Anticipation of Perturbances in Telco Services

Tânia Margarida Marques Carvalho
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência dos Computadores
2019

Orientador
Rita Paula Almeida Ribeiro
Professora Auxiliar
Faculdade de Ciências da Universidade do Porto

Coorientador
Nuno Miguel Pereira Moniz
Professor Auxiliar Convidado
Faculdade de Ciências da Universidade do Porto

Orientador Externo
Carlos Miguel Silva Couto Pereira
NOS Comunicações

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________

Abstract

Many telecommunication industries run preventive maintenance models to assure lower operation costs, higher network stability and, more importantly, costumer's satisfaction. Many events, like equipment malfunction or service installation, require interventions by technicians in customers' houses. These interventions can have a substantial impact both on the client himself and his neighbours. Sometimes, solving one problem leads to another. The purpose of this thesis is to develop predictive maintenance models to allow planning of preventive actions. We propose to achieve this objective by detecting technical interventions that may affect the service of other clients in the same building.

A predictive model of technical interventions was developed, using network signals. The goal is to identify contaminant interventions, i.e., those that degrade the signals of a neighbouring customer. This project faces a highly imbalanced setting, as data shows few cases with contaminant interventions. Thus, specific techniques are applied to circumvent this issue. This study employs machine learning techniques, using several classification algorithms such as tree based, distance-based and artificial neural networks. We also resort to strategies for tackling imbalance domain learning problems. Given that the available data is collected at different time intervals, the experimental evaluation is carried out with growing and sliding window methodologies, using appropriate metrics for imbalanced classification.

Results show that each approach employed in this thesis concerning the prediction of contaminant technical interventions obtained results that are considerable better in comparison with base models. Thus, we conclude that the models proposed in this thesis are capable of improving the identification of contaminant interventions in an extremely imbalanced domain.

Resumo

Muitas indústrias de telecomunicações executam modelos de manutenção preventiva para garantir redução de custos de operação, maior estabilidade de rede e, mais importante, satisfação do cliente. Muitos acontecimentos, como falhas de equipamento ou instalação de serviços, exigem intervenções por parte de um técnico em casa dos clientes. Estas intervenções podem ter um impacto substancial no próprio cliente e nos seus vizinhos. Algumas vezes, resolver um problema pode desencadear outro. O objetivo desta tese é desenvolver modelos de manutenção preventiva para possibilitar o planeamento de ações preventivas. Para alcançar este objetivo propomos detetar as intervenções técnicas que podem afetar o serviço de outros clientes no mesmo edifício.

Para resolver este problema, foi desenvolvido um modelo preditivo de intervenções técnicas utilizando sinais de rede. O objetivo é identificar intervenções contaminantes, ou seja, aquelas que degradam os sinais de um cliente vizinho. Este projeto enfrenta um cenário altamente desbalanceado, pois os dados mostram poucos casos com intervenções contaminantes. Portanto, foram aplicadas técnicas específicas para contornar esse obstáculo. Este estudo aplica técnicas de *machine learning*, usando vários algoritmos de classificação tais como baseados em árvores, baseados em distâncias e redes neuronais artificiais. Também recorremos a estratégias para lidar com o problema de domínios desbalanceados. Dado que a informação disponível é coletada em diferentes intervalos de tempo, a avaliação experimental é realizada com as metodologias de *growing* e *sliding window*, usando métricas apropriadas para classificação desbalanceada.

Os resultados mostram que cada abordagem aplicada nesta tese acerca da previção de intervenções técnicas contaminantes obtiveram resultados que são consideravelmente melhores em comparação com os modelos base. Assim, concluímos que os modelos propostos nesta tese são capazes de melhorar a identificação de intervenções contaminantes em *datasets* extremamente desbalanceados.

Acknowledgements

First, I would like to thank my supervisors Professor Rita Ribeiro and Professor Nuno Moniz for all the support throughout this process, for all suggestions that improved my thesis and for this opportunity.

I also want to thank my partner company co-advisor Carlos Pereira for all dedication and guidance. Thanks for innovative ideas and all knowledge transmitted.

To Luís Costa and Pedro Oliveira for providing important advices and moments of happiness during some difficult hours.

I would also like to express my gratitude to the analytical team for the weekly meetings that helped improve my thesis and also for sharing other work done at the company.

A special thank you to my family. My parents Augusto and Cristina, because without them would never be where I am and would not be what I am today. My brother Pedro and my sister Beatriz.

Finally, to João. For all the support and ideas that helped me develop this project. For providing all the affection and confidence that I needed. And most importantly thanks for all the patience.

Dedicated to
My parents and João.

Contents

List of Tables

List of Figures

Acronyms

KDD	Knowledge Discovery in Databases
ROC	Receiver Operating Characteristics
AUC	Area Under the ROC Curve
RUS	Random Under-Sampling
ROS	Random Over-Sampling
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Support Vector Machine
RBF	Radial Base Function
KNN	*k*-Nearest Neighbours
ANN	Artificial Neural Network
GLM	General Linear Model
TO	Technical Order
TP	Technical Participation
DP	Distribution Point
CMTS	Cable Modem Termination System
CM	Cable Modem
HFC	Hybrid Fiber Coax
SNR	Signal-to-Noise Ratio

Chapter 1

Introduction

Transport, telecom and other companies aim to fulfil the customer's individual needs through a collaborative technology ecosystem. In manufacturing, one of the advancements in this new technological era is predictive maintenance. Equipment failure or unplanned downtime will have an adverse effect on planned operations as the necessary repairs cost time and money. One of the reasons for these occurrences derive from the fact that people are used to only have to repair their equipment too late, causing their production lines to slow or stop.

The goal of predictive maintenance is to predict and prevent unexpected events before they even happen by drawing upon huge sets of data and by doing so, they can proactively schedule the optimal time for maintenance, significantly reducing or even avoiding downtime. Properly maintained equipment lasts longer, cost less to maintain and operate more effectively, improving throughout and overall profitability.

1.1 Motivation

Leading network operators in the telecommunications industry perform preventive maintenance actions, ensuring lower operating costs and greater network stability. Doing so also allows efforts and investments to be focused on developing innovation.

Based on defined time intervals, this approach determines when future maintenance activities will be appropriate. These techniques are designed to help determine the condition of the equipment, thereby reducing the risk of equipment failure or performance degradation.

In the telecommunications industries, service perturbances can happen when a technician causes damage to the services of other clients. These perturbances often lead to new maintenance operations. By applying data mining techniques to network signals and technical interventions, it is possible to extract and create knowledge that allow us to predict the probability of perturbances for each intervention.

1.2 Objectives

This project aims at predicting the technical interventions that can perturb the service of other customers, i.e. predictive maintenance. To fulfil this objective, we apply machine learning approaches and study their combination with strategies for dealing with imbalanced domains. Then, we carry out empirical studies based on real-world data provided by the partner institution.

The accomplishment of this work lead to the following main contributions:

i. a review of related work is presented on imbalance domains and predictive maintenance;

ii. a dataset of network signals and technical interventions is presented followed by an exploratory analysis of this data;

iii. the prediction of perturbances in the services is structured as imbalanced domain learning task;

iv. an evaluation of the predictions approaches is presented.

1.3 Structure

This thesis is organised into four more chapters described bellow.

Literature Review Provides a review of previous work focusing on the most important concepts related to this project, such as data mining techniques, machine learning models, approaches to deal with imbalanced domains and suitable performance metrics for these domains and also, some time series concepts. This chapter ends with similar studies performed by other researchers.

Anticipation of Perturbances The third chapter describes with more detail the context of our target application, the methodology to solve the problem and a description of the dataset that we work on.

Experimental study Presents the experimental study, detailing the learning algorithms used, as well as the strategies for imbalanced domains and the performance indicators. This chapter also includes the experimental methodology followed by the results of all the implemented techniques.

Conclusions The fifth and final chapter presents a summary of the motivations and main contributions given by this work. Finished with future work directions that can go deeper into this subject.

Chapter 2

Literature Review

In this chapter, the methods and techniques necessary to achieve our objectives are reviewed. It starts with a brief overview of data mining, followed by the explanation of some machine learning algorithms. Next, techniques for handling imbalanced domains are introduced, and finally time series concepts. The chapter ends with the study of some similar work.

2.1 Data Mining

The amount of stored data from industrial activities makes it difficult to analyse them without the use of automated analytical techniques. With this problem in mind the area of Knowledge Discovery in Databases (KDD) emerged with new tools and computational techniques. These new techniques are mostly data mining approaches to extract useful knowledge, patterns, and tendencies.

The KDD process is a set of continuous activities that share the knowledge discovered from databases. According to Fayyad et al. [29] this set is composed of five steps: data selection, pre-processing and data cleaning, processing of data, data mining, interpretation and evaluation of results (cf. Figure 2.1). Data mining is a very important step in the KDD process for the resolution of this problem. It is the application of algorithms for extracting information from the

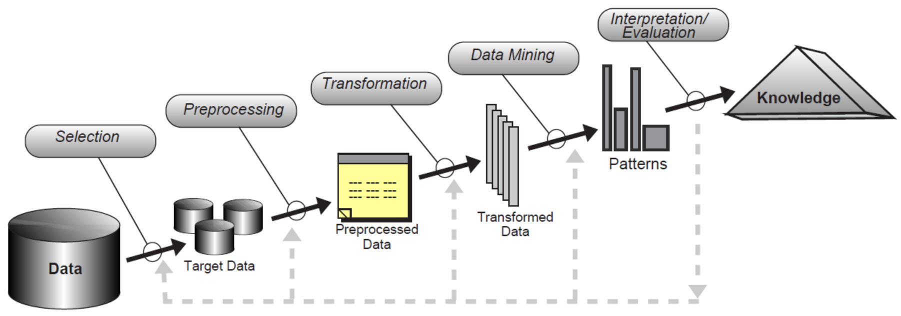


Figure 2.1: KDD process [29]

data.

The objective of Machine Learning is to understand the way data relate. It is based on algorithms that can learn models from the data and make predictions. Machine learning tasks are categorised as supervised, unsupervised or semi-supervised. In supervised learning the goal is to learn a mapping function from a set of predictor/independent variables to a target/dependent variable. When the predictor variables are known but the target variable is unknown, then this is a case of unsupervised learning. In semi-supervised learning, some of the target values are known.

Supervised learning tasks includes two categories of algorithms: classification and regression. The goal of classification is to predict a nominal target variable, i.e. classes. The classification model results from the analysis of the training data set (i.e., set of examples for which the class labels are known) and is used to predict the class label for which the class label is unknown. For example, a credit card company that receives hundreds of thousands of requests for new cards. The requests contain information about several different attributes, such as annual salary, any outstanding debts, age, etc. The point of classification, in this case, is to categorise the requests into those who have good credit or bad credit.

Whereas classification is discrete and has no order (categorical labels), regression is used to predict a numeric or continuous value. For example, a regression model can be used for predicting house price values. In addition to the value, the data may have the age of the house, square footage, number of rooms, number of floors, having a garage, and so on. The house value would be the target and the other attributes would be the predictors. A regression model estimates the value of the target as a function of the predictors for each case.

This thesis will focus on solving classification tasks, for which several learning algorithms are described below.

2.2 Classification Algorithms

The goal of classification tasks is to obtain a good approximation of the unknown function that maps predictor variables toward the target value. The unknown function can be defined as $Y = f(X_1, X_2, ..., X_p)$, where Y is the target variable, $X_1, X2_, ..., X_p$ are features and $f()$ is the unknown function we want to approximate. This approximation is obtained using a training dataset $D = \{\langle x_i, y_i \rangle\}_{i=1}^{n}$.

There are many classification algorithms available, but we will only focus on the seven algorithms used in this study: Decision Tree, Random Forest and XGBoost, also on Support Vector Machine (SVM), k-Nearest Neighbours (KNN), Logistic Regression and Artificial Neural Network (ANN).

2.2.1 CART Decision Trees

Decision trees can be used for classification or regression problems. These are structures used to represent data formed by a set of elements that store information on nodes. Each tree has a node called root, which has the highest hierarchical level (the starting point) and links to other elements, called leaves (cf. Figure 2.2). The node that does not have a leaf is known as a terminal node. With these concepts clarified we can conclude that a decision tree is nothing more than a data structure that stores rules on its nodes (attribute), in which each link represents the decision to be made (rule) and the leaves represent an outcome (categorical or continues value for the target variable).

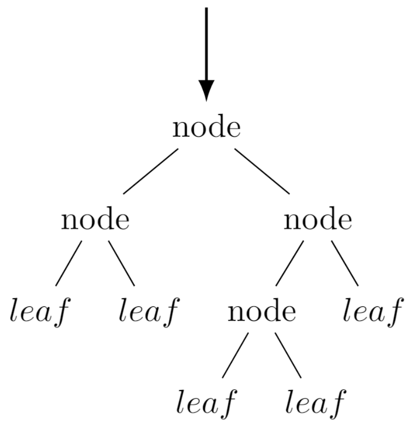


Figure 2.2: Decision tree example

There are some algorithms to build a decision tree, but we only focus on the CART algorithm proposed by Breiman et al. [9]. This algorithm uses a divide-to-conquer strategy: a complex problem is decomposed into more simple sub-problems. Recursively, the same strategy is applied to each sub-problem. In the execution of the classification task through the CART algorithm there are four elements involved in growing a tree:

- Set of binary questions

  Binary questions are used to divide each node. In cases where the answer is yes, it follows to the left node and in those that the answer is no, to the right node.

- Division

  The first task is to find out which of the attributes performs the best division. The criterion used by the algorithm to measure the impurity of a node is the Gini index, created by Gini [33] and is used to select the best division of the data. A node is pure when all cases belong to a single class. According to this criterion, if a dataset D contains examples from c classes, the impurity of a node is given by:

$$Gini(D) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.1}$$

where p_i is the relative frequency of class i in D.

After splitting D into two subsets D_1 and D_2 with sizes N_1 and N_2, the Gini index of the split data is defined as:

$$Gini(D) = \frac{N_1}{N} gini(D_1) + \frac{N_2}{N} gini(D_2) \tag{2.2}$$

The Equation 2.2 corresponds to the weighted average of each branch index. This procedure is performed for all the predictor variables. The one chosen for the division of the node will be the one with the lowest Gini index.

As this process is recursive, the binary tree is divided until the nodes are as pure as possible or until neither node can be further divided by some stopping criterion (e.g. the number of cases in the node).

- Associating a prediction with leaf

  The criterion used to associate a prediction with the leaf is the assignment of the most likely class or value within the target variable of the examples on this leaf.

- Pruning

  When decision trees are constructed, some sub-trees may reflect noise or errors causing overfitting. The pruning method is used to detect these sub-trees, whose objective is to improve the rate of success of the model for new examples, which were not used in the training set.

2.2.2 Ensembles of Decision Trees

An ensemble method is a combination of multiple learning algorithms. In order to deal with our particular case, we choose ensemble decision trees. Although decision trees are conceptually simple because of easy interpretation and they do not require much data processing, they are very powerful. Nevertheless, individual decision trees tend to overfit on its training data since they have low bias and high variance [54]. This problem can be mitigated by training multiple distinct trees and aggregating their predictions. By combining the result of many decision trees the variance is reduced while maintaining low bias.

Some of the most commonly used ensemble methods include Bagging and Boosting. Bagging, also known as bootstrap aggregating, was introduced by Breiman [10]. It is a technique used to reduce forecast variance that combines the results of several classifiers using averages, modelled on different sub-samples of the same dataset. Boosting is based on the question of whether a "weak" learning algorithm can be converted into a "strong" learning algorithm, posed by Kearns and Valiant [39].

Whereas in Bagging each model is built independently (parallel), in Boosting it is sequentially. Initially, each observation has equal weights. If the classes are predicted incorrectly, the misclassified observations get higher weights. The weights are redistributed after each training.

The new dataset is created via sampling, where previously misclassified observations will have a higher probability of being selected.

In the following, two of the most known ensembles methods are explained: Random Forest and XGBoost, which are based on the Bagging and Boosting strategies, respectively.

2.2.2.1 Random Forest

The Random Forest algorithm introduced by Breiman [11] is an ensemble method. As the name suggests, this algorithm creates a forest with a large number of decision trees, which are trained by performing Bagging to reduce the variance and prevent the over-fitting. This procedure randomly selects both a sample of features and examples from the original training dataset. Each of the obtained datasets are then used to construct each of the decision trees composing the Random Forest. Each classifier tree is pointed as a predictor component.

In the case of classification tasks, Random Forest constructs its decision by counting the votes of the predictor components in each class and then selects the winning class in terms of the number of votes accumulated, i.e. majority voting. The process of this algorithm is represented in Figure 2.3: the first phase consists of training each decision tree with data subsets from the training set. Then, the test cases are classified by majority vote.

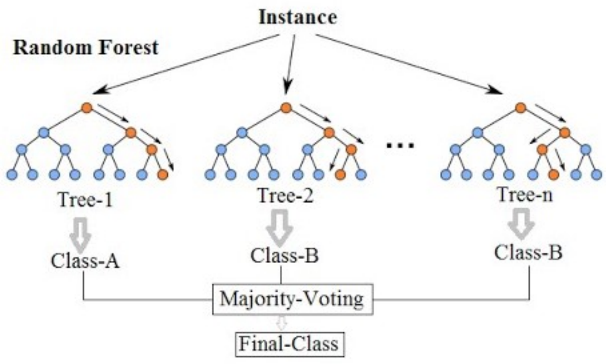


Figure 2.3: Example of Random Forest algorithm [36]

2.2.2.2 Gradient Boosting

The Gradient Boosting algorithm, created by Friedman [30], follows the same logic as Boosting [39]. It uses several iterations of a weak predictor, such as decision trees, to create a more robust predictor with superior performance. It begins by training a decision tree to generate predictions for each observation. Those predictions are used to determine a loss function that will be fitted a new model. This model will be added to the ensemble. Thus, the goal is to optimise the loss function of the previous learner by adding a new model that adds weak learners to reduce the loss function, making the current learner more effective than the previous one. Equation 2.3

represents the loss function that indicates a measure of how good the model coefficients are by adjusting the underlying data.

$$L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \sum_{i=1}^{K} \Omega(f_i) \tag{2.3}$$

where y_i and $\hat{y}_i$ are the true and the predicted target value for case i, respectively, and $\Omega(f_i)$ is used for regularisation at each f tree . The Ω function penalises complex trees, which is necessary to avoid overfitting.

XGBoost is an open source library created by Chen and Guestrin [16] and its name comes from an abbreviation for eXtreme Gradient Boosting. This model is a self-contained derivation of Gradient Boosting algorithm that focuses on computational speed and model efficiency. Since the Gradient Boosting computing the output at a very slow rate due to the sequential analysis of the dataset takes a longer time to execute it. The XGBoost emerged to improve these flaws. It supports parallelisation by creating trees parallelly, use distributed computing methods for evaluating any large and complex models, it also uses out-of-core computing to analyse huge and varied datasets and implements cash optimisation to make the best use of the hardware and resources.

2.2.3 Logistic Regression

In linear regression, the dependent variable is continuous and follows a normal distribution. The model allows estimating the average value of the dependent variable given a certain set of values of predictor variables. If the dependent variable is binary, the average of this variable is p, where p is the ratio of times the variable takes the value 1. To estimate the probability of p associated with a binary response, a technique called logistic regression is used. This technique was introduced by Agresti [2].

Figure 2.4 shows the main differences between linear and logistic regression. Basically, compared with linear regression, logistic regression is distinguished essentially by the fact that the independent variable is categorical.

