TÉCNICO LISBOA

Predict Lost Flights Connections. An Interpretable Machine Learning Approach

Hugo Miguel Silva Lopes

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Cláudia Alexandra Magalhães Soares
Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. Maria da Conceição Esperança Amado

December 2021

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

O trabalho desenvolvido nesta tese não teria sido possível se não fosse pelo constante apoio, aconselhamento e assistência dos meus orientadores, Professora Cláudia Soares e Professor Rodrigo Ventura, aos quais só tenho a agredecer a dedicação e ajuda. Adicionalmente gostava também de agradecer ao Departamento de Informática da FCT NOVA pela utilização do seu cluster, que me permitiu desenvolver os modelos computacionalmente exigentes presentes neste trabalho.

Igualmente importante é agredecer todo o suporte que obtive da minha familia, não só durante a execução do presente trabalho, mas também ao longo destes últimos 5 anos. Um muito obrigado a todos vós por tudo. Obrigado também a todos os meus amigos e aos meus colegas de curso, em particular aos 6, por me terem acompanhado ao longo deste percurso e por terem tornado a experiência de ser estudante em algo inesquecivel. Por último, o meu muito obrigado à Maria e à Teresa por me terem feito companhia ao longo destes últimos meses e por me terem motivado durante este periodo.

Resumo

A otimização da programação de voos e a satisfação dos passageiros são problemas que afetam profundamente as receitas do setor da aviação civil. A perda de voos de ligação, que muitas vezes resulta da falta de mecanismos preventivos, afeta as operações regulares das companhias aéreas e consequentemente as suas receitas e imagem. Propomos uma nova abordagem para a previsão do sucesso das conexões dos passageiros com um foco na interpretabilidade, uma vez que o sucesso das conexões é fundamental para o lucro das companhias aéreas, e que a tomada de decisões por parte dos dirigentes requer explicações que sustentem tais escolhas de gestão. Os modelos foram desenvolvidos a partir de dados da atividade da TAP Air Portugal de janeiro de 2019 a fevereiro de 2020. Os dados foram analisados em conjunto com alguma *feature engineering*, incluindo a codificação de variáveis e a geração de novos dados para reequilibrar o problema. No total, estudamos cinco modelos, dois não interpretáveis e três interpretáveis. Os resultados dos modelos interpretáveis não foram tão bons quanto os resultados dos modelos não interpretáveis, mas o desempenho dos modelos interpretáveis na classe minoritária, as conexões perdidas, foi próximo ao visto no melhor modelo não interpretável. As métricas usadas incluíram o *Recall* na classe minoritária e o *Recall macro-average* na tarefa de classificação global. Todos os modelos sugeriram que a variável mais crítica nas previsões é o tempo agendado para a conexão e todos eles não atribuiram grande importância a variáveis como a idade ou o género.

Palavras-chave: Aprendizagem Automática, Classificação em Dados não Balanceados, Explicação de Modelos, Modelos Interpretáveis, Voos de Ligação

Abstract

In airlines, flight schedule optimization and passenger satisfaction are problems that profoundly impact the airline industry revenue every year. Missed connections are often a consequence of unexpected disruptions and the lack of preventive mechanisms that affect airlines' regular operations and image. This thesis proposes a new approach for models to classify the success of passengers' connections through an airline hub, focusing on interpretability. This issue is key to airline profitability since decision-makers often want to have hard evidence before taking action. The models were trained on data from TAP Air Portugal's passenger activity from 2019 and the beginning of 2020, along with some data from airport movements. We analyzed the data and did some feature engineering, including encoding some features and generating new samples to re-balance the dataset. In total, we studied five models, two non-interpretable plus three interpretable models. The overall accuracy of the interpretable models was not as good as the results from the non-interpretable models. However, when looking for critical metrics for imbalanced data, as this is the case, and the performance on the minority class, i.e., missed connections, the interpretable models had a performance close to the one seen in the best non-interpretable model. These metrics included the Recall on the minority class and the macro-average Recall of the classification task as a whole. All models suggested that the most critical feature is the time scheduled for the connection and all of them gave none to marginal importance to features such as age or gender.

Keywords: Flight Connections, Imbalanced Classification, Interpretable Models, Machine Learning, Model Explanation.

Contents

List of Tables

List of Figures

List of Acronyms

ACARS Aircraft Communications Addressing and Reporting System

AIC Akaike Information Criterion

AMRP Aircraft Maintenance Routing Problem

AP Average Precision

AUC Area Under the Curve

BIC Bayesian Information Criterion

CART Classification and Regression Trees

CSP Crew Scheduling Problem

DNN Deep Neural Networks

EDA Exploratory Data Analysis

ELU Exponential Linear Unit

FAP Fleet Assignment Problem

FN False Negative

FNR False Negative Rate

FP False Positive

FPR False Positive Rate

FSP Flight Scheduling Problem

GBM Gradient Boosting Machine

GMM Gaussian mixture model

IML Interpretable Machine Learning

k-NN k-nearest neighbors

MAR Missing at Random

MCAR Missing Completely at Random

MCT Minimum Connecting Time

ML Machine Learning

MNAR Missing not at Random

NN Non-Schengen to Non-Schengen

NS Non-Schengen to Schengen

OCC Operations Control Center

PR Precision-Recall

PReLU Parametric leaky Rectified Linear Unit

ReLU Rectified Linear Unit

RIPPER Repeated Incremental Pruning to Produce Error Reduction

ROC Receiver Operating Characteristic

SHAP SHapley Additive exPlanations

SMOTE Synthetic Minority Oversampling Technique

SN Schengen to Non-Schengen

SS Schengen to Schengen

TN True Negative

TP True Positive

TPR True Positive Rate

XGBoost Extreme Gradient Boosting

Chapter 1

Introduction

With the ever-increasing passenger demand, airports worldwide have to face traffic congestion problems at several levels, including, but not limited to, arrival and departure delays and bottlenecks within terminal facilities. Hence, with the development of the Intelligent Transportation System (ITS), on which large amounts of data are recorded every day, many approaches have been proposed to deal with the different congestion problems based on data collection from ITS. The problems airports face are also extremely important for airlines that see the flight schedule as a critical factor for their profitability success and client satisfaction.

However, and this is not a problem specific to this domain, the approaches being developed nowadays tend to follow the widespread belief that the most accurate models must be inherently complicated and non-interpretable by humans. These are commonly called Black-Box models, and even the human entity in charge of modelling cannot understand what combination of variables the model is basing its predictions on. The result is a model with no cap on complexity.

The use of such models can be especially harmful when dealing with decisions that have a direct impact on people's life. As stated by Rudin and Radin [1] studies have shown that complicated Black-Box models used to predict the likelihood being arrested in the future are not more accurate than simple predictive models. Moreover, Angelino et al. [2] showed that rule lists with a certificate of optimality could be as accurate as a state-of-the-art proprietary risk prediction tool, but that is entirely interpretable. Non-interpretable classification models defined by Guidotti et al. in [3] include Neural Networks, Deep Neural Networks, Tree Ensembles and Support Vector Machines.

Conversely, White-Box models are models that try to answer the same type of questions as Black-Box models do but are fundamentally different from them in the sense that the former are mannered so that they provide reasoning on how the algorithm reached its predictions, something that did not happen with the latter. Interpretable models are often comprised of simpler models. However, most of them are not designed with interpretability issues in mind, they are just designed, like all other Machine Learning (ML) algorithms, to be as accurate as possible, and their interpretability is just an afterthought.[1]

The interpretability of ML models can be further branched according to several criteria. One of those criteria is the mechanism with which the interpretability is reached. Intrinsic and *post hoc* distinguishes if

interpretability is achieved by capping the ML model complexity or by applying methods that analyze the model after training, respectively [4]. An example of intrinsic interpretability refers to models with a simple structure and are therefore considered interpretable, such as shallow decision trees. In contrast, a *post hoc* interpretability model refers to applying explanation methods after the model training is concluded. *Post hoc* explanation methods can be applied to both White-Box models as well as Black-Box models. The definition of interpretability can also be branched in terms of the scope of the interpretability. Local or global if it explains a single instance, the model as a whole, respectively.

Examples of intrinsic interpretable models include Linear Regression, Logistic Regression, short Decision Trees and Decision Rules algorithms where RuleFit is included. Here, the depth of the tree is important since the number of leaves in a tree grows exponentially with the tree depth. The RuleFit algorithm was proposed by Friedman and Popescu [5] and it is capable of learning sparse linear models that include interaction effects in the form of decision rules that are automatically detected.

It is important to mention that despite the development of some algorithms capable of explaining Black-Box models predictions, this does not alleviate its inherent problems and the modeller confidence in the model's output still requires a high degree of abstraction. The solution to stop perpetuating bad practices is to design inherently interpretable models, as explained by Rudin in [6].

1.1 Topic Overview

This thesis addresses a conjugation of two problems and both of them have some literature on their specific broad topics. However, literature studying an interpretable approach to the prediction of missed connections is not yet available. In this section, we provide a wide but non-exhaustive review of works related to this thesis.

1.1.1 Model Interpretability

Interpretability in the context of ML was defined by Murdoch et al. [7] using a unique framework called PDR — Predictive, Descriptive, Relevant — for discussing interpretations. This framework provides three prerequisites for evaluation: predictive accuracy, meaning the quality of a model's fit; descriptive accuracy, the degree to which an interpretation method captures the relationships learned by the model; and relevancy, if it provides insight for a particular audience into a chosen domain problem with relevancy primarily judged relative to a human being. Before this definition, Doshi-Velez and Kim [8] described interpretability in terms of the model's ability to elucidate in intelligible terms a human.

Although only recently gaining importance, Interpretable Machine Learning (IML) models have been around under-explored for many years [9]. Linear regression models were used as early as the 19th century and have since then grown into, for example, generalized additive models [10] by Hastie and Tibshirani.

After the mid-2010s and even with the rapid development of deep learning models the research in the field of IML did not stop and many new IML methods have been proposed since. Many of those mod-

els are model-agnostic, but explanation techniques specific to deep learning and tree-based ensembles models were also subject to research. Nowadays, regression analysis and rule-based ML remain relevant topics and the linear regression model has seen many extensions proposed [11]. Rule-based ML research extensions have also been proposed. For example, these two domains are even blending as seen in model-based trees [12] or the RuleFit algorithm [5]. Both regression models and rule-based ML models serve as native ML algorithms, and as sub-blocks for many IML approaches.

