropriate team.

Figure B.3: Scenarios where the neural network identifies the correct team, even though its certainty rate may not reach a significant level.

```
inference_function('FEIJO_ANTONIO_ELVAS_MST040B1')
# ATENÇÃO - ESTE MUDOU DE NOME NO FIM DE FEVEREIRO
# POR ISSO É QUE O MODELO ERRA
```

```
No similar string found in a dataset, the model will suggest a team
Core e Plataformas: 0.00
Conetividade Móvel: 0.02
Rede Exterior: 0.98
Conetividade Fixa: 0.00
```

(a) An instance in which the alarm name underwent a modification, consequently the method of strings did not catch and as the neural network was also not aware, the model exhibited a fallacy in its prediction.

```
hierarchical_function('FEIJO_ANTONIO_ELVAS_MST040B1', df_fullname, df_rule)
```

```
No similar string found in a dataset, the model will suggest a team.
Core e Plataformas: 0.00
Conetividade Móvel: 0.97
Rede Exterior: 0.00
Conetividade Fixa: 0.03
```

(b) An example where, after a subsequent training session, the neural network was able to correctly assign the alarm to the appropriate team.

Figure B.4: A scenario where there was a change in the alarm name. After re-training the system, the string method continued to fail, but the model was able to recognize the associated team.