The logistic regression model is displayed by an S-shaped function obtained by Equation 2.4. This expression known as the Sigmoid function does not admit negative values nor values greater than 1.

$$p = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}} \tag{2.4}$$

where x correspond to independent variables and the parameters α and β determine the logistic intercept and slope.

The General Linear Model (GLM) created by McCullagh and Nelder [43] is a generalised

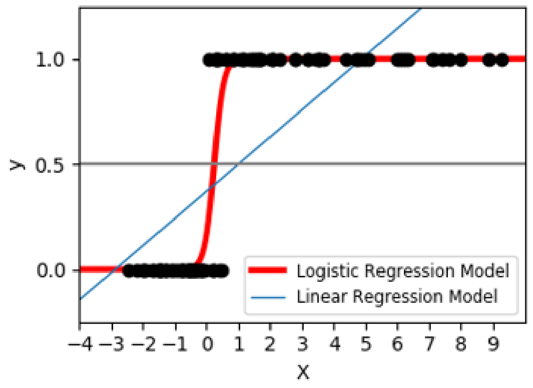
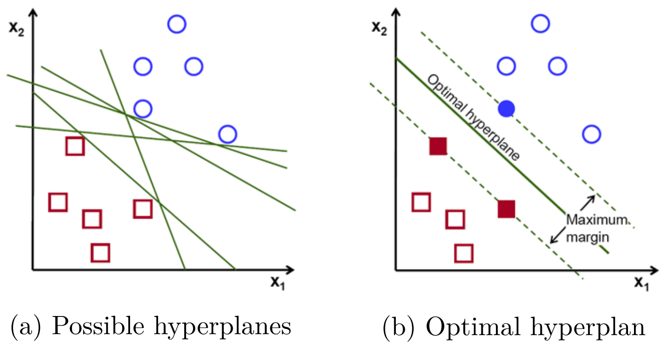


Figure 2.4: Example of Logistic Regression algorithm [55]

version of the linear regression. The GLM was formulated for many other statistical models, like logistic regression and Poisson regression.

2.2.4 Support Vector Machines

Cortes and Vapnik [21] proposed the Support Vector Machine (SVM) which consists of a method that tries to find the largest margin to separate different classes of data. The concept of SVM is to construct an optimal hyperplane so that it can separate different classes of data with as much margin as possible. In the following example (cf. Figure 2.5) we can see how SVM works. There are several straight lines that can be drawn to separate the data (cf. Figure 2.5a). The support vectors are data points that are closer to the hyperplane and they serve to choose the best hyperplane, represented by the filled points (cf. Figure 2.5b).



(a) Possible hyperplanes (b) Optimal hyperplan

Figure 2.5: Example of a Linear SVM [52]

The data is not always linearly separable. The approach used by the SVM to solve this type of problem consists of mapping the data to a space of higher dimension. For this, we need to introduce the concepts of soft margin and kernel trick. The idea of a soft margin is to allow some examples to be placed on the wrong side of the dividing hyperplane. The kernel transform non-separable data to separable data by adding more dimension [47]. Boser et al. [5] proposed nonlinear kernel functions to make linear SVM work in nonlinear settings. Table 2.1 provides some of the kernel functions. The Polynomial kernel (cf. Equation 2.5) represents the similarity of vectors on the training samples over polynomials of the original features. The Gaussian Kernel (cf. Equation 2.6) also called Radial Base Function (RBF) is a measure of similarity between two sets of features. The Sigmoid Kernel (cf. Equation 2.7) is similar to the Sigmoid function in Logistic Regression and uses an activation function that can be used for ANN. The r, p and γ are Kernel parameters that correspond to independent term, degree of polynomial function and Kernel coefficient, respectively.

Type of Kernel	**Formula**
Polynomial kernel	$K(x_i, x_j) = (x_i * x_j + r)^p, r \geq 0$ (2.5)
RBF kernel	$K(x_i, x_j) = exp(-\gamma \|\|x_i - x_j\|\|^2), \gamma > 0$ (2.6)
Sigmoid kernel	$K(x_i, x_j) = tanh(\eta x_i * x_j + v)$ (2.7)

Table 2.1: Kernel functions

2.2.5 k-Nearest Neighbours

The k-Nearest Neighbours (KNN), introduced by Cover and Hart [22], is a method based on distances that consider the proximity between the data in the realisation of predictions. The hypothesis of this method states that similar examples tend to be concentrated in the same region in the dispersion space of the data. Thus, given a new example to be classified, the KNN performs the following steps:

- calculates the distance between these examples and the others of the set, according to some measure of similarity;
- the k-nearest examples are selected;
- the example is classified in a certain category according to some criterion and grouping of the categories of the selected examples.

There are several distance metrics, and the choice of which to use varies according to the problem. The most used is the Euclidean Distance function.

Figure 2.6 shows the application of this method. In the centre is the test example that should be classified. The first step is to calculate the distance between this example and all other examples. Then the distances must be ordered from the lowest to the highest distance. If we considerer $k = 1$, the new example is assigned to class 1 since only exist one example that correspond to the class 1. If $k = 3$, the new example is assigned to class 2 because there are two examples of class 2 and only one of class 1 inside the circle defined by the 3-nearest neighbours. To avoid tie between classes the k parameter must be odd.

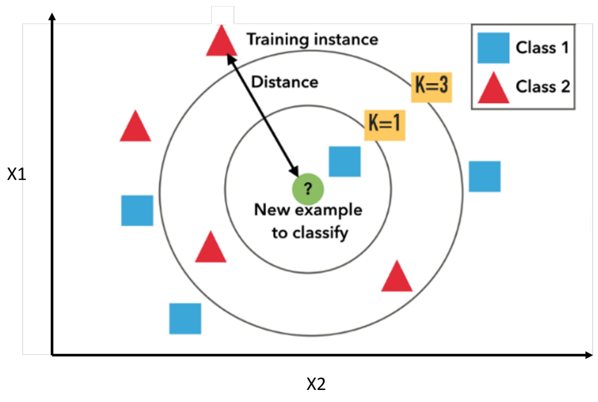


Figure 2.6: Example of KNN algorithm [1]

2.2.6 Artificial Neural Networks

An Artificial Neural Network (ANN) is a mathematical model inspired by biological neuronal networks. It can be defined as a complex structure interconnected by neurons, which have the ability to perform operations such as calculations in parallel, for data processing and knowledge representation. Mcculloch and Pitts [44] were one of the first to introduce this concept.

Artificial Neural Network Perceptron was later introduced by Rosenblatt [53]. This model deals only with a neuron classifying the result in a linear form. In Figure 2.7 the artificial neuron is a perceptron that calculates the weighted sum of several inputs, applies a function, and passes the result forward through an activation function.

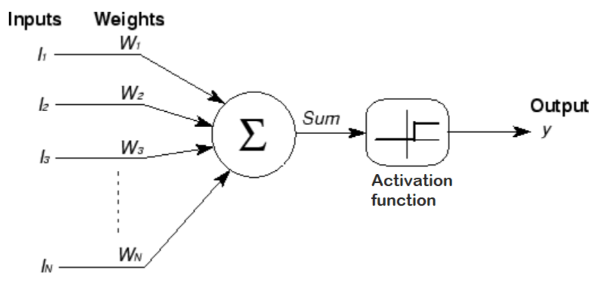


Figure 2.7: Schema of the artificial neuron model [38]

In order to deal with non-linear problems, layers of hidden neurons were added in the perceptron model, forming the Artificial Neural Network Multilayer Perceptron [37]. This new topology works as a progressive network. The output of one neuron connects to another neuron of the next layer formed by a set of neurons called nodes, as shown in the Figure 2.8. One of the learning methods is back-propagation that re-adjusts the weights of each neuron so that it can produce the desired output for a specific input at the end of the training.

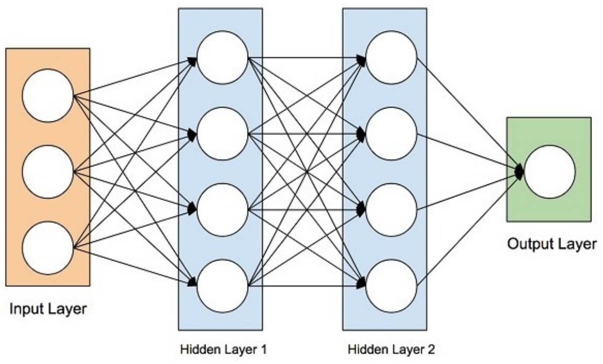


Figure 2.8: Artificial Neural Network Multilayer Perceptron model [63]

2.3 Imbalanced domain learning

The subject of this project faces an imbalance domain learning problem. This problem occurs whenever the user has an interest in cases that are rare and few exist in the training set. The combination of these two factors create obstacles at several levels. Namely, the models created by standard learning algorithms tend to be biased towards the majority class. Moreover, the evaluation metrics will not capture the competence of models in relevant cases [8]. Thus, it is necessary that the learning process focuses on the rare cases and that the evaluation metrics are biased towards the performance of the models in these rare cases.

2.3.1 Approaches for handling imbalanced domains

The problem of imbalanced data is a challenge when building a prediction model because it will bias standard classification methods over the majority class while having lower predictive accuracy over the minority class of interest. According to Branco et al. [8] there are four main strategies to solve this problem: data pre-processing, special-purpose learning methods, prediction post-processing, and hybrid methods. Figure 2.9 provides a general overview of these strategies.

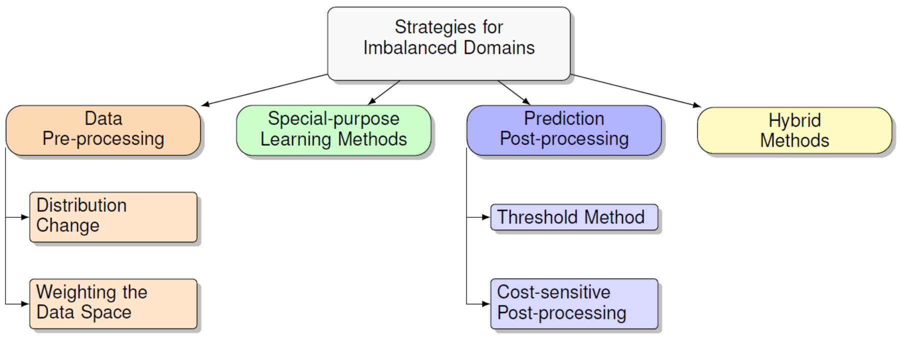

Strategies for Imbalanced Domains
Data Pre-processing
Special-purpose Learning Methods
Prediction Post-processing
Hybrid Methods
Distribution Change
Weighting the Data Space
Threshold Method
Cost-sensitive Post-processing

Figure 2.9: Approaches for handling imbalanced domain learning [8].

Data pre-processing

In the pre-processing strategies, the data will be pre-processed given the user preferences so that the learning algorithm is not applied directly to the training sample. Pre-processing strategies can be achieved both by Distribution Change or by Weighting the Data Space.

Distribution Change approach aims to balance the distribution of the classes in the training dataset using techniques that change the original distribution of the data. Estabrooks et al. [28] proved that changing the data distribution is an efficient solution to the imbalance problem. A technique for changing the data distribution is stratified sampling. There are several approaches to data sampling. The simplest ones are under-sampling, which decreases the number of irrelevant cases, and over-sampling, which increases the number of important cases. Another approach to deal with the data sampling is the generation of new synthetic data. The most popular algorithm is Synthetic Minority Over-sampling Technique (SMOTE) proposed by Chawla et al. [15]. As the name suggests, creates synthetic samples from the minority class instead of creating copies.

Weighting the Data Space is a form to implement cost-sensitive learning. This method changes the original data distribution by multiplying each example by a factor that is proportional to the importance.

Special-purpose Learning Methods

This technique focuses on modifying algorithms to better fit the user preferences. There are three main solutions: recognition-based methods in which the model is obtained with only examples of the target class, cost-sensitive algorithms where the costs are incorporated directly in the algorithm and development of new algorithms which are developed to specifically deal with this problem. There are several works that explained the transformation of different classification models into cost-sensitive ones (e.g. Maloof [42]; Akbani et al. [3]; Nunez et al. [48]).

Prediction post-processing

There are two main solutions to prediction post-processing: threshold method and cost-sensitive method. On the threshold method, each prediction is associated with a score that represents the degree to which an observation is a member of a class. Hernández-Orallo [34] explores several threshold choice methods. On the other hand, the cost-sensitive method associates costs to prediction errors and minimises the overall expected cost. Sinha and May [57] explored this technique that aims to change the model predictions for making it cost-sensitive.

Hybrid methods

The previous methods presents some disadvantages [6]. Recently, several contributions have been made from different approaches to dealing with these drawbacks. On this subject, the hybrid methods arose as a combination of strategies of different types. Basically, these methods take the main advantages of two selected strategies combining them into one. Estabrooks and Japkowicz [27] was one of the first to present hybrid methods.

2.3.2 Evaluation metrics

For these type of imbalanced domain learning problems, the standard evaluation metrics are inadequate and several measures have been proposed as alternatives to classification problems [8].

The confusion matrix gives the idea of what the model is getting right and what types of errors it is making. Table 2.2 shows the confusion matrix for a two-class problem where one of the classes is called positive and the other is called negative. In this table, the number of True Positive (TP) and True Negative (TN) are the instances that were correctly classified, and the number of False Positive (FP) and False Negative (FN) are the instances that were wrongly classified for each class.

		Predicted class	
		Positive	Negative
Actual class	Positive	TP	FN
	Negative	FP	TN

Table 2.2: Confusion matrix

The *Accuracy* is obtained from the confusion matrix and is frequently used for evaluating the performance of classification models. This metric can be defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{2.8}$$

In other words, *Accuracy* is the ratio between the correct predictions and the total number of predictions. However, it is not suitable for imbalanced domains because it can be misleading. For example, consider a two-class problem, where the majority class is 90% of the data, and the minority is 10%. If the classifier predicts every sample as belonging to the majority class, the *Accuracy* will be 90%, but this classifier is useless because the more interesting cases for the user, the minority class examples, are misclassified. In this context, the majority class examples will have a greater impact when compared with the least represented examples. The accuracy metric bias towards majority class is opposite to the user preferences. This phenomenon is known as accuracy paradox [69].

In the Table 2.3, are defined some examples of metrics, that take into account the data distribution according to user preferences. *Precision* assesses the proportion of positive predictions that were actually correct. This equation was defined by Kent et al. [40] as well as the *recall*, that assesses the proportion of actual positives that were identified correctly. The *specificity* assesses the cases that are identified as negatives and are in fact negatives. The *false positive rate* (FP_{rate}) is the ratio between the number of TP and the total number of actual negative values. Contrary, the *false negative rate* (FN_{rate}) is the ratio between the number of FN and the total number of actual positive values.

Measure	**Formula**
precision	$\frac{TP}{TP+FP}$ (2.9)
true positive rate (recall or sensitivity)	$\frac{TP}{TP+FN}$ (2.10)
true negative rate (specificity)	$\frac{TN}{TN+FP}$ (2.11)
false positive rate	$\frac{FP}{TN+FP}$ (2.12)
false negative rate	$\frac{FN}{TP+FN}$ (2.13)

Table 2.3: Evaluation measures based on confusion matrix

One problem with the previous metrics is that they give different but complementary insights regarding the model performance. Therefore, Table 2.4 presents some measures that result from the combination of these metrics.

Measure	**Formula**
Fscore	$\frac{(1+\beta^2)\times precision \times recall}{\beta^2 \times precision + recall}$ (2.14)
G-Mean	$\sqrt{sensitivity \times specificity}$ (2.15)
dominance	$sensitivity - specificity$ (2.16)

Table 2.4: Suitable metrics for imbalanced data regarding classification tasks

Fscore [51] is the association of *precision* and *recall*, and indicates how effective is the classifier regarding the positive class. If $\beta > 1$, a higher weight is assigned to *recall*, otherwise favouring *precision*. The greater the *Fscore* (best value is one), better the performance of the model.

Another commonly used metric is *geometric mean* (*G-Mean*), defined by Kubat and Matwin [41]. It tries to maximise the accuracies of two classes while obtaining good balance and it reaches the best value at one with the worst value at zero.

The most popular tools for dealing with imbalanced data are *Receiver Operating Characteristics (ROC)* curve and the *Area Under the ROC Curve (AUC)* [45] - both are alternatives to *Accuracy*. The ROC curve is a graph showing the trade-off between TP_{rate} and FP_{rate}. This curve plots TP_{rate} and FP_{rate} at different thresholds. The ideal model would obtain TP_{rate}=1 and FP_{rate}=0, i.e. should be closer to (0,1) point. Comparing many models through *ROC* curves is inefficient. However, there is *AUC*, which is more efficient in such scenarios. *AUC* measures the two-dimensional area underneath the entire *ROC* curve. In Figure 2.10 the blue dash is the *ROC* curve and the shaded area is AUC.

Still, *AUC* and *G-Mean* can yield the same result for several different combinations of FP_{rate}

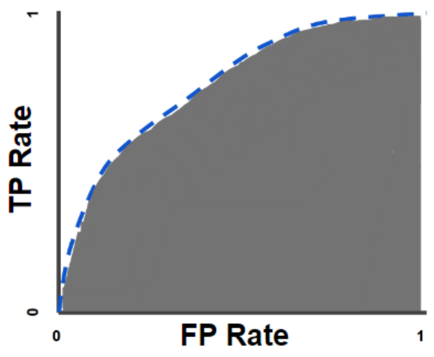


Figure 2.10: Example of AUC [25]

and TP_{rate} and, therefore, García et al. [32] proposed *dominance* (cf. Equation 2.16) that solves this issue. A *dominance* of +1 means that all examples of minority class were predicted correctly and all negative class examples were misclassified. Otherwise is -1, that corresponds to the opposite situation.

2.4 Time series

The concepts of data mining also are suited to analyse time series [26], revealing hidden patterns and predicting events. A time series is a collection of observations obtained chronologically. In this type of data the neighbouring observations are more closely related than more distant observations.

The main features of many time series are *trends* and *seasonal variations* [23]. In general, the *trend* is a systematic linear or nonlinear component that changes over time and that does not appear to be periodic. The *seasonal variation* refers to the repeating patterns within any fixed period, it is the opposite to the *trend*. Also the *noise* is an important feature for the analysis of a time series since describes random variations or unforeseen events. Checking whether or not the time series has *cycles* is also important. A *cycle* is a component that reflects repeated but non-periodic events. If the data does not show *trend*, *seasonal* effects or other time-dependent characteristics, then the time series is *stationary*.

A time series can be *regular* or *irregular*. It is called *regular* if there is an equally spaced interval of time [14]. An *irregular* time series is the opposite situation. Natural disasters like volcanic eruptions or earthquakes occur at irregular time intervals since the spacing of observation times is not constant.

There are some different performance measures for time series forecasting. The *Mean Absolute Percentage Error (MAPE)* and the *Mean Absolute Scaled Error (MASE)* are the most common. *MAPE* was defined by Armstrong and Collopy [4] and expresses the ratio of error to the actual values as a percentage. In the Equation 2.17, y_{t+i} is the actual value and $\hat{y}_{t+i}$ is the forecasted value i-steps ahead at data point t. Hyndman and Koehler [35] proposed *MASE* (cf. Equation 2.18) that is a measure of the accuracy of the forecast. Basically, this metric scales the errors using the *in-sample Mean Absolute Error (MAE)* from the naive forecast method.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_{t+i} - \hat{y}_{t+i}|}{y_{t+i}} \tag{2.17}$$

$$MASE = \frac{MAE}{\frac{1}{n-1}\sum_{i=2}^{n}|y_{t+i} - y_{t+i-1}|} \tag{2.18}$$

where $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$.