In terms of the current research stage, the IML field has seen consolidation in terms of knowledge with, for example, [13] and work about defining interpretability like [14] or evaluation of IML methods [15].

1.1.2 Flight Operations

Historically, the airline scheduling problem is separated into four more minor problems: Flight Scheduling Problem (FSP), Fleet Assignment Problem (FAP), Aircraft Maintenance Routing Problem (AMRP), and finally Crew Scheduling Problem (CSP), as stated by Eltoukhy et al. in [16].

In past work, we can see that most of the attempts did not include an effort to cover all sub-problems of the big picture problem at once. The majority of researches focused on one stage only, as seen in the work from Etschmaier and Mathaisel [17] that focused on the flight schedule, or in the work from Sherali et al. [18] and Gopalakrishnan and Johnson [19], which focused on fleet assignment and crew scheduling, respectively.

The first problem that airlines typically need to solve before start operations is the FSP since the other sub-problems are dependent on this. The goal is to reach the end of this stage with a timetable containing a list of all flight schedules, while considering some constraints and limitations, just as presented by [20] that considered the influence of market constraints such as passenger demand and ticket price.

Still in the topic of FSP Yan and Young first presented in [21] a study on the sub-problem that considered the expectation of demand variation by passengers. This work was later improved by Yan and Tseng [22]. The difference between this second work and the original research by Yan and Young was the consideration of the airline's market share. However, the two models failed to consider the variability and uncertainty of the market share percentage and the demand by passengers. To overcome these limitations, other studies by Yan et al. [23] and Jiang and Barnhart [24], investigated the variability of market share and passenger demand. These two works shifted towards a more authentic understanding of the airline industry since they consider market share fluctuations and passenger demand. Nonetheless, several other researchers worked on the flight scheduling problem and methods to increase its robustness. That was the case in the work of Lan et al. [25].

Regarding the other dimensions of the overall problem, Abara [26] presented a simple FAP solution by applying integer linear programming based on the structure of a connection network and then Hane et al. [27] and Rushmeier and Kontogiorgis [28] improved upon Abara's model. These models were good starting points. However, the assumptions of fixed flight schedules and deterministic passenger demand

compromise the applicability of these models in actual conditions.

When it comes to the other two pillars of the problem, the '90s saw researchers proposing solutions for the AMRP with a focus on the tactical side of the problem. The works of Liang and Chaovalitwongse [29] and Talluri [30] are examples of such approaches. However, these approaches fail to consider some of the characteristics of the requirements of operational maintenance. Over the 2000s, research on the AMRP continued. However, it shifted and was dedicated to proposing models with a focus on the operational side of the problem, like seen by Sriram and Haghani [31] that proposed an effective operational model. However, this model was limited in terms of the number of flights that it could handle. This limitation was then improved by Başdere and Ümit Bilge [32] to handle a large number of flights. The last pillar, the CSP, moved from a daily horizon planning, like in the work of Hoffman and Padberg [33], that despite being capable of handling large-scale problems and producing crew pairing to each specific day of the week had the limitation of assuming a daily repetition of all flights in order to simplify the computation. Since both the daily repetition and the weekly repetition assume a fixed departure time, a stochastic and robust crew pairing approach was developed by Muter et al. [34] to balance this point.

One of the main drawbacks of past solutions and approach is that each stage, i.e, each sub-problem, is solved independently of other sub-problems, which means that the solution for one part of the problem as a whole, might not be optimal for the following steps. This motivated researchers to pursue models that integrate multiple sub-problems simultaneously. An example of such innovation was the introduction of integrated models to simultaneously solve both the FSP and the FAP at once, by Lohatepanont and Barnhart [35].

Moreover, most approaches presented so far correspond to algorithms and mathematical solutions for their respective issues and have failed to work with real-world data. Studies using collected data tend to focus on the prediction of flight delays and ways to mitigate them. An example of that was the work introduced by Rebollo and Balakrishnan [36], on which they proposed a new model for predicting air traffic delays using both temporal and network delay states with a 2h to 24h advance. Wu and Law [37] investigated the relationship between delays, delay propagation, and delay causes with aircraft, crew connections and passenger connections using a Bayesian Network in a delay-tree framework. The work by Kafle and Zou [38] proposes a novel analytical-econometric approach to understand delay propagation patterns and associated mitigation measures using flight data as the backbone of the analysis.

Some authors also studied the repercussions of flights delays, complimenting the work from [38]. Li and Jing [39] proposed a delay causality network and Fageda and Flores-Fillol [40] investigated the mutual influence between the airline network structure and the airport congestion. Their study suggested that airlines with hub-and-spoke structures react less to delays than airlines operating fully connected networks.

The problem of predicting flight delays based on flight information only was the object of study in more recent research. For example, Yu et al. [41] presented a study analyzing data from an airport and presented a deep learning flight delay prediction model based on a multi-factor approach. The development of models with passenger data is generally more difficult due to the lack of available resources.

However, Bratu and Barnhart [42] established relationships between several factors. These included, both passenger and flight delays, and passenger cancellation rates and load factors. Guimarães [43] developed multiple ML frameworks centred around passenger data that additionally did a *post hoc* explanation analysis for the Black-Box model predictions. However, all those frameworks used the same Black-Box model, Extreme Gradient Boosting (XGBoost), and did not try to include intrinsically interpretable models.

1.2 Objectives and Deliverables

This work aims to develop two sets of distinctive predictive models belonging to the interpretable and non-interpretable categories.

The work uses different ML approaches to determine whether a passenger will have a successful connection or not. Afterwards, the models are compared to understand what types of benefits come with using one model category or the other. All models perform classification tasks trained on historical data collected by TAP regarding their operations, and all models receive the same information and make the same type of predictions. With these models, it is possible to identify what type of feature combinations the models are basing their predictions on.

This thesis contribution is a novel approach to determine whether a passenger will have a successful connection or not. This approach uses the same data structure and content across multiple algorithms and:

1. develops two sets of distinctive predictive models belonging to the interpretable and non-interpretable categories;
2. compares the models performance and costs to understand what types of benefits come with the use of one model category over the other;
3. identifies what type of feature combinations the models are basing their predictions on.

1.3 Thesis Outline

This document is organized as follows. The next chapter, Background, outlines the understanding of key concepts of the scope of this topic. Chapter 3, Data Preparation, analyzes the initial data while discussing its content and cleanup procedures employed, as well as encompassing the feature engineering and the generation of new data to re-balance the dataset. Chapter 4, Modelling, shows the different algorithms, parameters tuning and their result scrutiny. Finally, Chapter 5, Conclusions and Future Work, summarizes the developed work and enumerates future work directions.

Chapter 2

Background

The following sections introduce the theoretical concepts and techniques considered in this work. They will serve as a foundation for the assumptions in this work.

2.1 Supervised Learning

In the field of ML the primary learning mechanisms are:

1. supervised learning, when the correct output is known for each set of inputs;
2. reinforcement learning, when the output is unknown but we know a reward, i.e., a grade evaluating the action taken by the learning agent;
3. unsupervised learning, when no target output is known.

As the problem being studied is of the first type, only supervised will be studied further.

Central to supervised learning is the concept of training, the iterative improvement in learning. In this mechanism, the data used to feed the model is divided into training and testing sets (and possibly a validation set) [44]. A validation set, shall it be considered, is used simultaneously with the training set as a means to validate the performance of the model on the training set, which is helpful to tune the learning parameters of algorithms, also known as hyperparameters. The test set serves as the unseen dataset to test the generalization error of the model that was developed.

Furthermore, there is another way to divide supervised learning, this time regarding the type of predictions made by the algorithm. Regression and Classification are the most common learning tasks in supervised learning problems. In essence, classification is about predicting a discrete category label, whereas regression predicts a continuous value. In classification problems, the output variable is usually called label or category, and the mapping function that predicts the results given the input infers the category of a given observation. A common example to explain classification tasks relates to filtering emails for spam or not spam. In regression problems, the output of the learning process is either an integer or a floating-point, for a given observation. Regression problems can be extended and may

include more than one prediction of the output. As this work is a classification problem, the focus will be on this type of problem.

When dealing with supervised learning methods for classification, several different algorithms are available, such as Decision Trees, Neural Network, Nearest Neighbor Classifiers, and Support Vectors Machines. Tree-based algorithms are among the methods most widely used for classification tasks due to the simple interpretations they output.

2.1.1 Logistic Regression

Logistic Regression [45] is one of the simplest models available for binary classification problems. It works by finding the probabilities of the different instances belonging to each of the classes. We can think of the Logistic Regression model as an application of the linear regression to classification problems.

The Logistic Regression algorithm fills the gap that exists in linear regression models since these models fail to work appropriately in classification tasks. Classification tasks require anchor values to separate classes; however, the linear model output is the hyperplane that minimizes the distance between the data points belonging to each class and the hyperplane. In the case of the linear regression model, the meaning of the distance is not related with the probability of inclusion in a specific class or another. Furthermore, since linear models do not support output probabilities but only distances to the hyperplane, the boundary between classes is too rigid.

The solution to perform binary classification tasks is, therefore, Logistic Regression. This algorithm limits the model's output to the range between 0 and 1 and to be a probability instead of trying to fit a hyperplane, as with the linear Regression. The algorithm uses the logistic function, which is defined as:

$$\text{logistic}(\eta) = \frac{1}{1 + \exp(-\eta)} \tag{2.1}$$

Since the output values of the Logistic Regression are limited to the range between 0 and 1, the interpretation of its weights vastly differs from the interpretation of the weights of the standard linear regression model. Unlike the latter, the relationship between the coefficients of the Logistic Regression and the output is not linear. This happens since the weighted sum, η term of Equation (2.1), is converted by the logistic function to a limited range.