A time series is a sequence of real numbers, $t_1, t_2, t_3, ..., t_k$. Suppose that the time series consists of observations equally spaced in time with a set of points t_i, $i = 1, 2, 3...$, in $\mathbb{R}^k$. This sequence of observations is called as time-delay embedding [59]. The time series can be embedded into a dataset in the following manner (cf. Table 2.5). Taking successive $k - tuples$ from the sequence $t_1, t_2, t_3, ..., t_k$.

Predicted variables			Target
t_1	t_2	t_3	t_4
t_2	t_3	t_4	t_5
t_3	t_4	t_5	t_6
...	...	...	...
t_{k-3}	t_{k-2}	t_{k-1}	t_k

Table 2.5: Embedded time series

In many cases, the time series are embedded in datasets enabling the usage of standard learning algorithms to be applied without violating the temporal relationship between observations. The embedded time series is the case of our approach in this study.

2.5 Predictive maintenance in telco

In recent years there have been some case studies of predictive maintenance in the telecom industry, like mobile phone towers, power equipment, etc. Corazza et al. [20] have implemented a predictive maintenance approach for mobile service providers to predict failures in mobile phone networks. The purpose of the problem was to predict the alarm signals in a cell that are the cause of the malfunction. This work faced the imbalanced domain problem since the alarm signals represent only 1% of the total. To solve this issue, all the available positive samples were maintained, which were those with a smaller number of occurrences and the negative samples were randomly selected. This means that the random under-sampling technique was used.

The Mobile Broadband Network Ltd (MBNL) provides and manages the entire telecommunications infrastructure for two operators. When there are problems on the network, the company records them in a system. The purpose of these records was to be able to predict fan and air conditioner failures. The failures in such equipment can overheat the mast and induce mast downtime affecting network performance. To solve the problem the MBNL use the Natural Language Processing [13].

China Mobile collects a time series of network data and analyses the anomalies. The company uses algorithms that are applied to data to build failure prediction models for different types of services. They later discovered that this intelligent prediction can identify network failures several hours in advance [24].

The aim of this thesis is to try to prevent the customer from complaining about the maintenance in telecommunication services. Although this thesis involves imbalance problem like the case study from Corazza et al. [20], we use different approaches to solve it. We apply other pre-processing techniques and also post-processing to bypass the class imbalance. With such strategies, it is expected to improve the operation of the partner telecommunications company using machine learning on data containing network signals, router status and information on technical interventions.

2.6 Summary

Throughout this chapter, we study the most appropriate techniques and approaches to anticipate the service perturbance in telecommunications. In the next chapter, we introduce the problem as well as the methodology that was used. Since we work on a specific company dataset, it is important to understand the available data and which transformations should be made to obtain a dataset that best describes the problem. Therefore, we present a detailed description of the data and the feature engineering steps applied.

Chapter 3

Anticipation of Perturbances

In this chapter, we describe in detail the problem subject of this thesis and the methodology to solve it. We perform some exploratory data analysis on the data supplied by the partner institution and we proceed to several data transformations so to accomplish a new and richer dataset for further analysis.

3.1 Problem Definition

In an extremely competitive market, telecommunications companies need to focus on maintaining their customers as satisfied as possible with the services provided. One of the factors that impacts negatively customer satisfaction is perturbances in service.

In our context a perturbance can be considered an unusual change in the quality or behaviour of a service caused by some event. As such, the Quality of Service (QoS), that is a very important factor for many industries like telecommunications, can be compromised. These small perturbances in the service are an annoyance to the costumers and can possibly lead to a loss to the company since the user might ask for a change of provider.

Whenever there is a perturbance in a Distribution Point (DP), such as installations/disconnections in the network, it is necessary the participation of a technician. When those interventions are performed, there is an underlying probability that it will affect the services of other costumers in the following days which can lead to a Technical Participation (TP) or Technical Order (TO). In order to prevent these events from happening, it is necessary to forecast the probability of complaint or technical order for each intervention executed by technicians and to enable the planning of some preventive actions to reduce the number of interventions.

However, there are two types of errors that must be accounted for in this setting: no plans should be made when there is no need and, most importantly, no perturbance occurrence should be missed, as the cost of not taking actions is very high.

3.2 Methodology

Many of the problems involving data mining solutions use the Knowledge Discovery in Databases (KDD). As this process does not focus on business issues or model generation, but on the discovery of knowledge from the data, we follow the CRISP-DM (Cross-industry standard process for data mining) model [56]. In Figure 3.1, a detailed diagram of the process is shown.

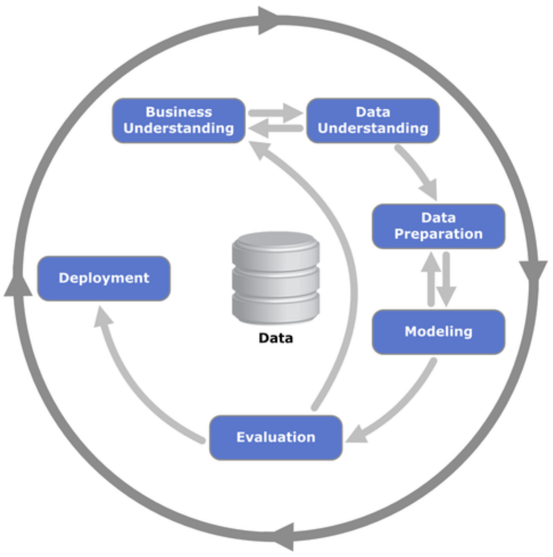


Figure 3.1: CRISP-DM process diagram [65]

Business and data understanding

The proposed methodology begins by understanding the main concepts about the telecommunications market in order to get a better insight into the problem that we intend to solve. In this phase, the business questions are translated to data mining goals. Once the partner institution provides the data, the workflow begins with a prior analysis to inspect the various attributes and verify data quality.

Data preparation

The data is spread across different sources. It is therefore important to collect this data and integrate it into a single structure. After integrating data, we consider a few criteria for data selection. Then, we clean the data, correcting or removing wrong records, analysing them from the perspective of time series. Subsequently, if the resulting data does not have enough features, we need to resort to a feature engineering techniques.

Modelling and evaluation

The last phase of this methodology is to select a model that is adequate to the problem. Since we are dealing with an imbalanced dataset, an approach to decrease the imbalance present on the data is made. This step is very important since we want learning algorithms to focus on accurately anticipating rare cases. Finally, a classification model is built with the training set and with suitable performance estimation metrics, and evaluated concerning how accurately it predicts unseen labels in a test set.

3.3 Dataset description

The data that we work on is divided between many aggregators that are distributed in the network (cf. Figure 3.2). The HUB is the maximum aggregator and there are only a few in the whole country. The Cable Modem Termination System (CMTS) is a router that centralises all communication with the Cable Modem (CM) of an Hybrid Fiber Coax (HFC) structure such as NET, cable TV, and others. At the end of the topology, there is a Distribution Point (DP). Usually DP serves multiple buildings.

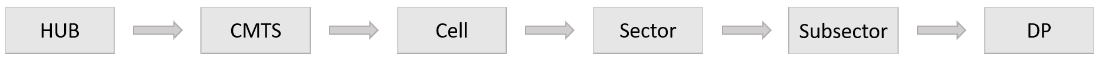


Figure 3.2: Network topology

The data provided by the partner institution has a large amount of observations as each corresponds to a daily record of network signal behaviours for the whole country. This information regards the number of hourly events per day above or below a threshold (defined by the partner institution). The partner institution has also provided data about TO, TP and the locality of the devices. The data period is from October 2018 until December 2018. Table 3.1 gives a succinct description of the most relevant features from the two tables of data. The table "Client Vision" contains information about the network signals, address information, router status, TOs, and TPs dates. The table "TO" contains additional information about TOs, like the type of problem, affected equipment and others. Each feature on the signals and router status can be represented as a time series. The dataset is composed of these several embedded time series.

Table	**Features**	**Description**	**Information**
Client Vision	TX power	Transmitted signal power in the up/downstream carrier	Signal
	RX power	Received signal power in the up/downstream carrier	
	Cer	Ratio of downstream errors received by the CM	
	SNR	Signal-to-Noise Ratio (SNR) of up/downstream related to CM	
	Sum up/down	Sum of up/downstream kpis	
	Median up/down	Median of up/downstream kpis	
	DP	Distribution point	Address information
	Building address	Address of the building	
	Unique Address	House identification of the building	
	Not online	CM status is not online	Router status
	Reboots	Number of daily reboots	
	TO date	TO day and hour	Dates
	TP date	TP day and hour	
TO	Symptom area TO	Problem type	Additional information on TOs
	Equipment	Type of affected equipment	
	TO situation	TO status	
	TO delegation	Zone	

Table 3.1: Description of the main features of the initial dataset

As an example, Figure 3.3 depicts the SNR of downstream behaviour for all customers in a particular building that went through a technical intervention. The x-axis represents the period of time and the y-axis represents the number of hours the signal was below the threshold. Thus, above zero means that the signal has been disturbed. First, the "Neighbour 12" made a call to the customer service and in the next day there was a TO, that is, the technician went to the that customer's home to solve the problem. With this technical intervention, the SNR of the neighbours of this client fired immediately causing the "Neighbour 7" to complain and suffered a TO later.

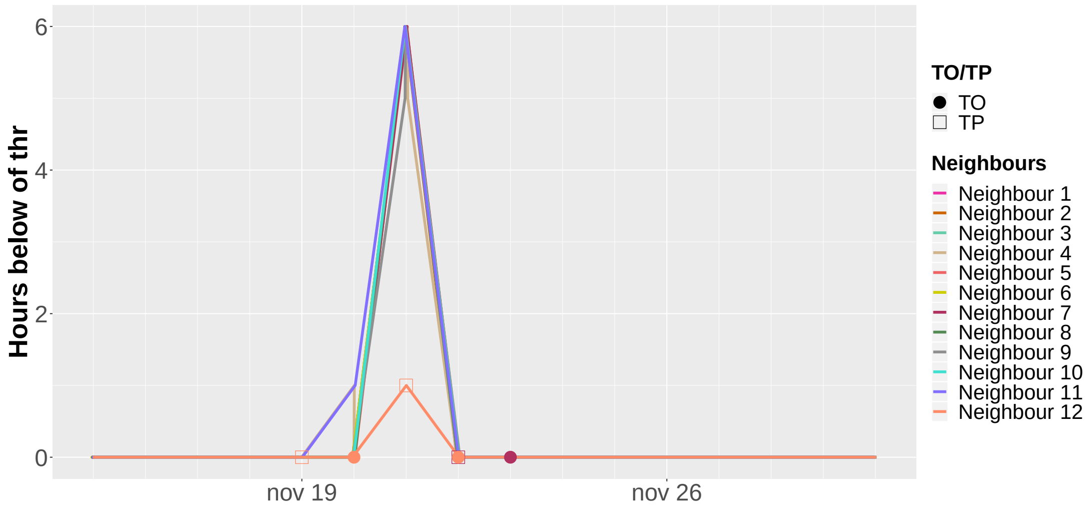


Figure 3.3: Signal-to-Noise Ratio (SNR) behaviour after a Technical Order (TO)

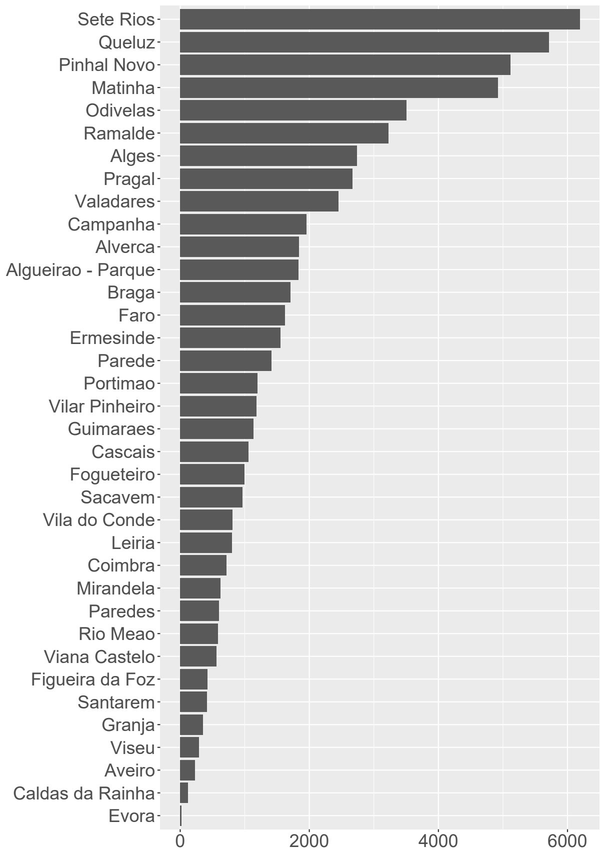
3.4 Data preparation

Due to the large dataset size, the partner company needed to split the dataset into smaller parts and passed us by a sharing folder because this data was not available in the data warehouse. Once we received the data, we joined some parts together and used HUE [19] to upload it into a data warehouse. The HUE is an open source SQL Cloud Editor that allows querying and visualising data.

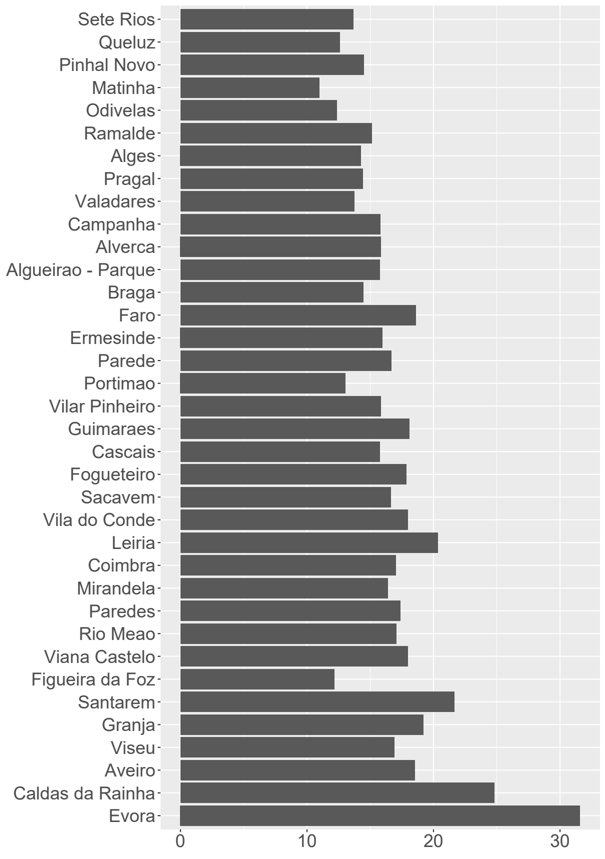
Then, we put together a single data set with the Zeppelin notebook [68]. Apache Zeppelin is a web-based notebook that supports a group of Spark interpreters [58], such as PySpark that we chose to use as we establish as the most mature. The partner institution supplied a virtual machine with the Zeppelin (2.2.0.cloudera2) [18] and thus we have access to the HUE via Zeppelin.

As the data came from two different tables we had to combine them together. As we did not need all the data, we made a selection that only considered buildings with more than two apartments. Also, we only considered the period of 5 days before and 7 days after a TO to compare the oscillations of the signals when this TO occurs. Initially, the dataset consisted of 70.753.957 observations, and with this selection, the dataset was drastically reduced, from tens of millions of observations to hundreds of thousands. Thus, it was possible to continue the pre-processing with R software [50].

In this stage we proceeded to the initial data exploratory analysis illustrating the devices per locality (in tens of thousands) and the TOs percentage that happens in those devices (cf. Figure 3.4). It can be observed that there is no direct relationship between the two graphs. And it is also verified that the percentage of TOs is globally identical in all localities.



(a) Mac address per locality (HUB)



(b) TOs percentage per locality

Figure 3.4: Devices distribution and corresponding percentage of TOs

The initial data did not contain any variable that could indicate whether a technical intervention was contaminant or not, so we had to construct it. The definition of a TO contaminant follows the principle that after the next day of a TO in a particular apartment in a building, a TO or a TP occurs in the following 6 days in some neighbour (cf. Figure 3.5). We chose to consider the starting date as the day after the TO because the records are daily, and if a TO occurs at the end of the day, the changes in the signals may not be noticeable on that day. The decision to only treat the next 6 days is because it is the standard time that a signal needs to stabilise. The TPs are important in the construction of the target because if an intervention happens in an apartment and a neighbour suffers with it, then it is expected that this neighbour calls to solve its problem and then there is an intervention.

In order to create relevant variables, it is important to note that the affected signal improves, worsens, or holds after an intervention. We then created the following variables: the aggregated maximum, mean, variance of each signal on the previous 5 days and the following TO day;

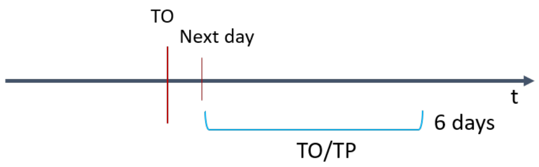


Figure 3.5: Time period to consider a Technical Order (TO) contaminant

maximum and minimum difference for each previous day in relation to the following day. We also added the numbers of neighbours for each apartment and the number of TPs before a TO. In total 149 features were created.

After all these steps, it is necessary to clean the data. We started by removing some columns that contained few unique values or even just a single value. In order to deal with missing values, we use two different approaches. In the attributes "n_reboots" and "avg_reboot_duration", the missing values were replaced by zero. The minimum value is 1 which means that only records in the positive case occur - when there are reboots on the device. In this case, it is reasonable to fill the missing spaces with zero. In the attributes of mean and median of up/downstream and standard deviation of signals, the *KnnImputation* function of the DMwR package [61] was applied with the method of weighted average. With so many variables it is necessary to confirm the relation between them, and for that reason we use the correlation coefficient with significance level of 0.05. The *findCorrelation* function of the package caret [31] identifies the variables correlated for a confidence level at 95% (cf. Figure 3.6). With this auxiliary help, we identified a correlation between *(1)* the mean of the Rx power signal in the downstream carrier and the variance thereof for the days prior to a TO; *(2)* the mean of the SNR in the downstream carrier and the variance for the previous days; *(3)* the correlation between the variance of signal cer and the mean for the day following a TO in the downstream carrier. The output of this function suggested the removal of variables that are related to the mean and, as such, we removed them.

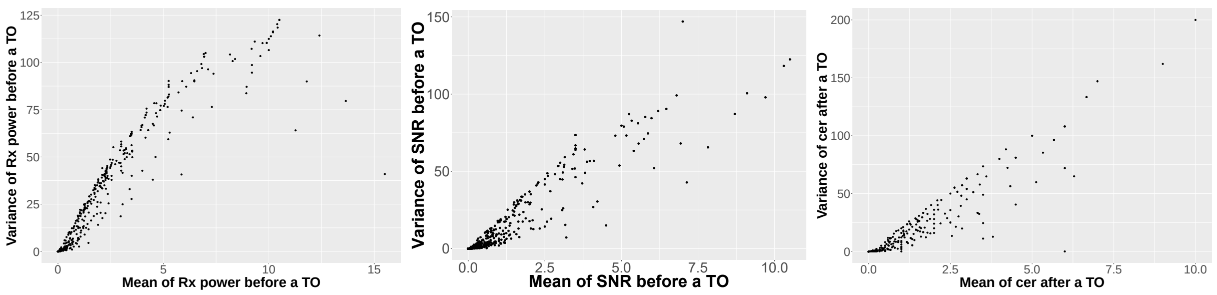


Figure 3.6: Top 3 correlated variables for significance level at 5%

At the end of the data cleaning process we excluded 15 variables of the 149. All the 193 features are described in detail in Table A.1 in Appendix A.

The last process was to prepare a dataset for the machine learning models. The categorical features (additional information on TOs, address information and service account) of our dataset had too many levels. This reduces the performance level of the model [66]. Thus, we select only the TO observations and the numeric features (network signals, router status and TPs).

Figure 3.7 is a summary of the entire data preparation methodology. At the beginning we have a dataset with millions of observations and we ended with a sharp reduction in size.

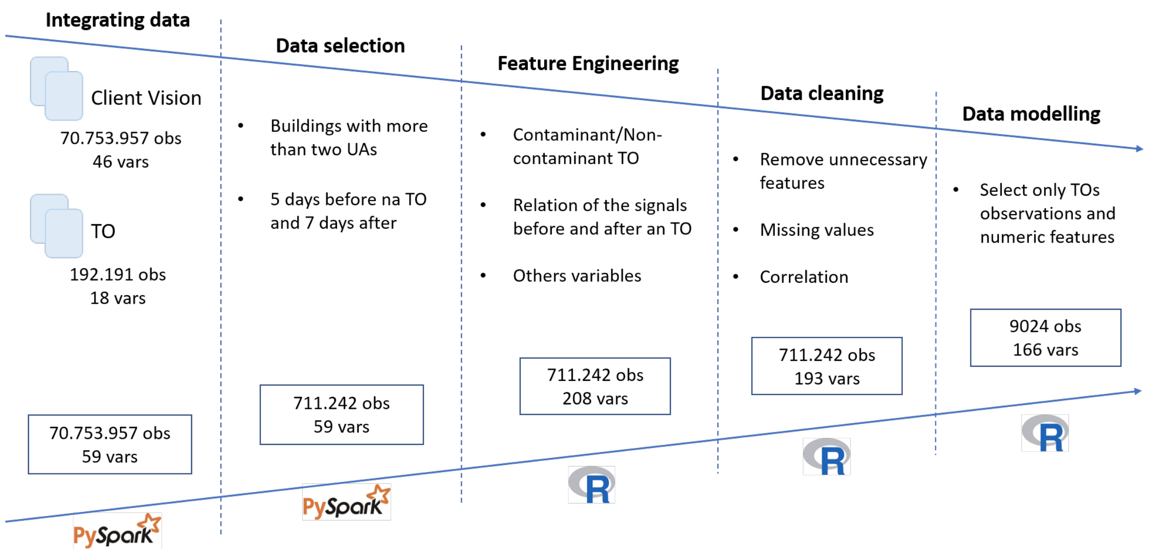
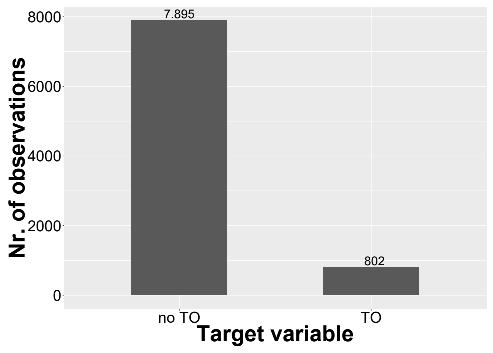
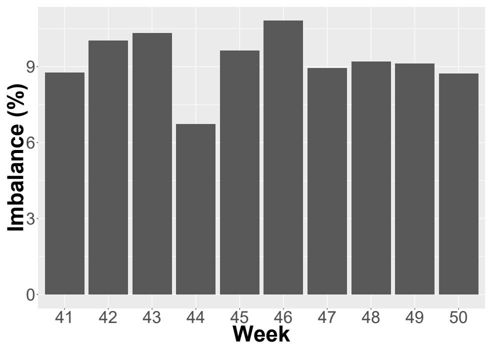


Figure 3.7: Data preparation methodology

As previously mentioned, this predictive modelling challenge is known as an imbalanced domain task - Figure 3.8a shows the difference between the two classes (contaminant/non-contaminant TO). The data has a very low percentage of possible contaminating TOs which only accounts 9,22% of the total TOs. Also, the percentage of possible contaminating TOs in each week is not constant as it can be seen in Figure 3.8b.



(a) Imbalance in entire dataset



(b) Imbalance of target per week

Figure 3.8: Distribution of the target variable on dataset

An intervention can be made to disconnect devices (when the customer ceases the contract), maintenance, installation (new customer) and alteration of service. Calls can be distinguished between complaints and requests from the customer. However, both usually lead to the scheduling of an intervention. Figure 3.9 shows the relationship between technical interventions and calls after the first TO occurring in each building during the following 7 days. Most of the days on which there are no, or almost none, TOs records correspond to Sundays.

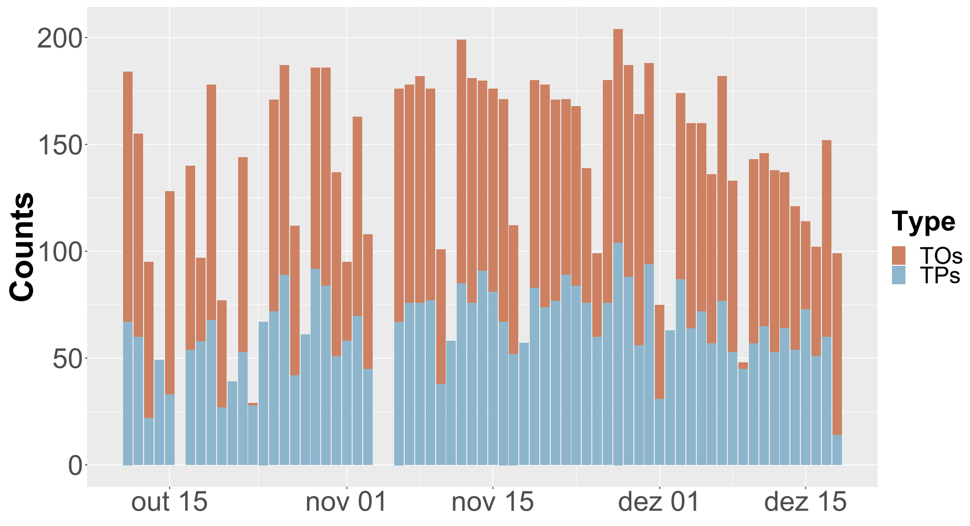


Figure 3.9: Relationship between TOs and TPs after a TO

3.5 Summary

In this chapter, we realised what kind of data we were provided and built a dataset with richer information. We have performed data understanding and preparation phases of the CRISP-DM methodology. We reduced the initial dataset with millions of observations to thousands. We built new variables to notice changes in signals after a technical intervention, and we also built the target variable and others such as the number of neighbours and the number of TPs before a TO.

Next, we apply the last part of the proposed methodology, modelling and evaluation. We start by applying machine learning algorithms and strategies to solve the imbalance domain problem. Finally, we present and discuss the obtained results from this methodology.

Chapter 4

Experimental study

In this chapter we describe the methods applied to the development of the project. The first step is to detail the algorithms used as well as the techniques for dealing with the data imbalance, and the evaluation metrics. Then, we present the experimental setup and the results, along with a discussion.

4.1 Experimental setup

The solution to the subject of this thesis involves the application of classification algorithms. In this section, the algorithms that were used as well as the chosen strategies to handle imbalanced domains and the experimental methodology is described.

4.1.1 Learning Algorithms

In order to have a good sense of the data and what models fit it better, it is crucial to select different types of models. For this purpose, we select tree based, ensemble, distance based and artificial neural networks models. The CART Decision Tree [9], Random Forest [12], XGBoost [16], Support Vector Machine (SVM) [21], General Linear Model (GLM) [43] for Logistic Regression, k-Nearest Neighbours (KNN) [22] and Artificial Neural Network (ANN) [44] were used.

The Decision Tree algorithm was the first to be tested. This model consists of training a single decision tree using the CART algorithm (Subsection 2.2.1). We use the rpart package [60] and took into account the *minsplit* and *cp* parameters. The *minsplit* is the minimum number of observations that must exist in a node to make the division and the *cp*, complexity parameter, is a combination of the size of a tree and the ability of the tree to separate the classes of the target variable. If the best division in the growth of a tree does not reduce the overall complexity of the tree to some extent, the split will not be executed.

After we tried with a tree we headed for an ensemble of trees, the Random Forest (Subsection 2.2.2.1). To apply this algorithm we use the ranger package [67] with the *mtry*, *num.trees* and *max.depth* parameters. The *mtry* is the number of features that are chosen randomly to split at in each node. With the *num.tree* we specify the number of trees. The *max.depth*, as the name suggests, is the maximal tree depth.

Another model that using trees is the XGBoost, but this one follows the boosting technique (Subsection 2.2.2.2) while the Random Forest follows the bagging. In this case, we use the xgboost package [17] with the parameters *eta*, *max_depth*, *nround* and *colsample_bytree*. The *eta* parameter controls the learning rate. Such as *max.depth* of ranger, the *max_depth* of xgboost controls the maximum depth of the trees. The maximum number of iterations is defined by the *nround* and, lastly, *colsample_bytree* controls the number of variables supplied to a tree. We also specified the learning task and corresponding learning objective with the $binary : logistic$ which means logistic regression for binary classification and output of probabilities.

The SVM was another algorithm we experienced (Subsection 2.2.4) with the e1071 package [46]. We use the *cost* parameter that is the soft margin cost function, in other words, how much we penalise the SVM for data points within the margin. We also use the *gamma* parameter that corresponds to Radial Base Function (RBF) function (cf. Equation 2.6). The *gamma* is used as a similarity measure between two points.

The GLM (Subsection 2.2.3) was trained using the stats package [49] and we specified the statistical model of logistic regression with the *logit* link function. Only was used the *epsilon* parameter which refers to the tolerance of positive convergence of iterations.

Regarding KNN algorithm (Subsection 2.2.5), this one has tested with the caret package [31] and the k parameter that represents the number of instances (neighbours) to include in the majority of the voting process for determination of the class. The data we work on has some features with a large scale compared to others. This is the case with variables *sum_upload* and *median_up_hour*, for example, which are in the order of thousands while most of the rest are in the order of units or tens. Therefore, we scaled the numerical data with the *build_scales* function from dataPreparation [62] package. This function calculates the standard deviation and divides each example by that standard deviation.

Lastly, an ANN (Subsection 2.2.6) was trained with the nnet package [64]. We used the *size*, *decay* and *maxit* parameters. This package fits a single hidden layer and the *size* is the number of units in this hidden layer. The *decay* is the weight penalty, in other words, is the regularisation to avoid over-fitting, and the *maxit* represents the number of iterations. When forming a neural network the data normalisation is very important so we use the *preProcess* function of the caret package with the method *range* that normalise values into the range between 0 and 1.

Table 4.1 summarises the learning algorithms, the packages and the set of parameters tested for each algorithm. Over this set of parameters we have performed a grid search. That is, we went sequentially through all the possible combinations of parameters, creating a model with them and testing in the validation set. After running all the combinations, we selected the best parameterisation and applied it to the test set.

Learning Model	**Name**	**Package**	**Parameters**
CART	CART	rpart [60]	$minsplit \in \{5, 10, 20, 30, 50\}$ $cp \in \{0.01, 0.05, 0.1\}$
Random Forest	RF	ranger [67]	$mtry \in \{1, 3, 5, 10\}$ $num.trees \in \{50, 100, 200, 400, 600\}$ $max.depth \in \{1, 5, 10\}$
XGBoost	XGBoost	xgboost [17]	$eta \in \{0.01, 0.05, 0.1\}$ $max\_depth \in \{5, 10, 15\}$ $nroud \in \{25, 100, 250, 500\}$ $colsample\_bytree \in \{0.3 \leq x \leq 0.9, by = 0.1\}$
SVM	SVM	e1071 [46]	$cost \in \{0.01, 0.1, 1, 3, 5, 10, 100\}$ $gamma \in \{0.01, 0.1, 1, 10, 100\}$
GLM	GLM	stats [49]	$epsilon \in \{10^i : 0 \leq i \leq 10\}$
KNN	KNN	caret [31]	$k \in \{3, 5, 7, 8, 10, 15, 20, 25\}$
ANN	ANN	nnet [64]	$size \in \{1, 2, 3\}$ $decay \in \{0.01, 0.1, 1, 10\}$ $maxit \in \{100, 200, 300, 500\}$

Table 4.1: Set of parameters for the models used during train

4.1.2 Strategies for imbalanced learning

Datasets where the classes are not evenly distributed, may be a challenge for the classification tasks. They are composed by a majority (negative) class and a minority (positive) class and, thus, the class distribution is skewed. Due to the characteristics of the dataset, and as previously explained in Section 3.3, learning from such data requires different approaches and evaluation criteria.

The approaches used to solve this challenge were the pre and post-processing (Section 2.3). Concerning pre-processing, we apply the techniques of Random Under-Sampling (RUS), Random Over-Sampling (ROS) and Synthetic Minority Over-sampling Technique (SMOTE) using the UBL [7] package. Regarding post-processing, we implement the threshold method.

Pre-processing

Through RUS, it is possible to achieve class balance by under-sampling the majority class, randomly selecting instances of the majority class from a dataset to be easy to learn characteristics about the minority class. In the case of ROS, the instances corresponding to the minority class are replicated. SMOTE uses a nearest neighbours algorithm to generate synthetic data. Is an over-sampling method which generates new artificial examples for the minority classes. We combined SMOTE with RUS as studied in the original work of Chawla et al. [15]. For SMOTE, we also need to define the k parameter, indicating the number of nearest neighbours used to generate new synthetic examples for the minority class.

The set of parameters used for each one of these techniques are present in the Table 4.2.

Technique	% under-sampling	% over-sampling	k
RUS	10,25,50,75,90	-	-
ROS	-	150,200,300,500	-
SMOTE+RUS	10,25,50,75,90	150,200,300,500	1,3,5

Table 4.2: Set of parameters for the pre-processing techniques

Post-processing

One of the main post-processing techniques is the threshold method. The classifiers provide a score that expresses the degree (probability) to which an example is a member of a class. This method uses the score to get a ranking of the examples and produces several learners by varying the thresholds for the classes.

To have more precise probabilities we calibrated probability thresholds on a validation dataset and determined the optimal threshold. The training dataset was splitted into a training and validation dataset with 70% and 30% of data, respectively. To maximise the score we tested some different thresholds between 0 and 1 by 0.01 and the best threshold was chose to be applied on the test set.

4.1.3 Evaluation metrics

There are several metrics and all offer different insights. As previously explained, some evaluation metrics typically used for classification problems, such as *Accuracy*, are not suitable for imbalanced datasets. As so, some metrics were taken into consideration such as *precision*, *recall*, *Fscore* with $\beta = 1$ and Area Under the ROC Curve (AUC) as a complementary metric. The positive class represents the contaminant Technical Orders (TOs) and the negative class represents the non-contaminant TOs.

4.1.4 Experimental methodology

Given the temporal character of our data the approaches used in the experimental methodology were growing and sliding window. Given a training set, the growing window adds the recent data to the current training set, this constantly increasing the size of this set. Taking into account our dataset, Figure 4.1 shows all the tests performed with the growing window. Based on the intuition of the domain we chose four weeks to the training set.

Assuming that recent data is more helpful in producing better models and the oldest part of training set is outdated, this can lead to a decreasing performance for the models. To overcome this problem, the sliding window removes the oldest data of the training set and adds the recent data maintaining a training set of constant size. Figure 4.2 is the application of sliding window for our dataset.

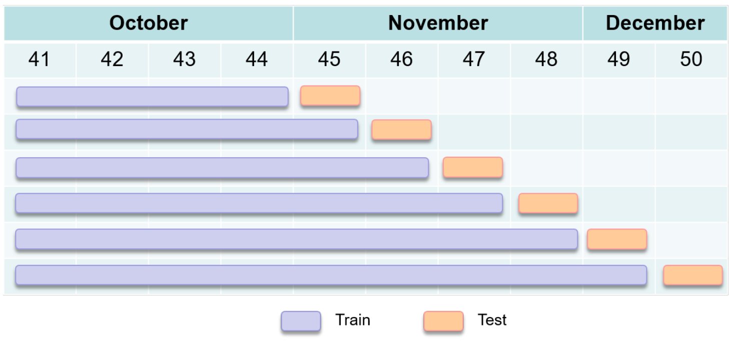


Figure 4.1: Growing window

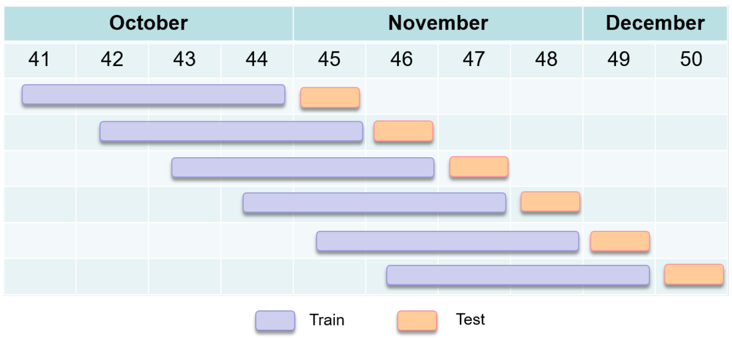


Figure 4.2: Sliding window

In both methodologies we get 6 pairs of training and testing set. For each grid parameter, we proceeded as the Figure 4.3 suggests.

i. For finding the best threshold, we split the training set into train and validation sets. Then, we apply re-sampling strategies to the later train set and build a classification model with this balanced training set. With the validation set, we get a set of predictions as probabilities. In order to achieve the best threshold, we calculate the $Fscore$ for each threshold. The selected threshold is the one which yields the best overall $Fscore$;

ii. After we define the best threshold, we re-sampled the original training set and build a new classification model on this data set. Then, the set of predictions was obtained with the testing set;

iii. The AUC was calculated with the set of probabilities;

iv. In order to calculate the $Fscore$, we first transform the set of predictions into a set of binary predictions with and without threshold;

v. At the end of 6 tests, an average is calculated to provide the final result.

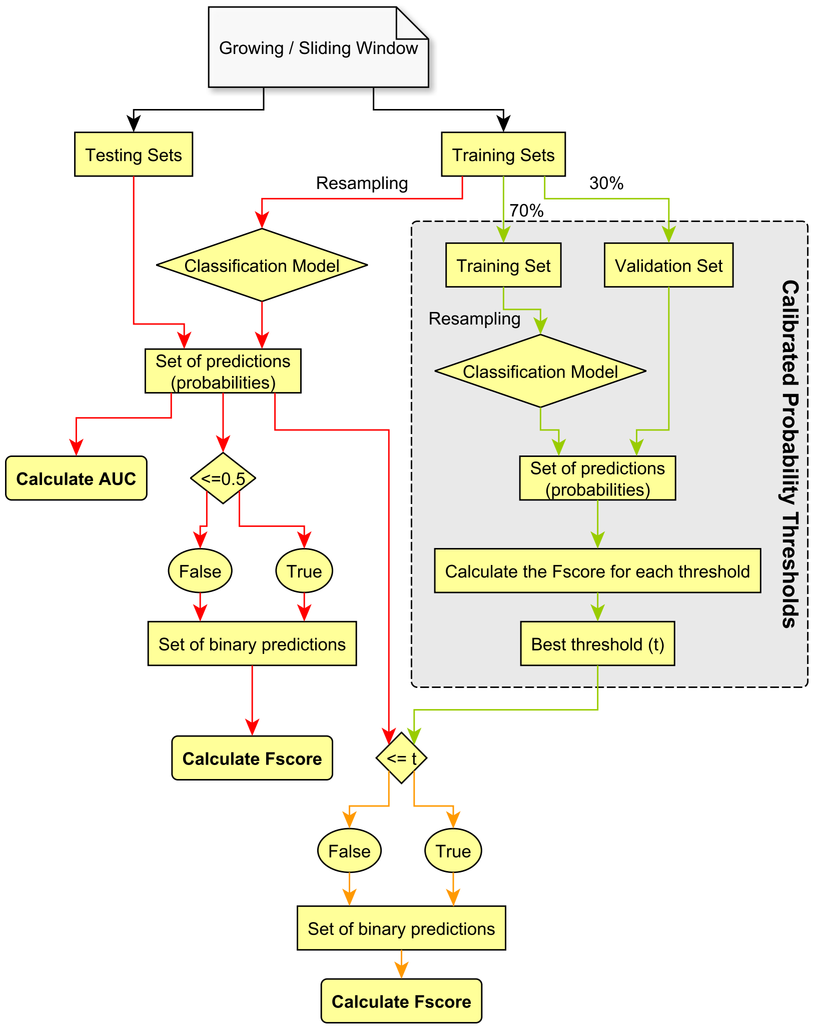


Figure 4.3: Flowchart of the experiment

4.2 Results

We start presenting the baseline results, with and without new features. And then, we use the pre-processing techniques to balance the dataset and compare the outcome with the post-processing technique. All these steps use growing and sliding approaches.