A solution to extract insight from the weights is to look at the "odds" function, which represents the probability of the occurrence of a given event divided by the probability of the same event not happening.

$$\text{odds} = \frac{P(y=1)}{P(y=0)} = \exp(\beta_0 + \beta_1 x_1 + ... + \beta_j x_j) \tag{2.2}$$

where x_j represents feature j value and β_j represents the weight attributed to feature j. A simple mechanism that makes the Logistic Regression more intelligible is to understand what it means to change the value of one of its features by 1 unit. To do so, we check the ratio between the two odds:

$$\frac{\text{odds}_{x_j+1}}{\text{odds}_{x_j}} = \exp(\beta_j) \tag{2.3}$$

Equation (2.3) shows that changing a feature by one unit induces a change in the odds ratio by a factor of $\exp(\beta_j)$. This means that when dealing with numerical features, the odds ratio changes by the factor mentioned above whenever the feature is increased of one unit. Categorical features, on the other hand, have a different interpretation and if j is a binary feature, changing the feature value from the reference point, changes the odds by the factor mentioned above and if j is a non-binary categorical feature, a preprocessing form of one-hot-encoding is required, and the interpretation is the same as in the case of binary categorical features. The $\exp(\beta_0)$ corresponds to the situation when all numerical features are zero and the categorical features are at the reference levels. This corresponds to the intercept value, and its interpretation is usually not relevant.

2.1.2 Decision Trees

Decision Trees [46] are a method that allows to classify or predict values. However, and unlike the Linear and Logistic Regression models, they are non-linear. Decision Trees as learning algorithms are attractive because their output is easy to interpret and study. For the construction of such structures, the principle is to separate all observations into different categories. In the end, the goal is to have similar observations included within the same category and different observations included in different categories. After the training stage, the result is a list of rules that can be easily expressed and explained to a human, translated to other machine languages, and applied to new, unseen observations to make predictions.

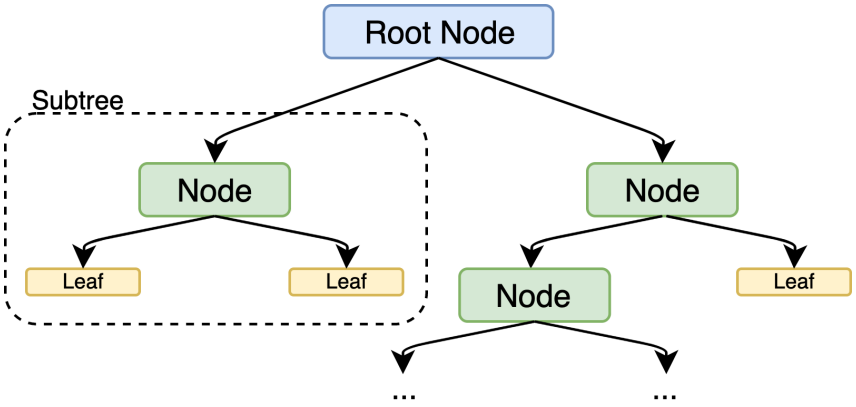


Figure 2.1: Decision Tree Schematic Representation. Illustrative representation of the basic structure of a decision tree, where the colors represent the hierarchical importance level, starting with blue followed by green and then yellow.

The process to build a decision tree entails two steps, growing and pruning a tree. The first step starts from the root node; then, the algorithm enumerates all possible splits to find the best split for the node being investigated. When a split is performed, multiple descendants are created, and the process

is further repeated. Nodes that are not split further become a terminal node, i.e., a leaf. Figure 2.1 shows the basic structure of a Decision Tree. The second step, the pruning phase reduces the risk of overfitting by removing overgrown sub-trees that do not add accuracy.

There are, in broad terms, three prominent families of tree growing algorithms: the Classification and Regression Trees (CART) family; the ML family; and the AID family. These three families mainly differ in the type of splitting criterion. The CART family, for example, uses the concept of impurity as defined by Breiman et al. [47].

The analysis will focus on the CART family as this will be the basis for one of the explainable models. As stated before, the reasoning behind the choice for splitting nodes is based on the impurity that can be measured either in terms of the Gini or Entropy criteria. In simple terms, the Gini impurity quantifies the frequency of mislabelled elements, and it is calculated according to:

$$\text{Gini} = 1 - \sum_j p_j^2 \tag{2.4}$$

where p_j is the probability of belonging to class j. The minimum value possible, 0, corresponds to the case when the node is pure and should not be further split, whereas the maximum possible value, 0.5, occurs when the probabilities of the two classes are the same.

The entropy, $H(x)$, is calculated using:

$$\text{H(x)} = -\sum_j p_j(x) \log_2(p_j(x)) \tag{2.5}$$

Where $p_j(x)$ is the probability of x belonging to class j. In simple terms, entropy measures the level of disorder. As with the Gini Index, the optimum split is chosen by the feature with less entropy. Here, the maximum value is attained when the probability of the two classes is the same, and a node is considered pure when the entropy has its minimum value, which is 0.

The interpretation of a Decision Tree is relatively simple: from the root node, we follow the path to the next node based on the relation between the instance being evaluated and the rules in the node being evaluated. Then, we repeat the earlier step until a leaf node is reached, which means we obtained the model's prediction for the outcome of the instance being evaluated.

The importance of each feature can be assessed by looking at all splits in which the specific feature was used and assessing how they contributed to the reduction in error compared to the parent node. We can think of each feature importance as a part of the sum of all features importance since this value is scale to 100 [48].

The decision tree algorithm presents some advantages, some of which were already disclosed. Firstly, the tree structure can capture non-linear relationships between features. Then, the interpretation is simple even when compared with other White-Box models since the interpretability is helped by the fact that the tree structure has a human-friendly graphical representation. Another characteristic, besides the structure, that helps its interpretability is that we do not need to transform features to extract insights from decision trees, unlike the case with Logistic Regression where, for example, some feature

engineering can provide a more intelligible interpretation.

However, using decision trees as classification algorithms also present some disadvantages. One of them is the fact that decision tress cannot learn linear relationships between features. These models try, instead, to approximate linear relationships using strict splits, which creates a step function that might not be efficient for some problems [48]. This disadvantage is highly correlated with another characteristic that is also not desirable, which is the lack of smoothness seen in these algorithms since minor changes in a feature can significantly impact the predicted outcome. Lastly, the primary advantage of this algorithm is also one of its significant limitations since the intrinsic interpretability only exists as long as the trees are short.

2.1.3 Ensemble Learning Algorithms

Ensemble is a ML concept where the idea is to use multiple classifiers belonging to the same learning algorithm class. These methods help reduce the leading causes of learning errors like noise, bias, and variability.

Bagging

Bagging [49], a word coming from the junction of the words bootstrap and aggregating, is an ensemble method to improve stability and accuracy. Bootstrapping is a type of sampling where the instances that will ultimately form the basis for the new set, are randomly chosen with replacement from the original set of data.

The models are then fed with the newly form sub-sets, and the model predictions are aggregated to be combined for the final prediction. Therefore the bagging mechanism includes a bootstrapping stage followed by an aggregation of the results. In Bagging, the result is obtained by gathering the responses of all learners — majority voting. A graphical explanation of the bagging concept is depicted in Figure 2.2(a).

Boosting

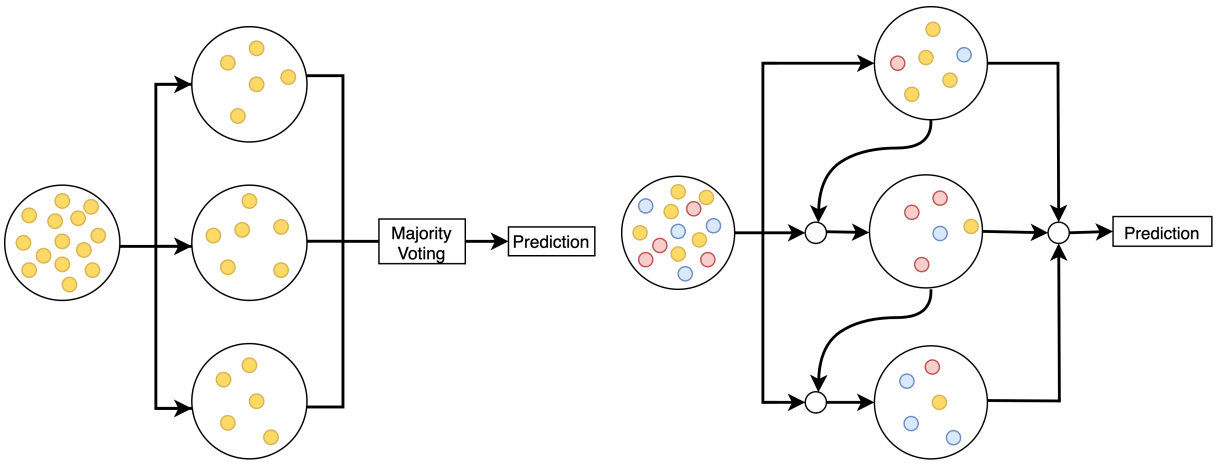
Boosting [50], contrarily to bagging, is not a parallel mechanism. Instead, it is a sequential ensemble algorithm where the assigned weights and elements depend on the previous fitted functions. With Boosting, the subset generation is not random and depends on the performance of the previous models: every new subset focus on the elements that previous models mislabeled.

In Boosting, the algorithm assigns weights to each resulting model based on their results on the training data, and the final prediction is obtained by taking into consideration the responses of all learners and their respective weights. A graphical explanation of the boosting concept is depicted in Figure 2.2(b).

Random Forests

Random Forests are an ensemble algorithm that combines tree predictors sampled from the training data, which are trained and then averaged. With this algorithm the model benefits from averaging since

trees are known to be able to capture complex structures in data but are also known for capturing noise in the process.



(a) Bagging (b) Boosting

Figure 2.2: Bagging and Boosting Schematics.

Gradient Boosting

Gradient Boosting, also known as Gradient Boosting Machine (GBM), creates an ensemble model of weak predictive models, e.g., decision trees, using the gradient descent technique. This technique is used to find local minimum of differentiable functions. Gradient Boosting aims to minimize the cost function by randomly sampling trees where each new tree is built and trained to reduce the errors of previous trees. In each iteration, the algorithm fits a weak learner, and the error yielded by the weak learner is used to compute the gradient of the loss function. With this approach, the algorithm determines the direction in which the parameters must be changed to minimize the error.

XGBoost

The XGBoost [51] is an ensemble ML algorithm based on the gradient boosting mechanism and has a decision tree basis. The XGBoost can be used to solve various ML problems where the tasks discussed so far are also included: regression and classification.