4.2.1 Baseline

We start by building prediction models that used the initial data set, composed by 21 numeric features, to have a benchmark to work with. The seven algorithms described above were used with the grid search present in Table 4.1. The prediction probability threshold in this implementation was set at 0.5. This means that if the prediction probability exceeds 0.5, the sample is predicted as contaminant TO (positive class), or else negative.

The first step in improving baseline results is to do some feature engineering, as it is important to understand signal oscillations before and after a TO. As we described in Section 3.4, we created some insights regarding the network signals values, router status, number of neighbours and number of Technical Participations (TPs). Since there are 7 different signals and we calculate the maximum, mean and the variance for each one, we end up with too many features in the final dataset. In total, we add to the initial dataset 145 features.

The best *Fscore* results obtained without feature engineering and adding new variables are represented in Figure 4.4. All the remaining results are in Appendix B.1, Table B.1 and Table B.3 for growing and sliding window without feature engineering, respectively. And, Table B.5 and Table B.7 with feature engineering.

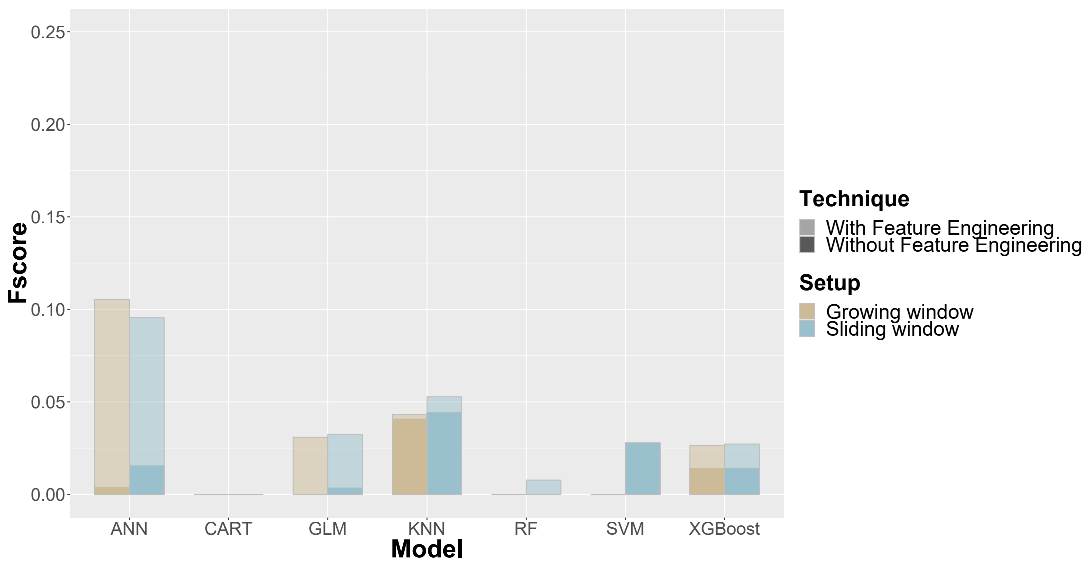


Figure 4.4: Baseline results

For both cases of growing and sliding window, the results without new features are, as expected, not very good - the models present a *Fscore* close to zero. Still, the sliding window revealed slightly better performance overall, when compared to the growing window. The results obtained with the new features are overall better, maintaining the patterns that were observed in the results without those features. The ANN obtained the best result and the one which increases the *Fscore* considerably, according to the growing window methodology. Finally, we should note that without balancing techniques applied, almost no positive cases were correctly predicted.

The parameters for the models associated with the results without the inclusion of new features can be consulted in Appendix B.1, Table B.2 and Table B.4. In the feature engineering case, the parameters are the same for both growing and sliding window (cf. Appendix B.1, Table B.6)

4.2.2 Pre-processing

The next step is to apply pre-processing techniques to balance the dataset. The best results for the three techniques using the growing and sliding window approaches are present in Figure 4.5.

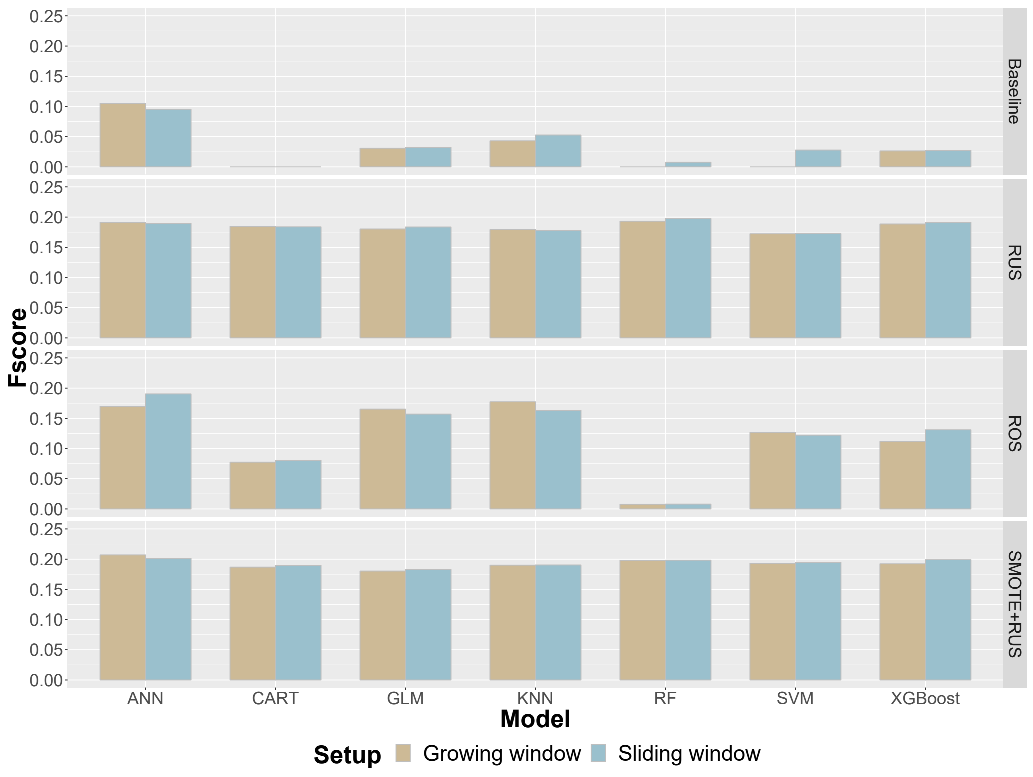


Figure 4.5: Pre-processing results compared to baseline

We started with RUS that balances the class distribution reducing the number of TOs that are not contaminating. With this balance, all the models improve their results a lot. The CART, Random Forest and SVM were the models in which the improvement is more noticeable since the previously result was zero. The SVM model was the only one that used the 25% under-sampling (cf. Appendix B.2, Table B.9). All the others used 10%, which means that the number of cases with the negative class was very low, almost the same of cases of the positive class.

ROS balances the domain by randomly introducing replicas of contaminating TOs to the training dataset. In general, the results are not as good as RUS, this is because, in some cases, duplicating samples does not help some models as it does not produce new useful data. In this case, we increased the number of cases of the positive class five times (parameters are present in Appendix B.2, Table B.11), which may have been the cause for overfitting. Based on the obtained results, we can state that this situation happens mainly for tree-based algorithms. Figure 4.6 shows the Random Forest behaviour during training and testing set. There are 250 iterations since we use a grid search where we have 4 examples of *mtry*, 5 *num.trees*, 3 *max.depth* and 4 percentages of over-sampling. The model fits very well with the previously observed data but it is ineffective in predicting new cases.

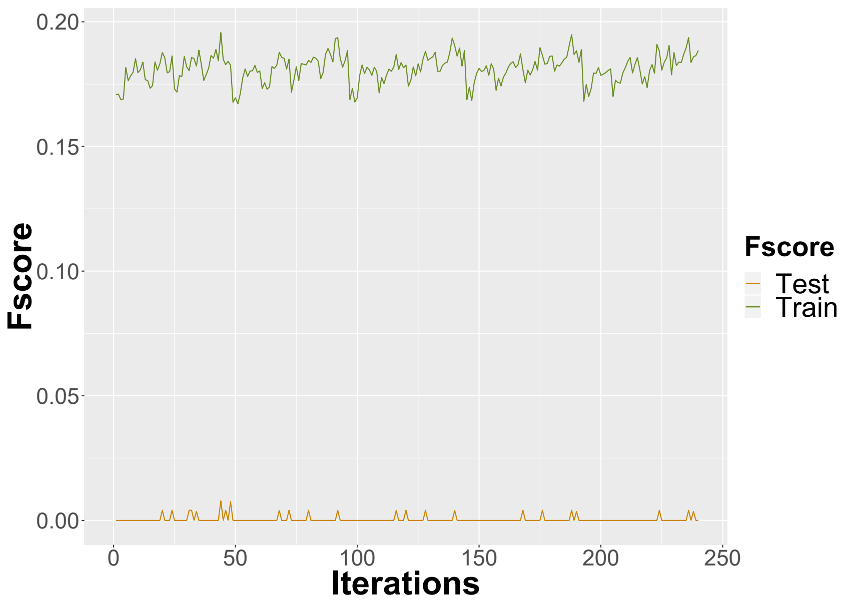


Figure 4.6: Random Forest overfitting on Random Over-Sampling (ROS) from Figure 4.5

We also test the combination of RUS and SMOTE. With this technique, all algorithms slightly improved their performance, except GLM which kept the same result as RUS. Comparing the three techniques, the SMOTE is the one that has better results (parameters in Appendix B.2 Table B.13).

Overall, with pre-processing techniques, the *Fscore* values of each model are similar using both growing window and sliding window. That is, no considerably different values were obtained for these methodologies. Regarding to the growing window results, these can be consulted in Appendix B.2, Tables B.8, B.10 and B.12, with respective parameters in Tables B.9, B.11 and B.13. The results concerning sliding window can be consulted in detail in Appendix B.2, Tables B.14, B.16 and B.18, with respective parameters in Tables B.15, B.17 and B.19.

4.2.3 Threshold calibration

The default threshold does not work well for imbalanced classification prediction. Therefore, we use the threshold adjustment as post-processing strategy. To find the best threshold we resort to the greedy strategy. The parameters used were the same as in Table B.6.

Figure 4.7 displays the obtained results (detailed in Appendix B.3, Table B.20 and Table B.21). As expected, the overall results improve considerably. The low threshold values (cf. Table 4.3) mean that the model is predicting almost every example as positive since the dataset is extremely imbalanced. This type of extreme model bias renders the results as non interesting, due to low interpretability. The growing window, in this case, presents results equal or greater than the sliding window.

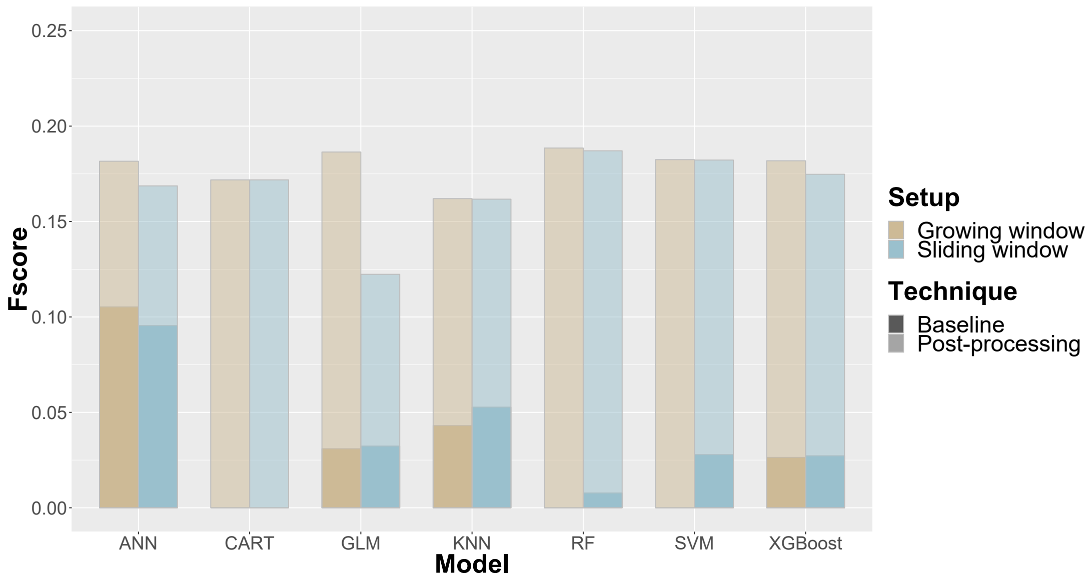


Figure 4.7: Threshold method results compared to baseline

Model	ANN	CART	GLM	KNN	RF	SVM	XGBoost
Thr growing	0.09	0.088	0.095	0.315	0.085	0.086	0.405
Thr sliding	0.055	0.088	0.385	0.315	0.078	0.206	0.403

Table 4.3: Threshold values of post-processing models from Figure 4.7

4.2.4 Combination of pre and post-processing techniques

The latter approach of modelling used in our approach was the combination of both pre and post-processing techniques. After balancing the dataset, we adjusted probability thresholds and then we compare the performance of each model (cf. Figure 4.8).

The ANN was the unique model that improved the results with threshold adjustment over RUS. The KNN and GLM are the ones that show no improvement with ROS, all the other models improved their performances followed by threshold calibration. This big difference from the combination of ROS and threshold adjustment is derived from the overfitting which is solved with this probability calibration. The KNN is the only one that shows a slight improvement over SMOTE. The results with growing window are presented in more detail in Appendix B.4, Table B.22 for RUS with threshold adjustment, Table B.23 for ROS and Table B.24 for SMOTE and the results with sliding window in Tables B.25, B.26 and B.27.

With the threshold adjustment over the pre-processing techniques, there are more variability of the *Fscore* values between the growing and sliding window. In general, the results of the pre and post-processing combination are worse than pre-processing. Overall threshold adjustment improved ROS results but not RUS or SMOTE results.

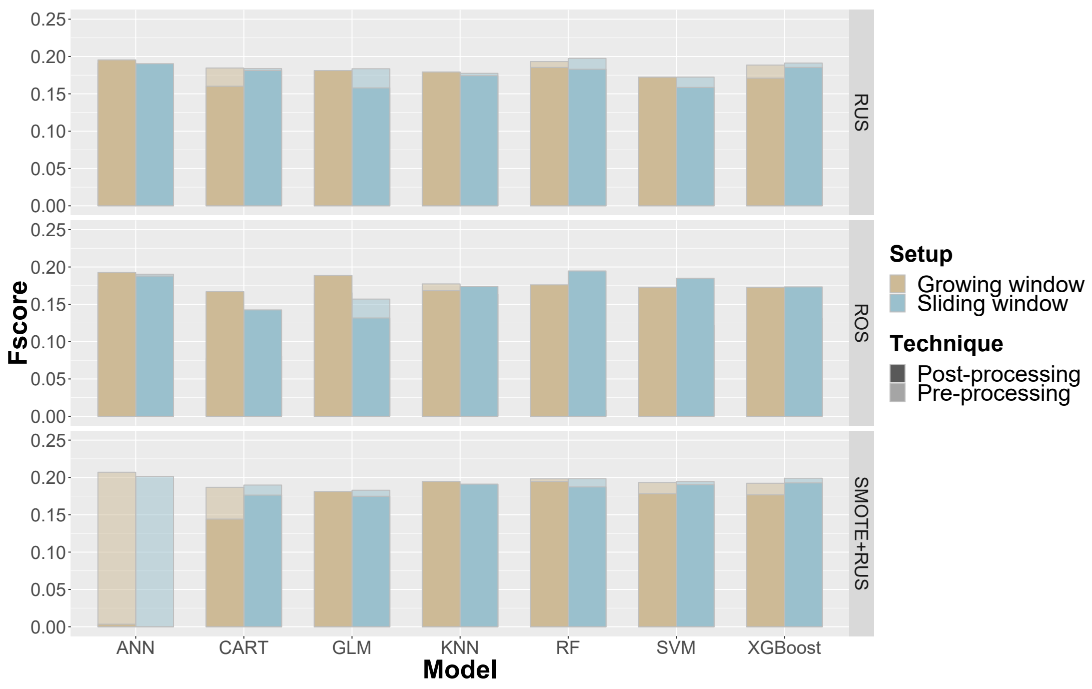


Figure 4.8: Pre and post-processing combination results compared to pre-processing

4.2.5 Feature Importance

At this point, we would like to understand if all the variables used are equally important to the model. Therefore, it is crucial to rank them by their contribution to the results. Removing some of the variables that do not make a big contribution to the solution can be a huge help. Since the best results were obtained using the SMOTE pre-processing technique with growing window methodology and the Random Forest model was one that achieved the best results, we applied the same approaches with the most important features in order to compare with the previous results.

The dataset was initially composed with a total of 165 variables including the target variable. For each of the six iterations, an analysis of the most important variables was performed with the ranger package using the *importance* parameter and, after that, we created a rank based on the place that each variable took in each iteration. Figure 4.9 represents the average of the rank that each variable had for the six iterations.

The six most informative variables are related to the router status. When the router is turned off, there are no network signal registers. This leads to a high probability of an intervention as it may indicate network failures in the building. The reboots often happen when the customer tries to solve the problem before resorting to TO, however often these attempts are in vain. The number of neighbours for each customer and the number of TPs are also important variables for the model (the description of each variable is present in Appendix A, Table A.1).

We later ran the Random Forest model using the SMOTE pre-processing technique and growing window with different number of features according to the rank of variables. The goal is

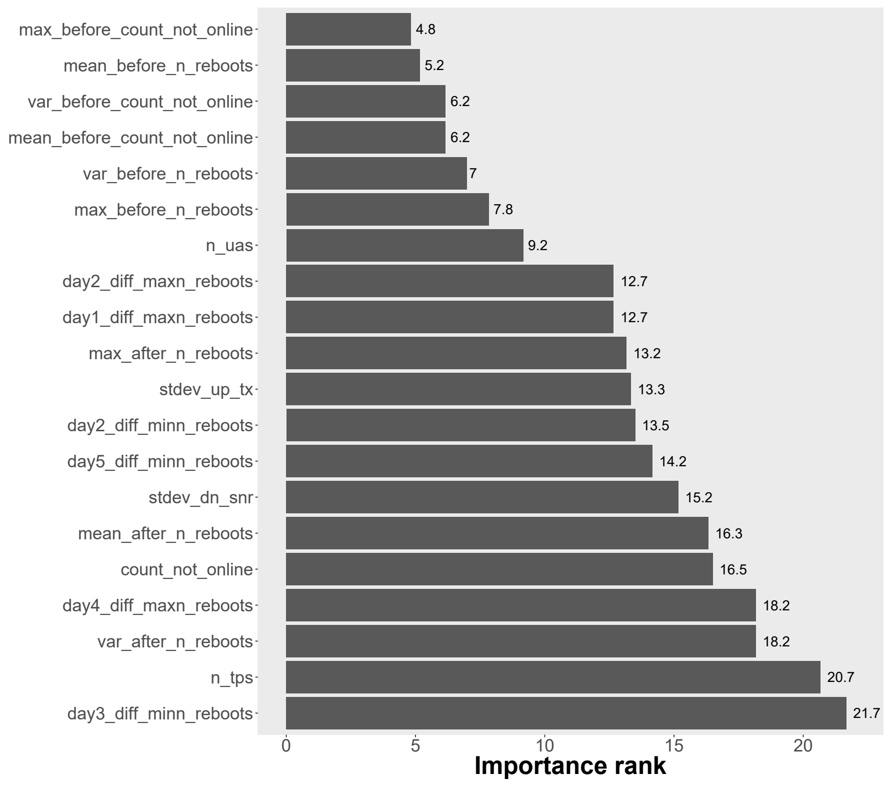


Figure 4.9: Rank for the 20 most important variables with Synthetic Minority Over-sampling Technique (SMOTE) using Random Forest and growing window methodology

to apply the best set of features to the remaining models. First with the top 20, then with 30, 50, 65, 80 and 100 variables. Figure 4.10 shows the performance of Random Forest for each one of these sizes (detailed results are in Appendix B.6, Table B.28 with the respective parameters in Table B.29).

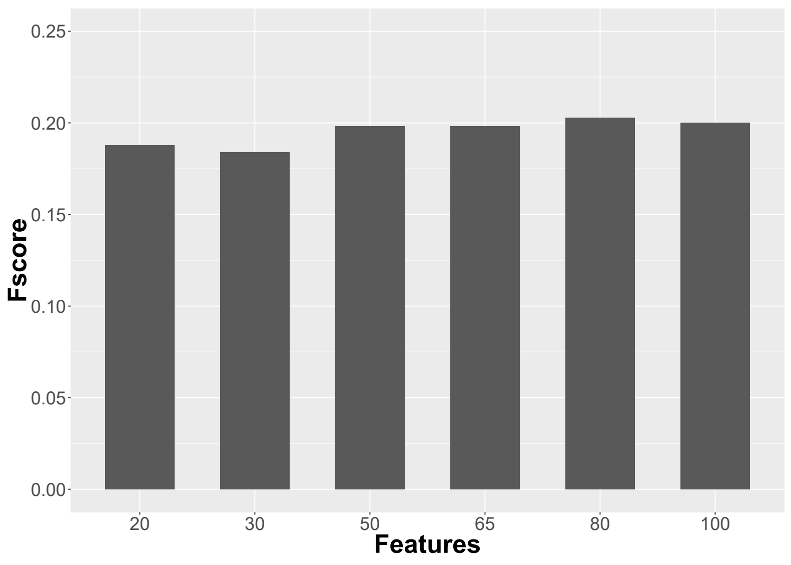


Figure 4.10: Performance of Random Forest using Synthetic Minority Over-sampling Technique (SMOTE) and growing window with different dataset sizes

The *Fscore* values do not gradually increase with number of attributes. The dataset with 80 most important variables is the one with the best performance, so we use this features to run the remaining models.

Finally, applying the SMOTE combined with RUS pre-processing technique, we compared the previous results of this technique with the result obtained through the new reduced dataset. Figure 4.11 shows the results of each model with the 80 most important variables (detailed in Appendix B.6 Table B.30 with the respective parameters in Table B.31).

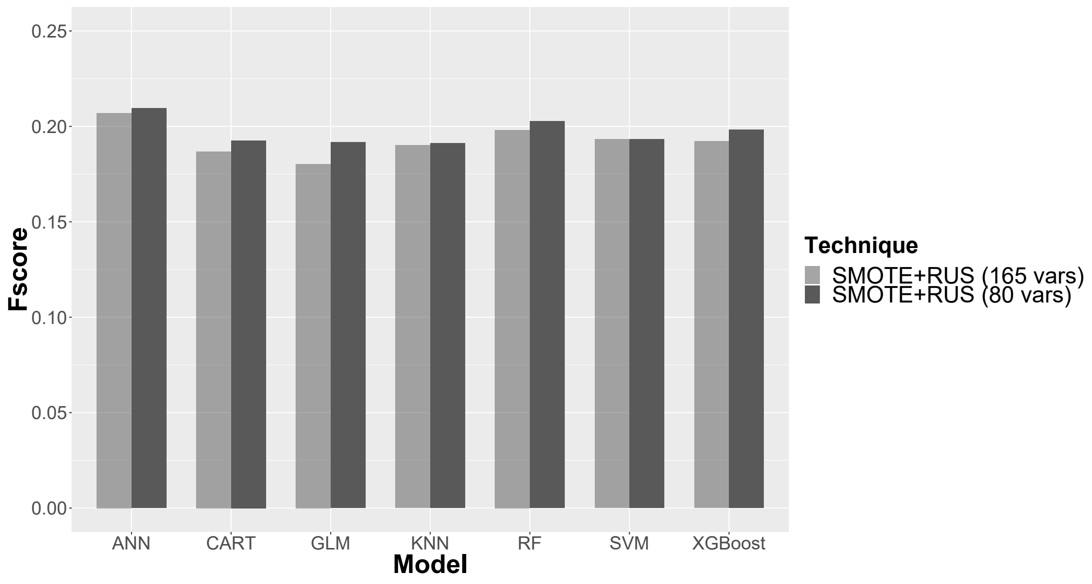


Figure 4.11: Results of Random Forest with Synthetic Minority Over-sampling Technique (SMOTE) using the 80 most important features and growing window methodology compared to all features

Overall, with this selection of variables, all models have improved their performance. GLM it is the one that has the biggest result difference, but nevertheless, ANN is the one that still has the best results.

4.2.6 Discussion

When the default threshold is set at 0.5, the performance of the models is very poor in terms of *precision* and *recall* and thus, low *Fscore* value. With the introduction of new features, we were able to improve the performance of some models, but still, it was not a considerable improvement, the *Fscore* values remained low. Due to dataset imbalance, we resorted to pre and post-processing techniques.

With RUS all the models improved their performances whereas with ROS, Random Forest and CART have very low *recall* and *Fscore* values and some tests have failed to predict any positive examples. The under-sampling discards potentially useful data, and on the other hand, over-sampling make exact copies of existing examples, producing overfitting. To bridge the flaws of these techniques we use the SMOTE combined with RUS. In fact, with this combination, the results were marginally better than RUS and we achieved considerable improvements than ROS.

The different thresholds yielded different *Fscore* values, so we used an adequate threshold that was set in the testing set. It is normal that the threshold of every model is lower than 0.5 since the dataset is extremely imbalanced, the probability distribution will be dragged to a lower value and, consequently, the threshold for the positive values will also drop. The threshold adjustment showed to be effective without pre-processing techniques, but when we combined both, the results does not improve for the three applied techniques. The ROS was the only one that had considerably improvements. Before, this one had very low *Fscore* values, which are a consequence of overfitting however with the threshold adjustment we get better values of *Fscore*.

Regarding the different experimental methodologies, growing window and sliding window, they show differences that in some cases are more notable. However, there are some results where the sliding window approach is no better than the growing window contrary to what was expected. The SMOTE pre-processing technique yield the best results from all applied approaches for both growing and sliding window.

After comparing the models between them and concluding which technique is best, one of the proposed objectives is also to understand what makes a technical intervention contaminating. By analysing what variables of the model influence the most the final outcome, it becomes clear what the problems are. The variables with the highest significant rank are related to the router status. Using the most important variables helped to improve previous results on the SMOTE pre-processing technique. *Fscore* values have increased for almost all models.

After all these attempts to improve the results, we can see that the best result was obtained with the dataset of 80 variables through the ANN model with the SMOTE pre-processing technique and the growing window approach. Table 4.4 shows the best results from different techniques using the growing window methodology. Values marked in bold correspond to the best model and italics the best technique.

Technique / **Model**	**Baseline**	**Pre-processing**			**Post-processing**	**Pre and Post**			**80 most important features with SMOTE**
		RUS	**ROS**	**SMOTE**		**RUS**	**ROS**	**SMOTE**	
ANN	0.105	0.191	0.169	**0.206**	0.181	**0.195**	**0.192**	0.003	***0.209***
CART	0	0.184	0.077	0.186	0.171	0.160	0.166	0.181	*0.192*
GLM	0.030	0.180	0.165	0.180	**0.186**	0.181	0.188	0.181	*0.191*
KNN	**0.043**	0.183	**0.177**	0.190	0.162	0.179	0.168	***0.194***	0.191
RF	0	**0.193**	0.007	0.198	0.171	0.185	0.187	**0.194**	*0.202*
SVM	0	0.172	0.126	*0.193*	0.182	0.172	0.172	0.178	*0.193*
XGBoost	0.026	0.188	0.111	0.192	0.181	0.171	0.172	0.176	*0.198*

Table 4.4: Best *Fscore* results of each model for all applied techniques with growing window methodology

Figure 4.12 shows the *Fscore*, *precision* and *recall* values for each week of the above mentioned best model. The *Fscore* and *precision* values does not present much variability over the 6 test weeks. Begins to rise slightly but drops to week 49 rises again at week 50. However, the *recall* it rises until week 48, has the worst peak at week 49 and rises again at week 50.

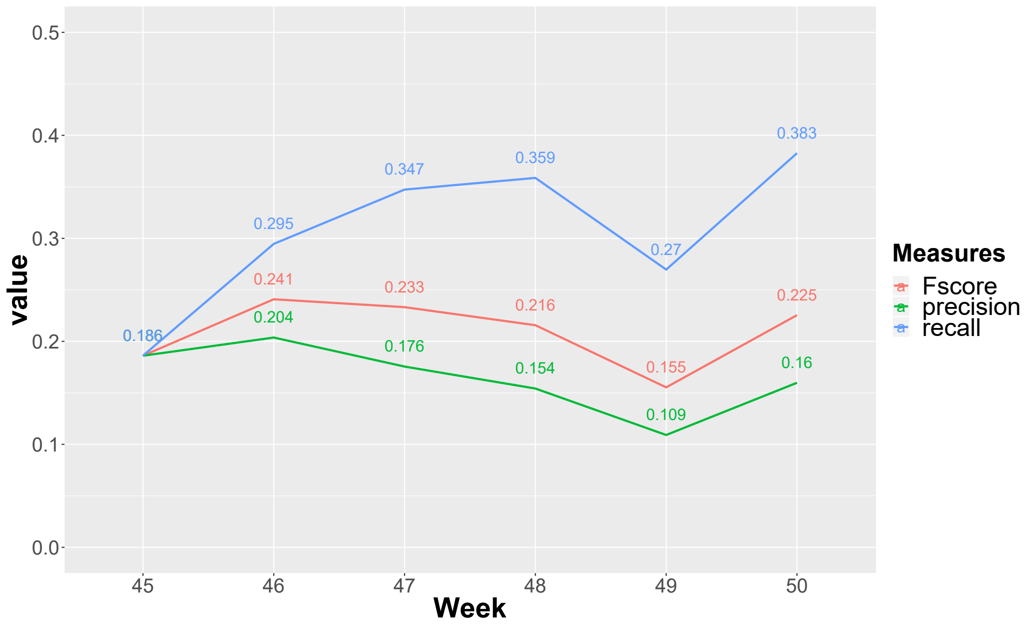


Figure 4.12: Evaluation metrics behaviour for each test week for Artificial Neural Network (ANN) model with Synthetic Minority Over-sampling Technique (SMOTE) and using growing window methodology

Chapter 5

Conclusions

The goal of this dissertation was to enable predictive maintenance area with focus on the identification of contaminating Technical Orders (TOs), i.e. TOs that could affect the service of other costumers in the same building. After the development of this project, it was possible to answer this question. The partner company has problems in the way the technicians deal with maintenance, and with the model developed they will have the opportunity to improve the customer experience.

This work focuses on technical interventions made at distribution points. Since interventions affecting neighbouring clients are a minority, we use pre and post-processing strategies on the data. This constitutes one of the main contributions of this study.

The final *Fscore* increases 2 times compared to the baseline results. The models created are useful to the partner company identify which TOs will cause more problems. However, it also outputs several False Positive cases that can not be seen as a complete misclassification, since some of them are indeed, contaminant interventions. The importance of feature engineering for a predictive modelling problem like this one should also be highlighted. In the top 80 most relevant variables for the model, 66 of them were a product of feature engineering.

During the project implementation, we faced several obstacles that had to be overcome. Working with a large amount of data was very challenging, especially with the limited available resources. The dataset had to be split into sub-folders and then rejoined to add to the data warehouse. Due to the size of the dataset, it was not possible to manipulate it locally on the computer and so PySpark was used. In addition to these challenges, we still deal with changes in the target variable definition as we were the first in the partner company to work on this subject. Thus, some definitions were sharpened during the development process.

In short, the project objectives were successfully met, the various challenges presented were overcome, and with some refinements in our model, the partner company can apply it reactively in order to deploy preventive maintenance actions that should be able to reduce the technical interventions and, consequently, improve customer satisfaction.

5.1 Future Work

Upon completion of the project, it was shown that the model has the potential to predict which TOs are contaminants. However, we can improve the results and therefore present some possibilities for future work.

In the beginning, we accessed data collected from Cable Modem (CM) that had hourly events but did not have all the information necessary for the completion of the project. More granular data allows better detection of signal differences after the occurrence of a TO. For this reason, we believe that hourly data rather than daily records, would improve the results.

The analysis of some of the time series could have been done to enrich the feature engineering. The studies on trend, seasonal variations, periodicity and stationarity of data would give better insight into the data.

As we saw earlier in the growing and sliding window techniques, we chose the size of the four week window for the training set given the domain intuition. However, the best approach for using these techniques is to tune the size of the training window because we do not know which size presents the best performing models.

In *Fscore* calculation we gave equal importance to *precision* and *recall* with the β parameter equal to 1. But, as we have already mentioned, we should not make plans if not necessary, focusing only on interventions that are contaminating. In this sense it is important to give more importance to *precision* than *recall*, and for that, we can set β to 0.5, doubling its importance.

Appendix A

Features description

Tables	**Features**	**Description**	**Used**	**Information**
Client Vision	pd	Distribution Point		Network topology
	mac	MAC Address of CM		
	model	CM model		
	cmts_name	CMTS		
	cell	Cell		
	sector	Sector		
	subsector	Subsector		
	ua	Unique Address		
	buildingid	Building address		
	sa	Service Account		Service Account
	count_above_up_tx	number of hourly events > thr_1 dBmV of the transmitted signal power in the upstream carrier	✓	Signals
	count_below_up_tx	number of hourly events < thr_2 dBmV of the transmitted signal power in the upstream carrier	✓	
	count_above_dn_rx	number of hourly events > thr_3 of the received signal power in the downstream carrier	✓	
	count_below_dn_rx	number of hourly events < thr_4 of the received signal power in the downstream carrier	✓	
	count_above_cer_dn	number of hourly events (ratio of downstream errors received by the CM) > thr_5	✓	
	count_below_dn_snr	number of hourly events (SNR of downstream related to CM) < thr_6 dBmV	✓	
	count_below_up_snr	number of hourly events (SNR of upstream related to CM) < thr_7 dBmV	✓	
	stdev_up_tx	Standard deviation of the transmitted signal power in the upstream carrier	✓	
	stdev_dn_rx	Standard deviation of the received signal power in the downtream carrier	✓	
	stdev_up_rx	Standard deviation of the received signal power in the upstream carrier	✓	
	stdev_dn_snr	Standard deviation of the SNR in the downstream carrier	✓	
	stdev_up_snr	Standard deviation of the SNR in the upstream carrier	✓	
	sum_download	Sum of kpis in downstream carrier	✓	
	sum_upload	Sum of kpis in upstream carrier	✓	
	median_up_hour	Median of kpis in upstream carrier	✓	
	median_down_hour	Median of kpis in downstream carrier	✓	
	count_not_online	CM status is not online	✓	Router status

Tables	Features	Description	Used	Information
Client Vision	n_reboots	Number of daily reboots	✓	
	avg_reboot_duration	Average duration of reboots	✓	
	date_to	Date and time of TO		Dates
	close_date_to	Date and time of completion of TO		
	date_pt	Date and time of TP		
	close_date_tp	Date and time of completion of TP		
	day_part	Analysis day		
	n_tps	Number of daily PTs	✓	TPs
TO	symptom_area_to	Problem area		Problem type
	symptom_sub_area_to	Sub problem area		
	symptom_type_to	Problem type		
	to_code	TO code		
	type_to	Type of TO		
	sub_type_to	Sub type of TO		
	user_area	Customer support identification		Customer support
	equipment_model	Model of the equipment		Equipment
	equipment_type	Type of the equipment		
	to_situation	TO status		TO status
	to_delegation	Zone		TO zone
	to_created_date	Date and time of TO		TO date
Created variables	date_to1	Date format of TO		Formatted dates
	date_tp1	Date format of TP		
	day_part1	Date format of analysis day		
	target	Target variable (TO contaminant/non-contaminant)	✓	Target
	tps_before	Number of TPs before a TO	✓	TPs before
	mean_before_count_below_up_tx	Average of aggregation of the previous 5 days a TO for each signal	✓	Average before
	mean_before_count_above_up_tx		✓	
	mean_before_count_above_dn_rx		✓	
	mean_before_count_above_cer_dn		✓	
	mean_before_count_below_up_snr		✓	
	mean_before_count_not_online	Average of aggregation of the previous 5 days a TO for router status	✓	
	mean_before_n_reboots		✓	
	var_before_count_below_up_tx	Variance of aggregation of the previous 5 days a TO for each signal	✓	Variance before
	var_before_count_above_up_tx		✓	
	var_before_count_below_dn_rx		✓	
	var_before_count_above_dn_rx		✓	

Tables	Features	Description	Used	Information
	var_before_count_above_cer_dn		✓	
	var_before_count_below_dn_snr		✓	
	var_before_count_below_up_snr		✓	
	var_before_count_not_online	Variance of aggregation of the previous 5 days a TO for router status	✓	
	var_before_n_reboots		✓	
	max_before_count_below_up_tx		✓	
	max_before_count_above_up_tx		✓	
	max_before_count_below_dn_rx		✓	
	max_before_count_above_dn_rx	Maximum of aggregation of the previous 5 days a TO for each signal	✓	
	max_before_count_above_cer_dn		✓	Maximum before
	max_before_count_below_dn_snr		✓	
	max_before_count_below_up_snr		✓	
	max_before_count_not_online	Maximum of aggregation of the previous 5 days a TO for router status	✓	
	max_before_n_reboots		✓	
	mean_after_count_below_up_tx		✓	
	mean_after_count_above_up_tx		✓	
	mean_after_count_below_dn_rx	Average of each signal after a TO	✓	
	mean_after_count_above_dn_rx		✓	
Created variables	mean_after_count_below_dn_snr		✓	Average after
	mean_after_count_below_up_snr		✓	
	mean_after_count_not_online	Average of router status after a TO	✓	
	mean_after_n_reboots		✓	
	var_after_count_below_up_tx		✓	
	var_after_count_above_up_tx		✓	
	var_after_count_below_dn_rx		✓	
	var_after_count_above_dn_rx	Variance of each signal after a TO	✓	
	var_after_count_above_cer_dn		✓	Variance after
	var_after_count_below_dn_snr		✓	
	var_after_count_below_up_snr		✓	
	var_after_count_not_online	Variance of router status after a TO	✓	
	var_after_n_reboots		✓	
	max_after_count_below_up_tx		✓	
	max_after_count_above_up_tx		✓	
	max_after_count_below_dn_rx		✓	
	max_after_count_above_dn_rx	Maximum of each signal after a TO	✓	
	max_after_count_above_cer_dn		✓	Maximum after