The XGBoost algorithm is the most recent stage of the decision tree-based algorithm development timeline. This development process started with the simple decision tree concept and then evolved to the ensemble methods described earlier: bagging, boosting, random forests, and finally, gradient boosting. Like GBM, the XGBoost is a method that leverages the concept of boosting weak learners by using the gradient descent architecture. However, XGBoost takes this a step further from the GBM framework and improves it through both system and algorithmic optimizations.

In terms of system optimization, it is important to mention:

1. the parallelization since the XGBoost algorithm uses a parallel implementation of tree building;

2. the XGBoost uses a different stopping criterion than the one seen with the GBM and uses the specified maximum depth of the tree as a first stop criterion which significantly improves its computational performance;

3. the optimization introduced by XGBoost allows for efficient use of hardware resources available, and it is based on 2 foundations. The first is the allocation of internal buffers to store gradient statistics and the second is further enhancements like 'out-of-core' computing that optimizes available disk space allowing for the handling of big data-frames that would not fit into memory otherwise.

On the other hand, the XGBoost algorithm is optimized to leverage Regularization, Sparsity, and Cross-validation. With regularization, XGBoost penalizes complex models using L1 and L2 regularization, which helps prevent overfitting. Then the algorithm can efficiently handle different types of sparsity patterns in the data by using sparsity-aware splits.

2.1.4 RuleFit

The RuleFit, proposed by Friedman and Popescu [5], is a rule-based algorithm capable of learning sparse linear models.

RuleFit, together with decision trees, fills the gap for simple models that are also easy to interpret, just like linear models. However, RuleFit takes it a step further from decision trees and also integrates interaction between the features. In the modelling phase, RuleFit uses as features both the original features present in the data as well as some new features called decision rules. These newly generated pseudo-features are the key to capturing interactions among the original features. They are generated based in the structure of decision trees and each path from the root to the leafs of tree can be converted into a decision rule by merging all split criteria into one single rule.

Furthermore, Molnar [48] presents the major advantages and disadvantages of RuleFit. To start, the algorithm is versatile like Decision Trees and, unlike the Logistic Regression algorithm, is capable of handling classification and regression problems. Then, the generated rules are intelligible (as long as the number of conditions is low). Ideally, the number of split criteria used to generate a rule should not exceed three, which can be though of as setting a maximum depth of three for the trees. The algorithm interpretability is also helped because the rules are binary and either apply to an instance or not, and even with many rules, only a part of those will apply to each particular instance.

As a downside, RuleFit sometimes creates too many rules that get a nonzero weight, which degrades the interpretability of the model, and its performance, especially in classification tasks, may be disappointing.

2.1.5 Neural Networks

Besides the algorithms presented earlier, another type of algorithm that can be used for classification problems is the popular Neural Networks. These algorithms belong to the Deep Learning field and, in simple terms, consist of multilayer networks of neurons used to make predictions.

Among the most successful Neural Networks learning algorithms are the networks known as Deep Neural Networks (DNN). Neural Networks are inspired by human neural networks, with the difference that the former is more limited than the latter in terms of their domain of understanding. Within a Neural Network, information is fed to the input end of the network. Then it flows through the network's structure until it reaches the output layer. The output result can be presented in multiple forms, such as integers or floats. In the case of a binary classification problem, the output is a single float value indicating the probability of belonging to class 0.

Their basic architecture includes:

- **Input layer:** Includes neurons (with the same number of available features in the data set) that receive the input information and pass it onto the next layer.
- **Hidden layer(s):** Corresponding to all layers in between the input and output layers. They contain a variable number of neurons that apply transformations to the inputs they receive before passing them onto the next layer. As the network is trained, the weights are updated to have more predictive power. It is possible to have a Neural Network without any hidden layer.
- **Output layer:** Corresponds to the final layer of the network and depends on the type of network being built. In the case of a multiclass classification problem, it will have as many neurons (units) as classes are being predicted, and in the case of a simple two-class binary classification, the output layer will have only one unit. Figure 2.3 shows a simplified architecture of a Neural Network.

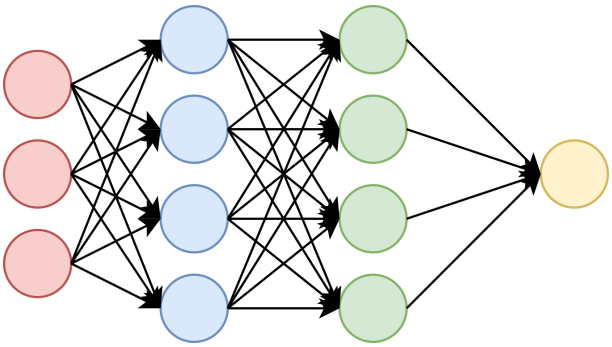


Figure 2.3: Neural Network schematic.

An important player within the structure of Neural Networks are the activation functions. In basic terms, the activation function take as input the output signal from the last neuron that belongs to the previous layer and transforms it into some form of information that can be taken as input to the next neuron.

The activation functions follow non-linear equations. The main reasons to use non-linear functions in a neural network instead of linear alternatives are performance and capabilities. The first reason is that the activation function's input is $Wx+b$, where matrix W corresponds to the cells' weights, x corresponds to the inputs, and then there is an additional term added called bias. If not restricted, this value can grow high in magnitude, especially in the case of profound networks, and lead to computational issues. The second reason to use non-linear functions is to add non-linearity to the Neural Network.

There are several types of nonlinear activation functions, where we discuss only a few:

- **Sigmoid:** The sigmoid function, also known as Logistic function, is defined as sig(t) $= \frac{1}{1+e^{-t}}$. This function is, mathematically, the same as the Logistic Regression algorithm presented earlier. Nowadays it is not widely because it is computationally expensive due to the exponential operation. As the output from the sigmoid function is not centered at zero and the function derivative at saturation approximates zero this function can suffer from the vanishing gradient problem. However, this method is still used for binary classification problems.

- **Softmax:** This function can be considered a generalization of the sigmoid function. The difference is that it is used in multiclass classification problems. Similar to sigmoid, it outputs values between 0 and 1 and is used as the final layer in classification models.

- **Tanh:** The tanh function, also known as Hyperbolic Tangent, is yet another variation of the sigmoid function and intents to solve the problem that the latter is not zero-centered. When compared to the sigmoid function, it solves just the problem of being zero-centered which means the vanishing gradient problem is still a reality. It is mathematically defined as: tanh(t) $= \frac{e^t - e^{-t}}{e^t + e^{-t}}$

- **ReLU:** the name stands for Rectified Linear Unit (ReLU), and the function is defined as: $f(t) = \max(0, t)$ and is a widely used activation function. Its wide acceptance comes from the fact that it is easy to compute and yet does not saturate or cause the Vanishing Gradient Problem seen in the previous activation functions. However, they also present some downsides like the fact that they are not zero-centered and the possibility of suffering from an effect called “dying ReLU”. The “dying ReLU” effect happens because the output of this function is zero for all negative inputs, which might cause some nodes to die completely only outputting 0 and therefore not learning.

- **Leaky ReLU:** is an improvement over the standard ReLU activation function. It keeps all good properties of ReLU and at the same time solves the issue with the dying ReLU problem. Mathematically, the Leaky ReLU function is defined as $f(t) = \max(\alpha t, t)$, where the hyperparameter α defines how much the function leaks, or in other words, the slope of the function on the negative domain. This slight slope ensures that Leaky ReLU never dies and that it does not suffer from the dying ReLU effect.

- **Parametric leaky Rectified Linear Unit (PReLU):** is a variation of Leaky ReLU function, where the variable α becomes a parameter that can be modified by backpropagation like the other parameters.

- **Exponential Linear Unit (ELU):** this is another variation of ReLU function that works by modifying the slope of the negative part of the function. Unlike the Leaky ReLU and PReLU functions, instead of making use of a straight line on the negative domain, the ELU function uses a log curve for the negative values. The main drawback seen when using this type of activation is that it is slower to compute than the ReLU and its variants discussed earlier.

Backpropagation refers to the backwards propagation of the gradients of the loss. To minimize the loss, we invoke an optimizer like Stochastic Gradient descent that iterates on the parameters in the direction of the steepest descent of the loss, thus minimizing it. The concept of Epoch represents the number of times the entire dataset is used to train the neural network. As a rule of thumb, multiple Epochs are needed; however, as the number of epochs goes up, the entire data is passed through the network more times, the weights of neurons are changed more often, and the learning curve goes from a situation of underfitting to overfitting. Finally, the batch size is another crucial tuning parameter representing the number of training examples present in each batch. This hyperparameter is fundamental since it is not feasible to pass the entire dataset into the neural net at once, so the dataset is divided into batches, i.e., subsets of the whole data.

Besides the items described above, adding a layer responsible for normalizing the data and keeping a 0 mean and standard deviation close to 1 before feeding it to the activation function is also beneficial. This layer must be added before the activation function to keep the data limited to an acceptable range and avoid very high numbers. Another standard customization method is to add a dropout layer, which is responsible for randomly assigning 0 to a share of the units in each layer. Units that have inputs different than 0 suffer a re-normalization with a factor of 1/(1 - dropout frequency). Thus, in the end, the sum over all units is kept the same.

2.2 Data Imbalance

When working on ML classification problems, it is common to face situations where there is a disparity between the number of elements in each one of the classes. This disparity can happen and is an important issue in both binary and multi-class classification problems. As an example of why this is relevant, we can think of a case of high data imbalance and a model that only outputs predictions belonging to the majority class. This model would have an overall performance inside acceptable values even though the classifier had no skill whatsoever. Since this is a common issue in many real-world problems, there are plenty of available approaches to tackle it. The approaches can be further divided into two levels regarding their focus: data and algorithm.

If we choose to fight data imbalance at the data level, one option is to, if possible, collect more data on the minority class. However, this may not be an option, so this approach is seldom taken. The next step is to understand how resampling the data, or in other words rebalancing the data classes, can solve the imbalance. The overall neutral disparity in terms of the number of elements in each class can be achieved by either making copies of instances belonging to the minority class or deleting instances from the over-represented class. The first method is called oversampling or sampling with replacement, and the second method is known as undersampling.