Tables	Features	Description	Used	Information
Created variables	max_after_count_below_dn_snr		✓	
	max_after_count_below_up_snr		✓	
	max_after_count_not_online	Maximum of router status after a TO	✓	
	max_after_n_reboots		✓	
	day1_diff_mincount_below_up_tx	Minimum difference of the first day before a TO and the day after	✓	Differences of first day before
	day1_diff_mincount_above_up_tx		✓	
	day1_diff_mincount_below_dn_rx		✓	
	day1_diff_mincount_above_dn_rx		✓	
	day1_diff_mincount_above_cer_dn		✓	
	day1_diff_mincount_below_dn_snr		✓	
	day1_diff_mincount_below_up_snr		✓	
	day1_diff_mincount_not_online		✓	
	day1_diff_minn_reboots		✓	
	day1_diff_maxcount_below_up_tx	Maximum difference of the first day before a TO and the day after	✓	
	day1_diff_maxcount_above_up_tx		✓	
	day1_diff_maxcount_below_dn_rx		✓	
	day1_diff_maxcount_above_dn_rx		✓	
	day1_diff_maxcount_above_cer_dn		✓	
	day1_diff_maxcount_below_dn_snr		✓	
	day1_diff_maxcount_below_up_snr		✓	
	day1_diff_maxcount_not_online		✓	
	day1_diff_maxn_reboots		✓	
	day2_diff_mincount_below_up_tx	Minimum difference of the second day before a TO and the day after	✓	Differences of second day before
	day2_diff_mincount_above_up_tx		✓	
	day2_diff_mincount_below_dn_rx		✓	
	day2_diff_mincount_above_dn_rx		✓	
	day2_diff_mincount_above_cer_dn		✓	
	day2_diff_mincount_below_dn_snr		✓	
	day2_diff_mincount_below_up_snr		✓	
	day2_diff_mincount_not_online		✓	
	day2_diff_minn_reboots		✓	
	day2_diff_maxcount_below_up_tx	Maximum difference of the second day before a TO and the day after	✓	
	day2_diff_maxcount_above_up_tx		✓	
	day2_diff_maxcount_below_dn_rx		✓	

Tables	Features	Description	Used	Information
Created variables	day2_diff_maxcount_above_dn_rx	Maximum difference of the second day before a TO and the day after	✓	Differences of second day before
	day2_diff_maxcount_above_cer_dn		✓	
	day2_diff_maxcount_below_dn_snr		✓	
	day2_diff_maxcount_below_up_snr		✓	
	day2_diff_maxcount_not_online		✓	
	day2_diff_maxn_reboots		✓	
	day3_diff_mincount_below_up_tx	Minimum difference of the third day before a TO and the day after	✓	Differences of third day before
	day3_diff_mincount_above_up_tx		✓	
	day3_diff_mincount_below_dn_rx		✓	
	day3_diff_mincount_above_dn_rx		✓	
	day3_diff_mincount_above_cer_dn		✓	
	day3_diff_mincount_below_dn_snr		✓	
	day3_diff_mincount_below_up_snr		✓	
	day3_diff_mincount_not_online		✓	
	day3_diff_minn_reboots		✓	
	day3_diff_maxcount_below_up_tx	Maximum difference of the third day before a TO and the day after	✓	
	day3_diff_maxcount_above_up_tx		✓	
	day3_diff_maxcount_below_dn_rx		✓	
	day3_diff_maxcount_above_dn_rx		✓	
	day3_diff_maxcount_above_cer_dn		✓	
	day3_diff_maxcount_below_dn_snr		✓	
	day3_diff_maxcount_below_up_snr		✓	
	day3_diff_maxcount_not_online		✓	
	day3_diff_maxn_reboots		✓	
	day4_diff_mincount_below_up_tx	Minimum difference of the fourth day before a TO and the day after	✓	Differences of fourth day before
	day4_diff_mincount_above_up_tx		✓	
	day4_diff_mincount_below_dn_rx		✓	
	day4_diff_mincount_above_dn_rx		✓	
	day4_diff_mincount_above_cer_dn		✓	
	day4_diff_mincount_below_dn_snr		✓	
	day4_diff_mincount_below_up_snr		✓	
	day4_diff_mincount_not_online		✓	
	day4_diff_minn_reboots		✓	

Tables	Features	Description	Used	Information
	day4_diff_maxcount_below_up_tx		✓	
	day4_diff_maxcount_above_up_tx		✓	
	day4_diff_maxcount_below_dn_rx		✓	
	day4_diff_maxcount_above_dn_rx	Maximum difference of the fourth day before a TO and the day after	✓	
	day4_diff_maxcount_above_cer_dn		✓	Differences of fourth day before
	day4_diff_maxcount_below_dn_snr		✓	
	day4_diff_maxcount_below_up_snr		✓	
	day4_diff_maxcount_not_online		✓	
	day4_diff_maxn_reboots		✓	
	day5_diff_mincount_below_up_tx		✓	
	day5_diff_mincount_above_up_tx		✓	
	day5_diff_mincount_below_dn_rx		✓	
Created variables	day5_diff_mincount_above_dn_rx	Minimum difference of the fifth day before a TO and the day after	✓	
	day5_diff_mincount_above_cer_dn		✓	
	day5_diff_mincount_below_dn_snr		✓	
	day5_diff_mincount_below_up_snr		✓	
	day5_diff_mincount_not_online		✓	
	day5_diff_minn_reboots		✓	Differences of fifth day before
	day5_diff_maxcount_below_up_tx		✓	
	day5_diff_maxcount_above_up_tx		✓	
	day5_diff_maxcount_below_dn_rx		✓	
	day5_diff_maxcount_above_dn_rx		✓	
	day5_diff_maxcount_above_cer_dn	Maximum difference of the fifth day before a TO and the day after	✓	
	day5_diff_maxcount_below_dn_snr		✓	
	day5_diff_maxcount_below_up_snr		✓	
	day5_diff_maxcount_not_online		✓	
	day5_diff_maxn_reboots		✓	

Table A.1: Features description of dataset

Appendix B

Results

B.1 Baseline results

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	0	887	0	92	NaN	0	0.905	0.5	0
RF	0	887	0	92	NaN	0	0.905	**0.543**	0
XGBoost	0	882	4	91	0.137	0.007	0.902	0.505	0.014
SVM	0	887	0	92	NaN	0	0.905	0.487	0
GLM	0	886	0	92	NaN	0	0.905	0.513	0
KNN	2	865	21	90	0.098	0.025	0.885	0.507	**0.040**
ANN	0	885	1	92	NaN	0.001	0.904	0.515	0.003

Table B.1: Best results without feature engineering using growing window methodology

Model	**Parameters**
CART	$minsplit = 5$, $cp = 0.01$
RF	$mtry = 1$, $num.trees = 50$ $max.depth = 5$
XGBoost	$eta = 0.1$, $max\_depth = 5$ $nroud = 500$, $colsample\_bytree = 0.4$
SVM	$cost = 0.01$, $gamma = 0.01$
GLM	$epsilon = 1e^-10$
KNN	$k = 3$
ANN	$size = 3$, $decay = 0.01$ $maxit = 200$

Table B.2: Parameters that led to the best results presented in the Table B.1

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	0	887	0	92	NaN	0	0.905	0.5	0
RF	0	887	0	92	NaN	0	0.905	**0.537**	0
XGBoost	0	883	3	91	0.120	0.007	0.903	0.495	0.014
SVM	14	739	147	77	NaN	0.166	0.769	0.487	0.027
GLM	0	886	0	92	NaN	0.001	0.905	0.521	0.003
KNN	2	862	24	89	0.099	0.028	0.883	0.512	**0.044**
ANN	0	884	2	91	0.336	0.007	0.904	0.527	0.015

Table B.3: Best results without feature engineering using sliding window methodology

Model	**Parameters**
CART	$minsplit = 5$, $cp = 0.01$
RF	$mtry = 1$, $num.trees = 50$ $max.depth = 5$
XGBoost	$eta = 0.1$, $max\_depth = 15$ $nroud = 500$, $colsample\_bytree = 0.7$
SVM	$cost = 0.01$, $gamma = 0.01$
GLM	$epsilon = 1e^{-}10$
KNN	$k = 3$
ANN	$size = 3$, $decay = 0.01$ $maxit = 200$

Table B.4: Parameters that led to the best results presented in the Table B.3

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	0	888	0	92	NaN	0	0.905	0.5	0
RF	0	888	0	92	NaN	0	0.905	0.549	0
XGBoost	1	883	5	91	0.244	0.014	0.902	0.560	0.026
SVM	0	888	0	92	NaN	0	0.905	**0.567**	0
GLM	1	882	6	90	0.222	0.017	0.900	0.565	0.030
KNN	2	864	24	89	0.100	0.028	0.883	0.527	0.043
ANN	7	858	30	85	0.198	0.074	0.881	0.523	**0.105**

Table B.5: Best results with feature engineering using growing window methodology

Model	**Parameters**
CART	$minsplit = 5$, $cp = 0.01$
RF	$mtry = 1$, $num.trees = 50$ $max.depth = 1$
XGBoost	$eta = 0.01$, $max\_depth = 10$ $nroud = 25$, $colsample\_bytree = 0.9$
SVM	$cost = 0.01$, $gamma = 0.01$
GLM	$epsilon = 0.01$
KNN	$k = 3$
ANN	$size = 3$, $decay = 0.01$ $maxit = 500$

Table B.6: Parameters that led to the best results presented in the Table B.5

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	0	888	0	92	NaN	0	0.905	0.5	0
RF	0	888	0	92	NaN	0.003	0.906	**0.569**	0.007
XGBoost	1	882	6	91	0.157	0.014	0.900	**0.569**	0.027
SVM	14	740	148	77	NaN	0.166	0.769	0.548	0.027
GLM	1	878	10	90	0.128	0.018	0.896	0.555	0.032
KNN	3	864	24	89	0.115	0.034	0.884	0.526	0.052
ANN	6	843	45	85	0.134	0.074	0.866	0.517	**0.095**

Table B.7: Best results with feature engineering using sliding window methodology

B.2 Pre-processing results

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	39	594	294	52	0.118	0.426	0.666	0.548	0.184
RF	53	480	408	38	0.116	0.581	0.544	**0.574**	**0.193**
XGBoost	54	465	423	38	0.112	0.583	0.528	0.557	0.188
SVM	92	2	886	0	0.094	1	0.996	0.444	0.172
GLM	44	540	348	48	0.111	0.474	0.595	0.551	0.180
KNN	10	799	88	81	0.112	0.461	0.606	0.552	0.183
ANN	46	542	346	45	0.118	0.501	0.601	0.565	0.191

Table B.8: Best Random Under-Sampling (RUS) results using growing window methodology

Model	**Parameters**	**% under-sampling**
CART	$minsplit = 30$, $cp = 0.05$	10
RF	$mtry = 10$, $num.trees = 50$ $max.depth = 5$	10
XGBoost	$eta = 0.05$, $max\_depth = 15$ $nroud = 25$, $colsample\_bytree = 0.4$	10
SVM	$cost = 5$, $gamma = 1$	25
GLM	$epsilon = 1$	10
KNN	$k = 25$	10
ANN	$size = 1$, $decay = 0.1$ $maxit = 100$	10

Table B.9: Parameters that led to he best results presented in Table B.8

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	8	840	47	83	NaN	0.052	0.866	0.513	0.077
RF	0	887	1	92	NaN	0.004	0.905	**0.579**	0.007
XGBoost	10	832	56	82	0.134	0.108	0.859	0.540	0.111
SVM	10	830	58	82	0.148	0.112	0.856	0.516	0.126
GLM	15	813	75	77	0.167	0.163	0.844	0.564	0.165
KNN	34	624	264	57	0.118	0.377	0.671	0.534	**0.177**
ANN	22	748	140	69	0.134	0.424	0.789	0.560	0.169

Table B.10: Best Random Over-Sampling (ROS) results using growing window

Model	**Parameters**	**% over-sampling**
CART	$minsplit = 50$, $cp = 0.01$	500
RF	$mtry = 10$, $num.trees = 50$ $max.depth = 5$	500
XGBoost	$eta = 0.01$, $max\_depth = 5$ $nroud = 25$, $colsample\_bytree = 0.9$	500
SVM	$cost = 100$, $gamma = 0.01$	500
GLM	$epsilon = 0.01$	500
KNN	$k = 7$	500
ANN	$size = 1$, $decay = 0.01$ $maxit = 300$	500

Table B.11: Parameters that led to he best results presented in Table B.10

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	55	433	455	37	0.112	0.601	0.499	0.546	0.186
RF	57	462	426	35	0.118	0.615	0.532	**0.591**	0.198
XGBoost	60	413	475	32	0.112	0.654	0.483	0.559	0.192
SVM	47	532	356	44	0.120	0.520	0.589	0.567	0.193
GLM	44	540	348	48	0.111	0.474	0.595	0.551	0.180
KNN	51	499	389	41	0.115	0.550	0.560	0.563	0.190
ANN	29	722	165	62	0.155	0.322	0.767	0.578	**0.206**

Table B.12: Best Synthetic Minority Over-sampling Technique (SMOTE) results using growing window methodology

Model	**Parameters**	**% under-sampling**	**% over-sampling**	**k**
CART	$minsplit = 5$, $cp = 0.05$	25	300	5
RF	$mtry = 5$, $num.trees = 400$ $max.depth = 5$	25	300	3
XGBoost	$eta = 0.1$, $max\_depth = 5$ $nroud = 100$, $colsample\_bytree = 0.5$	10	150	5
SVM	$cost = 0.01$, $gamma = 0.01$	25	300	5
GLM	$epsilon = 1$	10	150	1
KNN	$k = 25$	50	500	3
ANN	$size = 2$, $decay = 1$ $maxit = 100$	25	200	1

Table B.13: Parameters that led to he best results presented in Table B.12

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	52	458	430	39	0.109	0.570	0.518	0.538	0.183
RF	54	482	406	37	0.118	0.592	0.547	**0.577**	**0.197**
XGBoost	51	494	394	40	0.115	0.560	0.554	0.566	0.191
SVM	92	3	885	0	0.0.94	1	0.097	0.458	0.172
GLM	42	558	330	49	0.116	0.465	0.611	0.541	0.183
KNN	42	545	342	49	0.110	0.461	0.599	0.549	0.177
ANN	43	571	317	49	0.119	0.465	0.627	0.574	0.189

Table B.14: Best Random Under-Sampling (RUS) results using sliding window methodology

Model	**Parameters**	**% under-sampling**
CART	$minsplit = 30$, $cp = 0.01$	10
RF	$mtry = 10$, $num.trees = 200$ $max.depth = 5$	10
XGBoost	$eta = 0.01$, $max\_depth = 5$ $nroud = 500$, $colsample\_bytree = 0.4$	10
SVM	$cost = 5$, $gamma = 1$	25
GLM	$epsilon = 0.01$	10
KNN	$k = 25$	10
ANN	$size = 3$, $decay = 1$ $maxit = 100$	10

Table B.15: Parameters that led to he best results presented in Table B.14

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**
CART	6	847	41	85	NaN	0.072	0.869	0.495	0.080
RF	0	888	0	92	NaN	0	0.906	**0.572**	0.077
XGBoost	9	850	38	83	0.194	0.100	0.875	0.560	0.130
SVM	20	710	71	178	0.136	0.233	0.744	0.541	0.122
GLM	15	799	89	76	0.150	0.167	0.829	0.561	0.157
KNN	32	619	269	60	0.106	0.349	0.663	0.537	0.163
ANN	27	722	166	65	0.141	0.294	0.763	0.569	**0.190**

Table B.16: Best Random Over-Sampling (ROS) results using sliding window methodology

Model	**Parameters**	**% over-sampling**
CART	$minsplit = 50$, $cp = 0.01$	500
RF	$mtry = 5$, $num.trees = 50$ $max.depth = 5$	500
XGBoost	$eta = 0.01$, $max\_depth = 5$ $nroud = 25$, $colsample\_bytree = 0.9$	500
SVM	$cost = 100$, $gamma = 0.01$	500
GLM	$epsilon = 1e - 4$	500
KNN	$k = 7$	500
ANN	$size = 1$, $decay = 0.01$ $maxit = 300$	500

Table B.17: Parameters that led to he best results presented in Table B.16

Model	TP	TN	FP	FN	Precision	Recall	Accuracy	AUC	Fscore
CART	48	521	367	44	0.116	0.519	0.581	0.561	0.189
RF	43	589	299	49	0.127	0.469	0.646	**0.591**	0.198
XGBoost	63	408	479	29	0.116	0.686	0.481	0.586	0.198
SVM	69	334	553	22	0.111	0.755	0.412	0.588	0.194
GLM	44	544	344	48	0.113	0.477	0.599	0.540	0.182
KNN	74	279	609	18	0.108	0.800	0.360	0.568	0.190
ANN	39	627	261	52	0.131	0.428	0.679	0.564	**0.201**

Table B.18: Best Synthetic Minority Over-sampling Technique (SMOTE) results using sliding window methodology

Model	Parameters	% under-sampling	% over-sampling	k
CART	$minsplit = 50$, $cp = 0.01$	25	500	3
RF	$mtry = 3$, $num.trees = 400$ $max.depth = 1$	50	500	5
XGBoost	$eta = 0.01$, $max\_depth = 10$ $nroud = 500$, $colsample\_bytree = 0.4$	10	150	3
SVM	$cost = 0.1$, $gamma = 0.1$	25	300	5
GLM	$epsilon = 1$	10	150	1
KNN	$k = 25$	10	500	5
ANN	$size = 1$, $decay = 0.01$ $maxit = 100$	75	500	5

Table B.19: Parameters that led to he best results presented in Table B.18

B.3 Post-processing results

Model	TP	TN	FP	FN	Precision	Recall	Accuracy	AUC	Fscore	Thr
CART	92	0	888	0	0.094	1	0.094	0.5	0.171	0.088
RF	91	6	882	0	0.093	0.992	0.099	0.549	0.171	0.085
XGBoost	65	321	566	26	0.109	0.700	0.390	0.560	0.181	0.405
SVM	57	418	469	35	0.111	0.608	0.480	**0.567**	0.182	0.086
GLM	49	492	396	43	0.124	0.534	0.545	0.565	**0.186**	0.095
KNN	25	690	198	66	0.114	0.279	0.729	0.527	0.162	0.315
ANN	38	580	308	54	0.125	0.420	0.628	0.523	0.181	0.09

Table B.20: Threshold adjustment results using growing window methodology

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**	**Thr**
CART	92	0	888	0	0.094	1	0.094	0.5	0.171	0.088
RF	60	392	496	31	0.110	0.648	0.468	**0.569**	**0.187**	0.078
XGBoost	70	247	641	21	0.107	0.761	0.316	**0.569**	0.174	0.403
SVM	69	276	612	23	0.106	0.744	0.349	0.548	0.182	0.206
GLM	32	632	256	60	0.130	0.321	0.683	0.555	0.122	0.385
KNN	25	689	199	66	0.114	0.278	0.728	0.526	0.161	0.315
ANN	48	454	434	43	0.106	0.518	0.517	0.517	0.168	0.055

Table B.21: Threshold adjustment results using sliding window methodology

B.4 Pre and post-processing combination results

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**	**Thr**
CART	32	657	231	59	NaN	0.350	0.699	0.548	0.160	0.568
RF	59	394	494	33	0.110	0.642	0.460	**0.574**	0.185	0.483
XGBoost	55	403	485	37	0.103	0.589	0.469	0.557	0.171	0.471
SVM	88	45	842	4	0.094	0.956	0.136	0.444	0.172	0.678
GLM	40	581	306	52	0.115	0.431	0.636	0.551	0.181	0.538
KNN	51	456	432	40	0.108	0.549	0.552	0.552	0.179	0.476
ANN	40	611	277	52	0.126	0.433	0.664	0.565	**0.195**	0.546

Table B.22: Results of Random Under-Sampling (RUS) and threshold adjustment combination using growing window methodology

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**	**Thr**
CART	78	139	749	13	0.098	0.848	0.220	0.513	0.166	0.223
RF	35	618	269	56	0.110	0.648	0.468	0.569	0.187	0.078
XGBoost	46	502	386	45	0.109	0.484	0.562	**0.579**	0.172	0.456
SVM	42	515	373	49	0.113	0.459	0.573	0.516	0.172	0.14
GLM	37	614	273	54	0.129	0.409	0.666	0.564	0.188	0.353
KNN	36	575	313	55	0.108	0.399	0.627	0.534	0.168	0.381
ANN	51	484	404	40	0.121	0.560	0.549	0.560	**0.192**	0.335

Table B.23: Results of Random Over-Sampling (ROS) and threshold adjustment combination using growing window methodology

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**	**Thr**
CART	59	336	551	33	NaN	0.639	0.406	0.546	0.144	0.481
RF	65	384	504	27	0.114	0.698	0.455	**0.591**	**0.194**	0.485
XGBoost	48	481	407	44	0.109	0.528	0.539	0.559	0.176	0.555
SVM	44	536	352	47	0.122	0.482	0.594	0.567	0.178	0.51
GLM	40	581	306	52	0.115	0.431	0.636	0.551	0.181	0.538
KNN	63	396	491	29	0.114	0.677	0.471	0.563	**0.194**	0.441
ANN	0	888	0	92	NaN	0.001	0.905	0.578	0.003	0.885

Table B.24: Results of Synthetic Minority Over-sampling Technique (SMOTE) and threshold adjustment combination using growing window methodology

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**	**Thr**
CART	55	424	463	36	0.112	0.585	0.490	0.538	0.181	0.435
RF	64	348	540	27	0.107	0.686	0.422	**0.577**	0.182	0.476
XGBoost	73	269	619	19	0.106	0.777	0.351	0.566	0.185	0.396
SVM	78	135	753	13	0.088	0.831	0.223	0.458	0.158	0.658
GLM	31	635	253	60	0.119	0.340	0.676	0.541	0.157	0.82
KNN	50	424	42	0	0.105	0.540	0.528	0.549	0.174	0.476
ANN	52	492	396	40	0.115	0.554	0.553	0.574	**0.193**	0.485

Table B.25: Results of Random Under-Sampling (RUS) and threshold adjustment combination using sliding window methodology

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**	**Thr**
CART	74	176	712	18	NaN	0.804	0.244	0.495	0.142	0.223
RF	61	395	493	31	0.110	0.648	0.468	**0.572**	**0.187**	0.318
XGBoost	56	398	490	35	0.109	0.484	0.562	0.560	0.172	0.441
SVM	58	390	498	34	0.113	0.629	0.461	0.541	0.184	0.178
GLM	39	573	315	52	0.141	0.404	0.633	0.561	0.131	0.488
KNN	41	541	347	50	0.108	0.452	0.597	0.537	0.173	0.346
ANN	45	538	350	46	0.118	0.489	0.599	0.569	**0.187**	0.341

Table B.26: Results of Random Over-Sampling (ROS) and threshold adjustment combination using sliding window methodology

Model	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**Accuracy**	**AUC**	**Fscore**	**Thr**
CART	54	416	472	37	0.106	0.601	0.476	0.561	0.176	0.48
RF	60	410	478	32	0.113	0.635	0.482	**0.591**	0.187	0.488
XGBoost	55	465	423	36	0.117	0.590	0.533	0.586	**0.192**	0.545
SVM	72	286	602	19	0.108	0.786	0.369	0.588	0.190	0.441
GLM	48	477	411	43	0.107	0.521	0.541	0.540	0.174	0.475
KNN	43	379	509	28	0.111	0.684	0.456	0.568	0.191	0.556
ANN	0	888	0	92	NaN	0	0.905	0.564	0	0.83

Table B.27: Results of Synthetic Minority Over-sampling Technique (SMOTE) and threshold adjustment combination using sliding window methodology

B.5 80 most important features

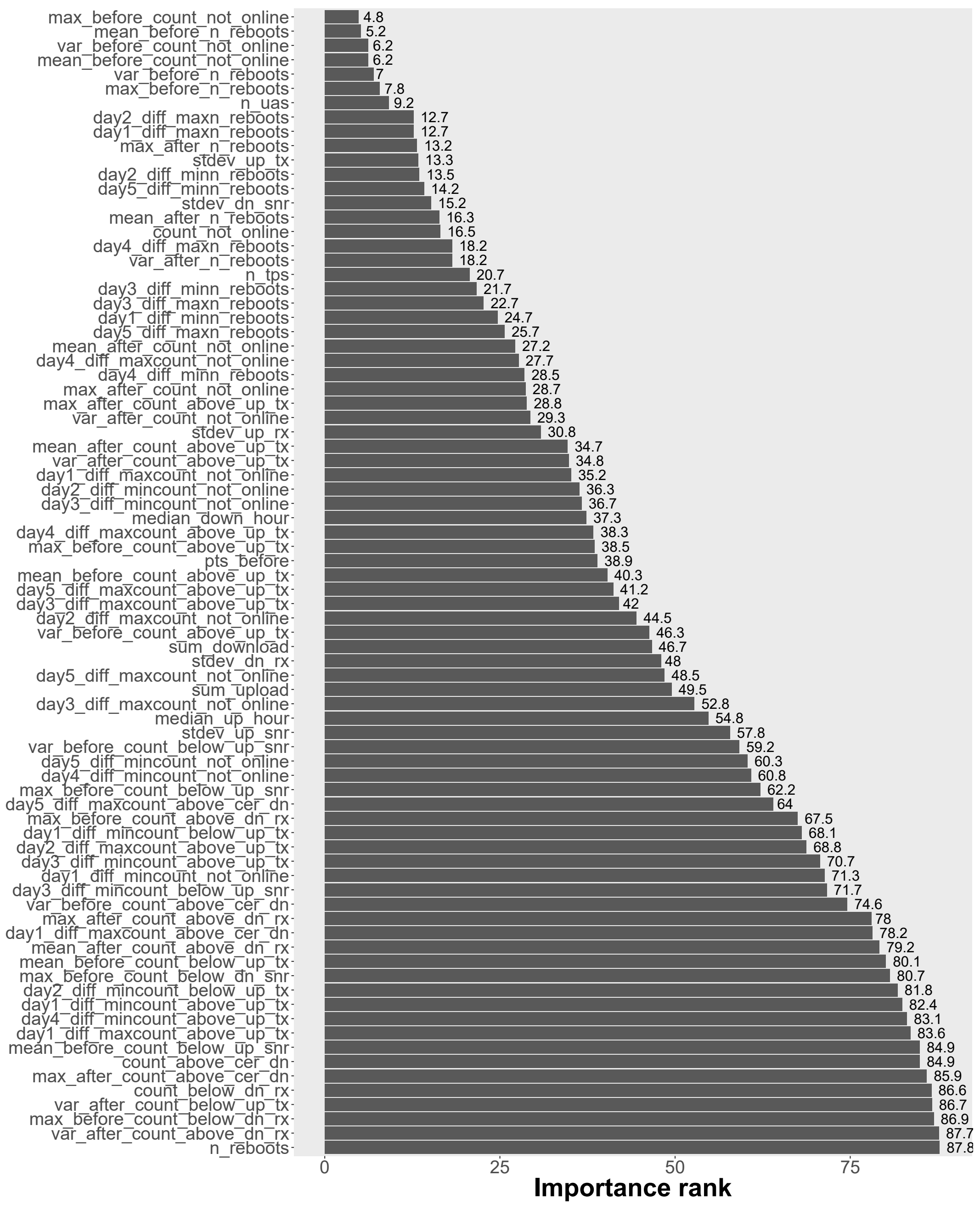


Figure B.1: Rank for the 80 most important variables with Synthetic Minority Over-sampling Technique (SMOTE) using Random Forest

B.6 Feature importance results

Top N Features	TP	TN	FP	FN	Precision	Recall	Accuracy	AUC	Fscore
20	77	226	662	14	0.105	0.843	0.310	0.566	0.187
30	80	190	697	12	0.103	0.864	0.277	0.563	0.184
50	45	571	316	47	0.125	0.489	0.631	**0.594**	0.198
65	46	551	337	45	0.124	0.503	0.611	0.584	0.198
80	39	634	253	53	0.134	0.423	0.688	0.593	**0.202**
100	50	529	358	42	0.122	0.542	0.592	0.593	0.200

Table B.28: Best results for each top N of most important features with Random Forest

Top N Features	Parameters	% under-sampling	% over-sampling	k
20	$mtry = 10$, $num.trees = 400$ $max.depth = 1$	25	300	1
30	$mtry = 10$, $num.trees = 100$ $max.depth = 1$	25	300	5
50	$mtry = 1$, $num.trees = 400$ $max.depth = 1$	50	500	3
65	$mtry = 1$, $num.trees = 50$ $max.depth = 1$	50	500	3
80	$mtry = 1$, $num.trees = 100$ $max.depth = 5$	50	500	5
100	$mtry = 3$, $num.trees = 50$ $max.depth = 5$	25	300	5

Table B.29: Parameters that led to he best results presented in Table B.28

Model	TP	TN	FP	FN	Precision	Recall	Accuracy	AUC	Fscore
CART	46	544	344	45	0.120	0.504	0.605	0.560	0.192
RF	39	634	253	53	0.134	0.423	0.688	**0.593**	0.202
XGBoost	64	392	495	27	0.115	0.703	0.465	0.573	0.198
SVM	70	324	564	22	0.110	0.761	0.402	0.587	0.193
GLM	44	558	330	47	0.119	0.484	0.616	0.564	0.191
KNN	60	411	477	31	0.112	0.653	0.480	0.581	0.191
ANN	28	739	149	64	0.164	0.306	0.783	0.580	**0.209**

Table B.30: Best results for 80 most important features with Synthetic Minority Over-sampling Technique (SMOTE) using growing window methodology

Model	**Parameters**	**% under-sampling**	**% over-sampling**	**k**
CART	$minsplit = 5$, $cp = 0.05$	25	200	3
RF	$mtry = 1$, $num.trees = 100$ $max.depth = 5$	50	500	5
XGBoost	$eta = 0.01$, $max\_depth = 15$ $nroud = 100$, $colsample\_bytree = 0.6$	10	150	5
SVM	$cost = 0.01$, $gamma = 0.1$	50	500	5
GLM	$epsilon = 1$	10	150	1
KNN	$k = 25$	25	300	5
ANN	$size = 3$, $decay = 1$ $maxit = 100$	25	200	3

Table B.31: Parameters that led to he best results presented in Table B.30

Bibliography

[1] Adi Bronshtein. A quick introduction to k-nearest neighbors algorithm, 2017. Accessed: 2019-04-29.

[2] Alan Agresti. *An introduction to categorical data analysis.* Wiley, New York, 1996. ISBN: 0471113387 9780471113386.

[3] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Machine Learning: ECML 2004*, pages 39–50. Springer, 2004.

[4] J. Armstrong and Fred Collopy. Error measures for generalizing about forecasting methods: Empirical comparisons. *International Journal of Forecasting*, 8(1):69–80, 1992.

[5] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN: 0-89791-497-X. doi:10.1145/130385.130401.

[6] Paula Branco. Re-sampling Approaches for Regression Tasks under Imbalanced Domains. Master's thesis, Faculty of Sciences of the University of Porto, 2014.

[7] Paula Branco, Rita P. Ribeiro, and Luís Torgo. UBL: an r package for utility-based learning. *CoRR*, abs/1604.08079, 2016.

[8] Paula Branco, Luís Torgo, and Rita P. Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Comput. Surv.*, 49(2):31:1–31:50, 2016. ISSN: 0360-0300. doi:10.1145/2907070.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[10] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, August 1996. ISSN: 0885-6125. doi:10.1023/A:1018054314350.

[11] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. ISSN: 0885-6125. doi:10.1023/A:1010933404324.

[12] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. ISSN: 0885-6125. doi:10.1023/A:1010933404324.

[13] Andrew Burgess. Predictive maintenance using ai – case study for mbnl, 2019. Accessed: 2019-04-09.

[14] IBM Knowledge Center. Regular and irregular time series, 2018. Accessed: 2019-01-28.

[15] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16: 321–357, 2002.

[16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM, 2016. ISBN: 978-1-4503-4232-2. doi:10.1145/2939672.2939785.

[17] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng, and Yutian Li. *xgboost: Extreme Gradient Boosting*, 2019. R package version 0.82.1.

[18] cloudera. Apache zeppelin - cloudera, . Accessed: 2019-08-22.

[19] cloudera. Hue database, . Accessed: 2019-08-22.

[20] Anna Corazza, Francesco Isgrò, Luca Longobardo, and Roberto Prevete. A machine learning approach for predictive maintenance for mobile phones service providers, 01 2017. doi:10.1007/978-3-319-49109-7_69.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995. ISSN: 0885-6125. doi:10.1023/A:1022627411411.

[22] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13 (1):21–27, September 2006. ISSN: 0018-9448. doi:10.1109/TIT.1967.1053964.

[23] Paul S. P. Cowpertwait and Andrew V. Metcalfe. *Introductory Time Series with R*. Springer, 1st edition, 2009. ISBN: 0387886974, 9780387886978.

[24] James Crawshaw. A phased approach to ai adoption in network operations & maintenance, 2018. Accessed: 2019-04-09.

[25] Google Developers. Classification: Roc curve and auc, 2019. Accessed: 2019-07-21.

[26] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1): 12:1–12:34, 2012. ISSN: 0360-0300. doi:10.1145/2379776.2379788.

[27] Andrew Estabrooks and Nathalie Japkowicz. A mixture-of-experts framework for learning from imbalanced data sets. In Frank Hoffmann, David J. Hand, Niall Adams, Douglas Fisher, and Gabriela Guimaraes, editors, *Advances in Intelligent Data Analysis*, pages 34–43. Springer, 2001. ISBN: 978-3-540-44816-7.

[28] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, 20(1):18–36, 2004. doi:10.1111/j.0824-7935.2004.t01-1-00228.x.

[29] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 82–88. AAAI Press, 1996.

[30] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[31] Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, and Tyler Hunt. *caret: Classification and Regression Training*, 2019. R package version 6.0-83.

[32] Vicente García, Ramón A. Mollineda, and J. Salvador Sánchez. A new performance evaluation method for two-class imbalanced problems. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 917–925. Springer, 2008. ISBN: 978-3-540-89689-0.

[33] C. Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche. [/.]*. Number pt. 1 in Studi economico-giuridici pubblicati per cura della facoltà di Giurisprudenza della R. Università di Cagliari. Tipogr. di P. Cuppini, 1912.

[34] José Hernández-Orallo. Probabilistic reframing for cost-sensitive regression. *ACM Trans. Knowl. Discov. Data*, 8(4):17:1–17:55, 2014. ISSN: 1556-4681. doi:10.1145/2641758.

[35] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, pages 679–688, 2006.

[36] Ilan Reinstein. Random forests(r), explained, 2017. Accessed: 2019-04-12.

[37] Nahua Kang. Multi-layer neural networks with sigmoid function— deep learning for rookies (2), 2017. Accessed: 2019-09-16.

[38] Kiyoshi Kawaguchi. The mcculloch-pitts model of neuron, 2000. Accessed: 2019-05-05.

[39] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, January 1994. ISSN: 0004-5411. doi:10.1145/174644.174647.

[40] Allen Kent, Madeline M. Berry, Fred U. Luehrs, and J. W. Perry. Machine literature searching viii. operational criteria for designing information retrieval systems. *American Documentation*, 6(2):93–101, 1955. doi:10.1002/asi.5090060209.

[41] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: one-sided selection. In *Proc. 14th International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[42] Marcus A. Maloof. Learning when data sets are imbalanced and when costs are unequeal and unknown, 2003.

[43] P. McCullagh and J.A. Nelder. *Generalized Linear Models, Second Edition*. Chapman and Hall/CRC Monographs on Statistics and Applied Probability Series. Chapman & Hall, 1989. ISBN: 9780412317606.

[44] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[45] Charles E. Metz. Ce: Basic principles of roc analysis. In *Seminars in Nuclear Medicine*, pages 8–283, 1978.

[46] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2019. R package version 1.7-0.1.

[47] Avinash Navlani. Support vector machines with scikit-learn, 2018. Accessed: 2019-09-16.

[48] Haydemar Nunez, Luis Gonzalez-Abril, and Cecilio Angulo. Improving svm classification on imbalanced datasets by introducing a new bias. *Journal of Classification*, 2017. doi:10.1007/s00357-017-9242-x.

[49] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018.

[50] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019.

[51] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979. ISBN: 0408709294.

[52] Rohith Gandhi. Support vector machine – introduction to machine learning algorithms, 2018. Accessed: 2019-05-19.

[53] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[54] scikit learn. Ensemble methods. Accessed: 2019-09-16.

[55] scikit learn. Logistic regression, 2019. Accessed: 2019-07-09.

[56] Colin Shearer. The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4), 2000.

[57] Atish P. Sinha and Jerrold H. May. Evaluating and tuning predictive data mining models using receiver operating characteristic curves. *J. Manage. Inf. Syst.*, 21(3):249–280, 2004. ISSN: 0742-1222. doi:10.1080/07421222.2004.11045815.

[58] Apache Spark. Apache spark - unified analytics engine for big data. Accessed: 2019-08-22.

[59] Floris Takens. *Detecting strange attractors in turbulence*, pages 366–381. Springer Berlin Heidelberg, Berlin, Heidelberg, 1981. ISBN: 978-3-540-38945-3. doi:10.1007/BFb0091924.

[60] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2018. R package version 4.1-13.

[61] L. Torgo. *Data Mining with R, learning with case studies*. Chapman and Hall/CRC, 2010.

[62] Emmanuel-Lin Toulemonde. *dataPreparation: Automated Data Preparation*, 2019. R package version 0.4.0.

[63] Tutorialspoint. Machine learning - artificial neural networks, 2019. Accessed: 2019-05-05.

[64] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.

[65] William Vorhies. Crisp-dm – a standard methodology to ensure a good outcome, 2016. Accessed: 2019-07-13.

[66] Doug Wielenga. Identifying and overcoming common data mining mistakes. *Forum American Bar Association*, pages 1–20, 01 2007.

[67] Marvin N. Wright and Andreas Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017. doi:10.18637/jss.v077.i01.

[68] Apache Zeppelin. Apache zeppelin. Accessed: 2019-08-22.

[69] Xingquan Zhu and Ian Davidson. Knowledge discovery and data mining: Challenges and realities. 01 2007. doi:10.4018/978-1-59904-252-7.