Undersampling the Majority Class

This method aims to remove random instances of the majority class to match the amount of the non-dominating class. This is usually a method to avoid since it requires some possibly valuable data to be

lost.

Oversampling the Minority Class

This method works in the opposite way of the previous one. The procedure is to choose random samples of the minority class and generate exact copies of those instances up to the point where there is a match between the under-represented class and the number of cases in the dominating class. This approach also presents some drawbacks because this algorithm essentially creates data duplicates, which might not be a good approach when the minority class is not evenly distributed across the feature span.

Synthetic Minority Oversampling Technique

This is another type of class oversampling, similar to the previous method in the sense that both work by generating samples. The difference is that Synthetic Minority Oversampling Technique (SMOTE) considers the characteristics of existing instances of the minority class in order to create new and synthetic new samples. The characteristics of the synthetic instances are interpolated from the features of the original data.

Bootstrap Sampling Method

This is another oversampling technique, but unlike the previous ones, the basic idea here does not pass by separating the data in classes. Instead, the bootstrap method estimates parameters about a population by averaging the parameters from multiple small data samples. It is a resampling algorithm that works by initially sampling with replacement to form n starting sets from the original data, all with the exact dimensions.

Gaussian mixture model (GMM)

Known to be a generative model, the GMM can be used to oversample the minority class instances present in an imbalanced dataset. The GMM method assumes that, within each feature, there are many sub-populations that follow their own normal distribution. The GMM algorithm works by modeling the data as a composition of multiple Gaussian probability distributions.

The GMM is mostly known for its clustering properties. However, it can also be seen as an algorithm capable of estimating densities and this makes it fall into the category of generative models. This means that, after the GMM is fitted to some data, we obtain a generative probabilistic model of the data. In the end, the GMM is a modeling technique capable of generating synthetic data close to the distribution of the fitted data, i.e., the original data.

But before generating more synthetic instances, it is necessary to find the optimal number of clusters so that, in the end, the original data is correctly described by the generated clusters. That is one of parameters that needs to be chosen.

The other parameter to be chosen is the covariance matrix of the GMM, and it can take one out of the following four forms:

- **Spherical:** in this case, each of the components has its particular variance, which means that the shape of each cluster is spherical.
- **Diagonal:** in this case, each of the components has its particular diagonal covariance matrix, which means that the shape of each cluster is an ellipsoid.
- **Full:** in this case, each of the components has its covariance matrix, which means that the shape of each cluster is independent of the remaining.
- **Tied:** in this case, all components have the same covariance matrix, which means that the shape is the same in all of them.

To generate new samples, the first step is to select the relevant elements such as the number of Gaussian distributions (K), the mean (μ_k) and co-variance (Σ_k) of each of those distributions.

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \Sigma_k) \tag{2.6}$$

The model in (2.6) is known to be capable to approximate any distribution as long as the number of models, K, is large enough. [52]

2.2.1 Model Selection Criteria

When the modeller is faced with choosing between different options, some model selection criterium is needed to assist the modeller. The standard practice is to choose the model that shows the best performance on an unseen test dataset. Whenever this is not possible, an alternative approach involves using models that quantify the performance of the training and the complexity of the developed model. Examples of such statistical models include the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC).

Akaike Information Criterion

The AIC [53] is used as a model selection criterion and is defined by

$$\text{AIC} = -2\log(L) + 2K \tag{2.7}$$

where L is the likelihood and K the number of parameters. This is an evaluation criterion for the performance of the model being studied and whose parameters are estimated by the maximum likelihood method. It prioritizes the model performance over model complexity, which might result in the selection of more complex models. The lowest the AIC value, the better the model being considered is.

Bayesian Information Criterion

The BIC [54] is also used as a model selection criterion, just like in the case of the AIC. However, the BIC answer a different question from the AIC. In simpler terms, the BIC tries to find the actual model among the candidates, whereas the AIC tries to select the model that most adequately describes the problem. The BIC is defined by

$$\text{BIC} = -2\log(L) + 2K\log(N) \tag{2.8}$$

where L is the likelihood, K the number of parameters, and N the number of samples. The criterion penalizes the model complexity significantly, meaning that more complex models are less likely to be selected. With the BIC, the quantity calculated is naturally different from AIC, but they are related.

2.3 Data Preprocessing

Over the next Section, some techniques used to prepare the data for the modeling phase will be presented. In broad terms, they tend to address the overfitting issue and/or are required for the correct implementation of specific learning algorithms.

2.3.1 Feature Scaling

In ML, feature scaling tries to standardize the importance of data points across datasets of varying scales of features. There are two popular approaches to scaling. One of them is normalization (or MinMax scaling). In this method, the data within each feature is scaled to a fixed range, that defaults to the range from 0 to 1. This may lead to minor standard deviations and consequently repress the outliers. Another popular scaling mechanism is the standard scaling (or Z-score normalization). This method transforms the data so that at the end, each feature has a mean 0 and a standard deviation of 1. The standard scaling method does not have a bounding range so, even in the case the dataset have outliers, they will not be affected by standardization.

2.3.2 Categorical Encoding

Despite some machine learning algorithms having the capability of learning directly from categorical data with no data transformation required, many of them cannot operate on categorical data directly and require all input variables to be numeric.

Ordinal Encoding

If we consider n as the number of distinct labels for each feature, hereafter cardinality, this type of encoding method basically assign an integer value from 0 through $n-1$ to each one the distinct labels. As a downside, this type of encoding makes the model believe the data has some ordinal order and incorrectly infers relationships between labels belonging to the same feature. All in all, this encoding procedure has the potential to deteriorate the model predicting capacity.

One-Hot Encoding

The downside of using ordinal encoding can be overcome using the one-hot encoding method. This method splits each column that contains categorical variables into multiple binary columns, one for each unique label category on the original column. For each new column, the values are replaced by 1s or 0s whether the observations have the column-represented value or not, respectively. This method may produce many columns if the cardinality is high and significantly slow down the learning process. There is also the possibility of omitting one of the newly formed binary columns, and in that case, there would be necessary $n - 1$ columns to encode a feature with cardinality n.

Target Encoding

For each category of value, the new feature value depends on the target's mean value computed using the target values of all instances that present the specific category and in the case of a binary classification problem, the target's mean value also corresponds to the probability. Like ordinal encoding, target encoding does not increase the data volume, which is particularly helpful for faster learning and dealing with large datasets.

Leave One Out Encoding

The Leave One Out technique uses a similar approach as the one seen in the target encoding technique but is less sensitive to outliers. This method achieves this by ignoring the target of the instance to which it is calculating the target mean. This algorithm proves to be particularly useful for high cardinality categorical features.

2.3.3 Imputation

Missing values can occur in the field of ML and their existence might compromise the results. The origin of such missing values can be diverse, and the missing values can potentially affect the performance of the algorithms. Since most ML algorithms do not accept missing values as inputs, it is crucial to solve their occurrences shall they exist. To do so, one has to choose whether to impute or remove the data, i.e., fix the missing value and update it to the correct value or completely ignore the instances where missing values occur.

Types of Missing Data

Depending on the reason behind the missed data, the occurrences can be classified into different categories of missing data. Each of these types has different constraints on how they should be handled and the overall approach that should be followed, either to impute or ignore them.

- **Missing Completely at Random (MCAR):** When the data is missed entirely at random, and there is no relationship between the the fact that the data is missing and any values observed or missing. When such missing data issues are faced, the instances where missing data is observed

can be safely removed, and the analysis proceeds using only observations that have the complete information.

- **Missing at Random (MAR):** In this case, there is a relationship between some of the observed data and the tendency for a data point to be missing some feature(s). In these cases, it is assumed a probabilistic relationship between the data that is missing and data that is not, which means that data MAR can, in some cases, be predicted from other features not missing.

- **Missing not at Random (MNAR):** In this category of missing data, there is a relationship between the tendency of a value to be missing and the value that is believed to be true, i.e., its hypothetical value. In such cases, the fact that the data is missing is related to external factors. Occurrences of MNAR should be addressed because the missing data mechanism itself has to and can be modelled, and some reverse engineering can be used to recover the missing values.

In the case of MCAR, the standard practice is to remove the instances that include missing values. In the second case, MAR, instances with missing values can be removed or imputed. Furthermore, in the third case, instances MNAR, the standard procedure is to work backwards and impute the missing data.

2.4 Classification Evaluation Metrics

In classification problems, there are plenty of metrics capable of assessing a model's performance. In this section, different metrics will be presented.

The most common evaluation metric for classification problems is the confusion matrix. This matrix assesses the performance for ML classification problems whose output can have either two or multiple classes. It corresponds to a table that has four different combinations of predicted and actual values in the case of a binary classification problem.

The interpretation of the different combinations is relatively easy, and it is as follows:

- **True Positive (TP):** the predicted value is the positive class and the actual class is positive.

- **True Negative (TN):** the predicted value is the negative class and the actual class is negative.

- **False Positive (FP):** the predicted value is the positive class and the actual class is negative.

- **False Negative (FN):** the predicted value is the negative class and the actual class is positive.

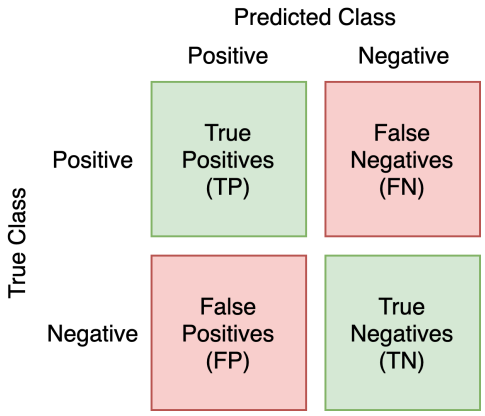


Figure 2.4: Confusion matrix for a binary classification problem.

The data extracted from the table depicted in Figure 2.4 will prove to be extremely useful when computing other types of metrics, such as Accuracy, Recall, Precision, and most importantly, Receiver Operating Characteristic (ROC) and Precision-Recall (PR) curves. A common way to access the quality of the predictions is to compute the accuracy that, in simple terms, measures the number of correctly predicted values among all the dataset. Although commonly used, this metric does not consider class imbalance and might produce a biased interpretation in such cases.

Some alternative metrics may be used to overcome the limitation of accuracy in dealing with imbalanced data. These include Precision, Recall, and F1-score. Precision measures how many labels are positive (TP) from all instances predicted as positive (TP plus FP). Recall, which is also called sensitivity or True Positive Rate (TPR), measures how many positive labels were predicted correctly (TP) from all positive class instances (TP and FN). Moreover, the F1-score assesses the results both in terms of Recall and Precision and uses a harmonic mean that penalizes extreme values. Table 2.1 summarizes all metrics presented:

Table 2.1: Classification metrics description.

Metric	Computation
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1-score	$2 \times \frac{precision \cdot recall}{precision+recall}$

The Confusion Matrix and the metrics derived from it presented good insights to evaluate the performance on balanced datasets. However, when we are dealing with imbalanced datasets, some other evaluation metrics may be needed to reason the trade-offs between the metrics mentioned above. This is where ROC and PR curves come into existence.

The ROC curve is a plot of the pairs True Positive Rate (TPR)/False Positive Rate (FPR) on the y-axis and x-axis, respectively. These values are plotted for every possible classification threshold. Both the TPR and the FPR are contained in the range from 0 to 1.

To generate the ROC curve, the procedure is to plot the pairs of TPR and FPR for all possible classification thresholds. In the case of a straightforward binary classification task, the thresholds range from 0 to 1. This is a significant benefit of using the ROC curve as a metric instead of more superficial metrics to evaluate a classifier, such as accuracy, since the ROC curve visualizes all possible classification thresholds. In contrast, accuracy and the other metrics previously presented, assess the classification for a single threshold which defaults, for binary classification problems, to 0.5, so anything below this is considered to belong to the negative class and anything above the threshold is considered to belong to the positive class. It is important to note that it is impossible to see the thresholds used to generate the ROC curve anywhere on the plot. In terms of performance, a classifier that can correctly separate the classes will present a ROC curve that touches the upper left corner of the plot. At this point, FPR equals 0, and TPR equals 1 that corresponds to a perfect classification.

Other relevant points include the upper right corner that corresponds to both FPR and TPR equal to 1, which means that the model only predicts the positive class; and the lower-left corner that corresponds to both the FPR and the TPR equal to 0, which means that the model only predicts the negative class. Conversely, the plot of the ROC gets closer to the diagonal of the graph as the algorithm becomes a worse classifier. The worse case scenario is when the algorithm is only making random guesses for the predictions and in this case the plot corresponds to the diagonal of the graph.

Naturally, there are ways to compare different classifier performances using ROC plots, and for that purpose, the Area Under the Curve (AUC) is used. AUC corresponds to the area of the graph under the curve and can be though of as the probability of ranking a positive instance higher than a negative instance when both are randomly selected. There is no rule of thumb to extract information solely from the ROC-AUC value, but very poor classifiers have an AUC around 0.5, and excellent classifiers have an AUC close to 1.

The PR curve is yet another helpful measure of the success of the model predictions when the classes are very imbalanced. The procedure is the same as in the ROC curve, except the metrics pairs are different. The PR curve plots multiple pairs of precision/recall points for different thresholds. As in the case of the ROC-AUC, the higher the area under the curve plotted, the better is the classifier performance. In the case of a high area, both high recall and high precision are seen.

High precision is usually associated with a low FPR, and high recall corresponds to a low False Negative Rate (FNR). In this case, the best classifiers in terms of performance will tend to touch the upper right corner of the graph and contrarily to the ROC, a random estimator would not have a specific AUC but instead would have a PR-AUC corresponding to the same proportion of positive cases, i.e., 10% positive outcomes would mean a 0.10 PR-AUC for a random estimator.

With interpretation issues in mind, we can also use the Average Precision (AP) metric to summarize the PR plot. The AP is the weighted mean of all the precision values achieved for the different thresholds. The gain in recall in going from a threshold to the following is used as the weight, $AP = \sum_n (R_n - R_{n-1})P_n$, where n refers to the nth threshold.

2.5 Shapley Additive Explanations

As ML algorithms evolve and become more advanced, they are able to produce more accurate predictions. However, at the same time, the output is getting less interpretable and intelligible. SHapley Additive exPlanations (SHAP) [55] is a method based on cooperative game theory introduced by Shapley that explains individual model predictions. The cooperative game theory assumes that the primary decision-makers might not only be single units but also groups of players, called coalitions, that may cooperate towards a common goal. The objective is to interpret the prediction of a given instance by assessing what was the contribution of each individual feature for the final prediction.

There are some benefits when using the SHAP approach. The first one is that it allows for global interpretability. SHAP values indicate the degree of contribution of each one of the features towards the target, either in the positive direction or in the negative direction. The second benefit of using SHAP is that it allows for local interpretability since the process done for all instances can be done explicitly for each occurrence, and each observation gets its own SHAP values. This increases model clarity since a unique and personalized explanation will be available to all instances. Another benefit is that SHAP is compatible with all tree-based models, while traditional methods do not always allow for this.

Chapter 3

Data Preparation

3.1 Exploratory Data Analysis

For the construction of the models developed in this work, we used data provided by TAP Air Portugal referring to the period between January 2019 and February 2020 encompassing 14 months. The data came further divided into three different datasets:

1. Serviço de Estrangeiros e Fronteiras;
2. Hub dataset;
3. Passenger dataset.

All datasets contained distinct, however, complementary data. Serviço de Estrangeiros e Fronteiras, known as SEF, is the force responsible for the control of the border in Portugal and the first dataset was a simple dataset with 8815 rows, three columns, and the information available on the dataset stated whether the connecting passenger should go through the immigration office or not, depicting all possible combinations between departure and arrival airports. Then the hub dataset compiled information regarding 109 attributes of all flight movements (both arrivals and departures), encompassing 345035 entries. These attributes included the date and time of arrival/departure, gates used, aircraft type, flight number, etc. Finally, the passengers' dataset included data from all 5,034,222 connecting passengers with at least one leg of their flight operated by TAP Air Portugal. This dataset included, in total, 21 features and contained information such as arrival and departure flight numbers, time and date, gender, age, state of the connection, among others.

Complexity-wise, the most manageable set to analyze was the one concerning SEF, given its small size and simple information. The other two were more difficult to analyze due to their volume and more complex information. As the final goal is to have a single dataset containing all relevant information with predictive power to build a model, we performed a preliminary exploration before merging all information from the three datasets since it was necessary to gain some insight about the data, such as to discover anomalies, find patterns and test hypotheses. The idea was to acquire the knowledge about the data needed to build the foundations for the modeling task ahead.

The approach chosen here was to join the Exploratory Data Analysis (EDA) with the data cleaning process due to this being the most pragmatic approach.

3.1.1 Pax dataset

As stated before, this dataset initially included 5,034,222 records and 21 attributes. The first attempt to clean the data was to eliminate duplicate entries, which reduced the number of rows to 5,034,214. Then a test was made, and it was confirmed that no missing values existed among the information. The following subsections summarize the main attributes of this dataset.

IDs

Each passenger has its identification number, which can indeed be present more than once in the data as long as the passenger has taken more than one flight. In total, 2,780,085 different IDs were present, meaning that each passenger has been in 2 connections on average. This is just a simple assumption that does not correspond to the truth since some passengers only took one connecting flight and others took well above 1.

Dates

The dataset also presented information regarding both the departure and arrival dates. The majority of the connections had same-day arrival and departure, but it could also happen that passengers needed to spend the night waiting for their connecting flight.

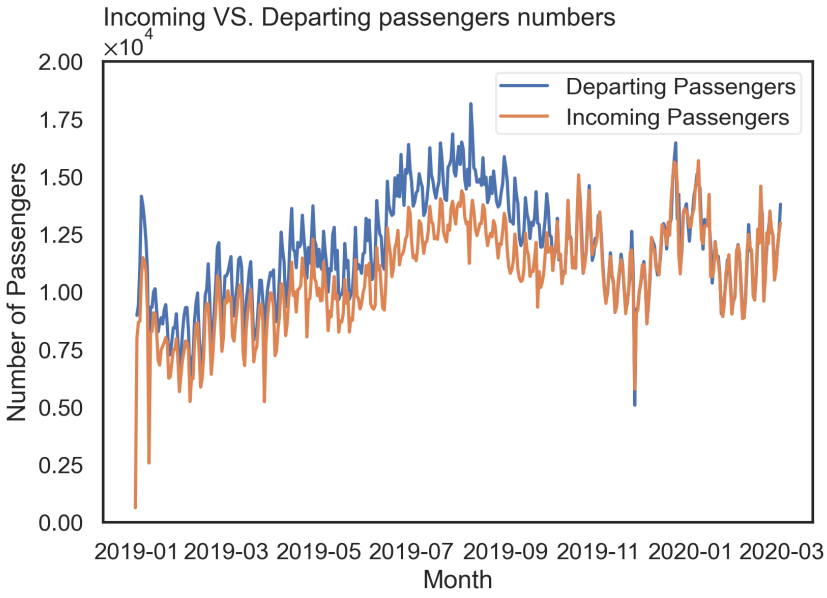


Figure 3.1: Difference between incoming and departing passengers throughout the 14 months. The separation between the number of incoming and departing passengers in each day of the year represents the number of passengers that missed the connection or had an overnight layover.

To begin with, one noticed that the feature corresponding to the date of departure has a #425 cardinality, whereas the feature corresponding to the date of arrival contained #427 different entries. This slight difference is because passengers leaving on January 1st 2019, could have arrived on December 31st 2018, and also to the fact that when a passenger misses his connection, the system, in some conditions not disclosed, adds '9999-12-31' has the default arrival date. This represents a problem that will be addressed later on. The two lines in Figure 3.1 should follow the same pattern. However, the number of arrival passengers is below the number of departing passengers for most of the year because the default arrival date when a passenger missed their connection is '9999-12-31' and therefore is not accountable. Therefore, the difference between the lines represents the number of passengers that missed the connection or had an overnight layover.

When thinking about the year as a whole, the day with the most passengers on transit was August 9th 2019 and other significant local *maxima* occurred around the holidays both at the beginning of 2019 and later that year during Christmas/New Year. During winter is where the minimum and local minima are also observed.

If one moves the analysis to the month instead of the day, it is visible that August 2019, followed by July 2019, saw the highest number of passengers among all months considered, whereas February 2019, followed by January 2019, saw the lowest number of passengers as shown in the Figure 3.2.

Finally, when looking at the day of the week, the highest number of departing passengers is observed on Saturdays, while the lowest numbers are seen on Wednesdays and Tuesdays, as depicted in Figure 3.3.

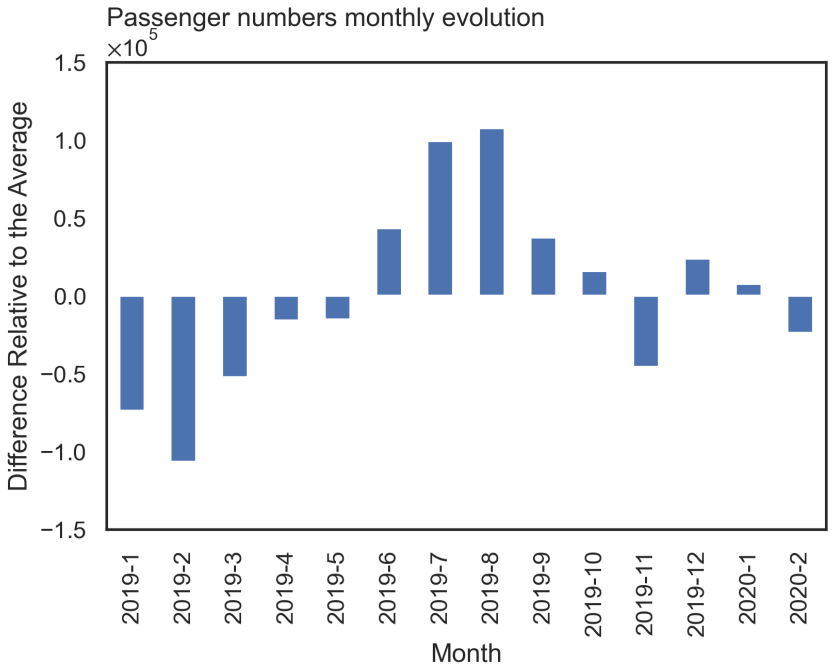


Figure 3.2: Distribution of passengers among the different months. In terms of passenger volume, August and July saw the highest passenger traffic while February 2019 saw the least connecting passengers.

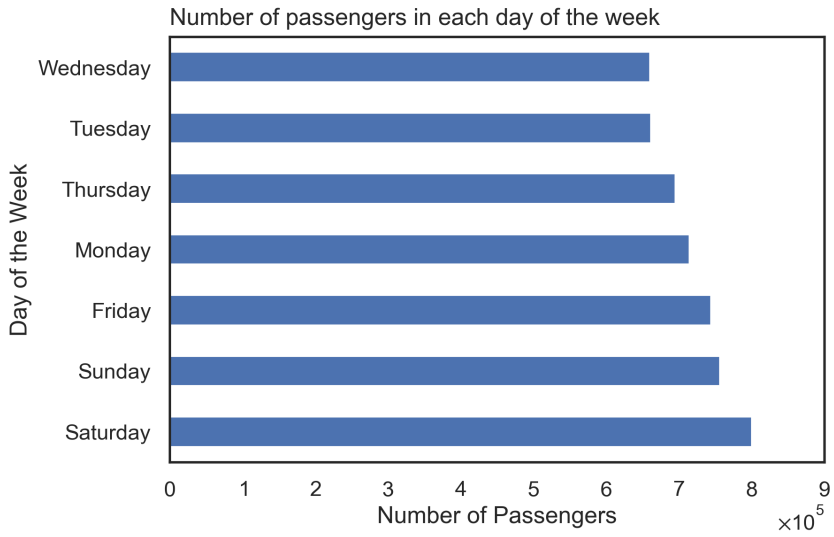


Figure 3.3: Distribution of the connections among days of the week. In terms of passenger volume, the cumulative number of passengers was the highest on Saturdays and the least on Wednesdays.

Arrival flights

Moving forward with the analysis, other valuable information comes from the origin airport, the flight code, and the class in which the passenger was travelling.

128 possible different airports had flights into Lisbon. From that, Porto (OPO), Paris- Orly (ORY), Funchal-Madeira (FNC), Sao Paulo/Guarulhos (GRU), and Faro (FAO) ranked the top 5 respectively.

Furthermore, the airline divides the passengers into 16 different travelling classes; however, some of those are just slight variations from one another and can, in the end, be considered the same class type. The complete list of classes is available in Figure 3.4. This figure also shows that the feature had 527,800 missing values, which corresponds to approximately 10.48% of the volume of data.

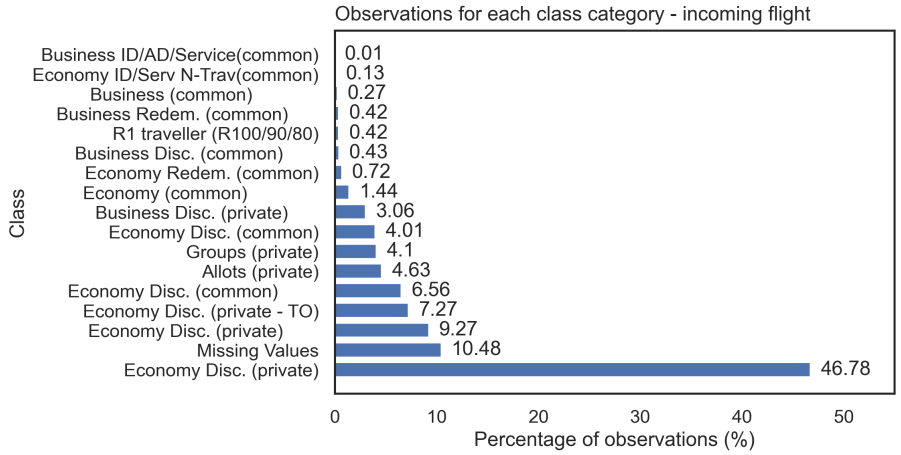


Figure 3.4: Distribution of the travelling classes on the incoming flights. The majority of connecting passengers transited while travelling in economy class products and the least amount of passengers were travelling in business class.

There are, in broad terms, five different types of classes: Economy, Allots, Groups, R1 and Business, and by far the most common travelling class was the Economy.

The cardinality of the feature corresponding to the flight number is high and stands at #642. Here not all flights are operated by TAP but some partner airlines instead. Most of the passengers arrived in TAP-operated flights, marked with 'TP', but there was a total of 59 different arrival airlines, with TAP transporting 97.6% of the total inbound passengers.

Departure Flights

A similar analysis can be done regarding the departing flights.

A similar analysis can be done regarding the departing flights. This time 110 possible different airports had flights scheduled from Lisbon. From that, Porto (OPO), Paris-Orly (ORY), Funchal-Madeira (FNC), Sao Paulo/Guarulhos (GRU) and Faro (FAO) ranked, respectively, the top 5. Although the order is the same, the absolute number of passengers travelling to each of these destinations was higher than the corresponding number of incoming passengers from the same airports.

Again, 16 different travelling classes were set by the airline as possible travelling classes; however, in broad terms, there are five different types of classes: Economy, Allots, Groups, R1 and Business; the same as defined earlier. There were no missing values detected in this feature. The complete list of classes is available in Figure Figure 3.5.

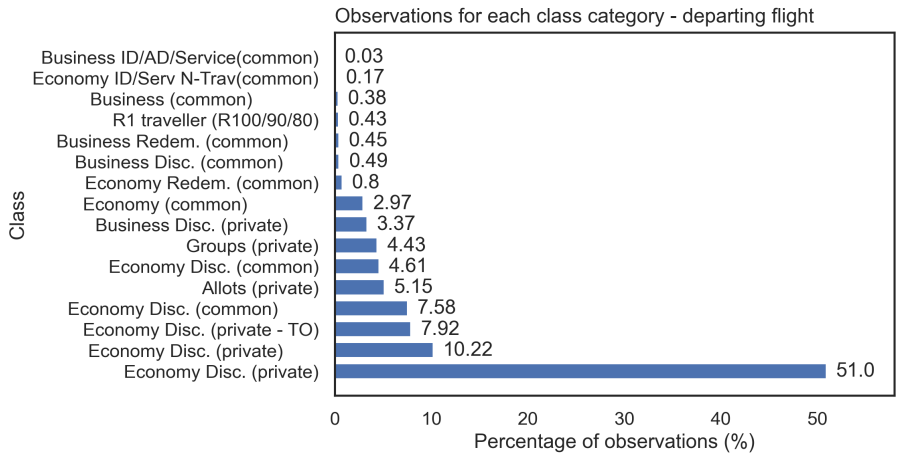


Figure 3.5: Distribution of the travelling classes on the departing flights. The majority of connecting passengers transited while travelling in economy class products and the least amount of passengers were travelling in business class.

Again, as in the case of arrival flights, the Economy class is the most common among the connecting passengers.

The cardinality of the feature corresponding to the departure flight number is still high but lower than in the arrival case and stands at #344. Here, all flights were operated by TAP.

3.1.2 Other Features

The remaining seven attributes are binary because they can only assume one out of 2 possibilities.

Is Group

This feature states whether the passenger was travelling on its own or within a group. This information was provided directly from TAP, and it was extracted from the ticketing system that indicates whether a specific reservation was made for a single person or multiple persons. The 'Is group' variable assumes 0 when the passenger was travelling alone and 1 otherwise. From the 5,034,214 total entries, 2,991,162 of them, or 59.4%, were travelling in a group. The remaining ones correspond to passengers travelling alone. All rows had valid data, and no missing data was detected.

Age

This feature is self-explanatory, and from the 5,034,214 total entries, 4,799,703 or 95.3% corresponded to adult passengers. The remaining 234519 entries are infants or children.

Gender

Just like the previous attribute, the 'Gender' feature is also self-explanatory. From all binary features, this was the only one that presented missing data. Here the distribution was as follows: 2,361,199 entries were Female, 2,639,064 were Male, and 33959 were missing. This translates into a 46.9%, 52.4% and 0.7% distribution, respectively.

Check bags

When travelling, passengers, depending on other factors, can either carry their luggage into the airplane cabin or handle it to the airline's responsibility. The 'Check bags' feature indicates whether the passenger chooses to check its bags with the airline, in which case the variable assumes the value 1 or 0 otherwise. From all entries, 3,433,408 or 68.2% corresponded to passengers that did not check their bags. The remaining 1,600,814 entries, or 31.8%, corresponded to passengers that checked their bags.

PAX Boarding

PAX Boarding is a binary variable that assumes 1 if the passenger has successfully boarded the flight in question and 0 otherwise. At this point, it is important to mention that when a passenger misses a connection and is allocated to another flight, its connection will appear duplicated among the data the same number of times as the passenger was reallocated. In cases when a reallocation occurs, the information regarding the passenger ID and arrival flight will be the same but information regarding the departure flight, connection status, and boarding status will be different. The data shows that 86.2% of the instances were marked as successfully boarded, leaving 693,933 total entries with 'Pax Boarding' set to 0.

Connection Status

This variable is highly correlated with the previous one, at a correlation factor of 0.82, as seen in Figure 3.6 that shows the correlation between these two variables and among all numeric features. The data analysis shows that 80.6% of the entries were marked as a successful connection while 974,845 (or 19.4%) were marked as unsuccessful. No matter what happens after the first missed flight, the value of this variable will always be 0, even on the sample where the original passenger boarded the new outbound flight.

Overnight

Finally, the 'Overnight' feature indicates when the missed connection implies that the airline must provide a place for the passenger to spend the night. These cases represent only a tiny portion, meaning that TAP only had to provide overnight accommodation to 74,787 passengers (or 1.49% of the total samples).

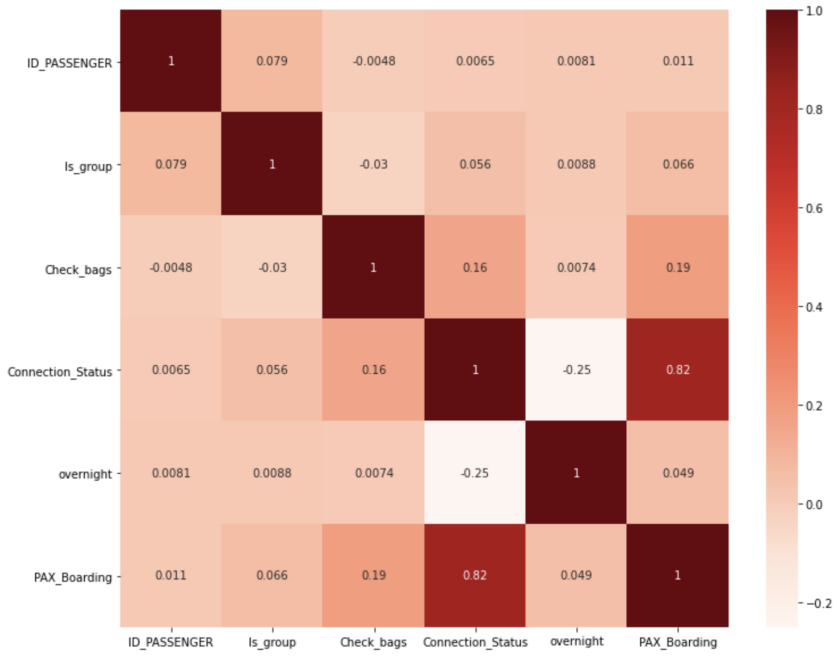


Figure 3.6: Correlation factor between all numeric features within the Pax Dataset.

3.1.3 Flight dataset

Moving on to the other dataset, it initially included 345,035 records and 108 attributes. Again, and as was performed in the PAX dataset, the first attempt to clean the data was to eliminate duplicate entries, which reduced the number of rows to 332,727. Then a test was made, and it was confirmed that several features presented missing values. At this early stage, not much concern was put into this issue since it was still necessary to check how relevant the information contained in those features was to the model.

Among all flights registered in the dataset, 167,695 (or 50.4%) were arrivals, and 165,032 (or 49.6%)

were departures. In total, and within the 108 features, 18 are related with a timestamp associated with the respective flight. The reason for this redundancy is that three different systems are in charge of collecting the same variables: the airport authority itself (APT), Aircraft Communications Addressing and Reporting System (ACARS) and finally, the Operations Control Center (OCC).

In total, there were three different types of timestamps for arrivals: Estimated Arrival Time, Landing Time, and On-Blocks Time; and another three types for departures: Estimated Arrival Time, Take-Off Time, and Off-Blocks Time. All three systems recorded these values for all the flights except for the Estimated Arrival Time to which there is no data from the ACARS system, which makes up for the 17 (out of 18) different features associated with timestamps.

The 'Estimated Departure Time' is an attribute that indicates the estimated time for the beginning of the aircraft movements associated with the departure stage. Both OCC and APT make this estimate (as stated earlier, the ACARS does not register this variable), and the information is shown to the passengers on the terminal information system to notify them about the flight status. The idea behind the 'Estimated Arrival Time' is the same but for the opposite movement, the arrivals. It represents the expected time for the arrival at the parking position, and all three systems measure it.

The 'On/Off-Blocks Time' details when the aircraft parks/pushes back the parking position, respectively. APT, ACARS and OCC estimate these two timestamps. The 'Landing/Take-Off Time' details the date and time at which an aircraft has landed/taken off from the runway, respectively. Again, APT, ACARS, and OCC estimate this feature.

Since the timing of the aircraft movements has a lot of relevance within the scope of the proposed study a deeper analysis was performed to these values. To begin with, it was studied what was the percentage of missing data within each of the variables in question. The results were very heterogeneous, but in general, the data provided by the airport authority had the least amount of missing data, as one can see in figures 3.7 and 3.8 for the arrival and departure features, respectively.

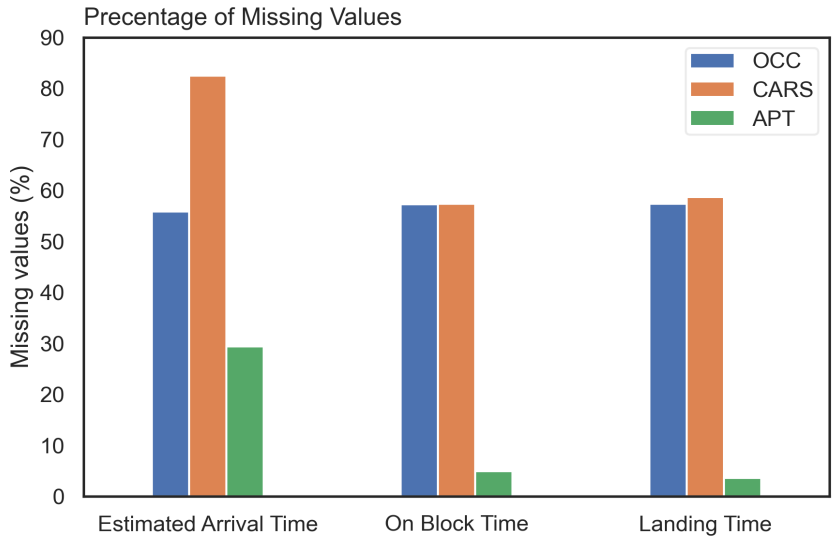


Figure 3.7: Distribution of missing timestamps among the arrival flights. Representing the three different timestamps measured by the three different systems.

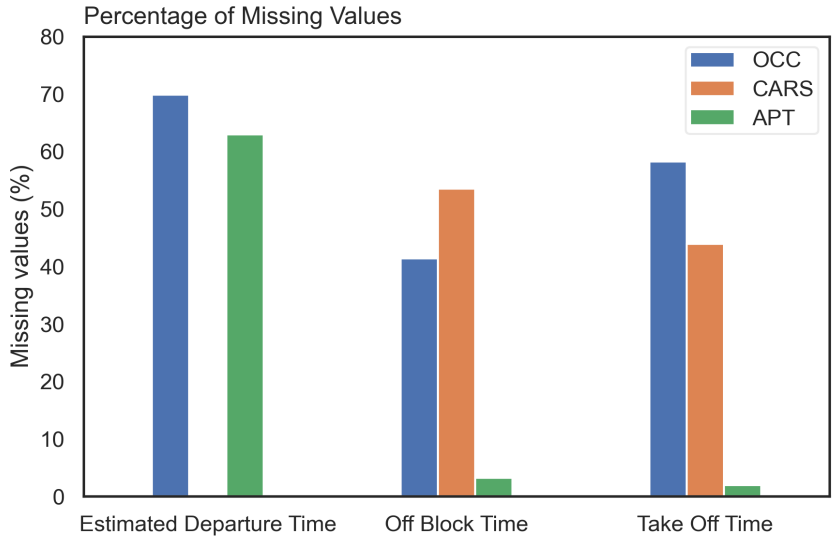


Figure 3.8: Distribution of missing timestamps among the departure flights. Representing the three different timestamps measured by the three different systems, except for the Estimated Departure Time since the CARS does not record this timestamp.

The 18th timestamp is known as 'Best time Hub Control' and records information for both inbound and outbound flights with no missing values. This attribute has the same information as the 'On-Block Time' or 'Off-Block Time' measured by the OCC depending if it is an arrival or departure flight, respectively.

Transport between the Terminal and the Aircraft

There are, in broad terms, two ways of commuting between the terminal building and the aircraft. The first and more common way of doing it with TAP departure flights is for passengers to commute by walking to the airplane, generally using jet bridges - structures that link the airport terminal and the airplane. The second way is to use buses to transport passengers from the terminal to where the airplane is parked. Regarding arrival flights, the situation is the opposite, and most of the passengers are transported to the terminal using buses. All passengers arriving on flights that require bus transportation are directed to one of 2 gates, depending on if they come from a Schengen or Non-Schengen flight. Non-Schengen airports are located outside the Schengen area, which is a supra-national agreement that includes most of the European Union plus some bordering countries.

To understand the details of the passengers boarding and unboarding of the aircraft, it was necessary to perform a thorough analysis of different attributes from the dataset. First and foremost, it was detected that the dataset shows what platform and gate were used by the specific flight movement. However, all arrivals have their gate set to a default value and are therefore considered missing. Among the departure flights, we found only 14,268 missing values (or 8.6% of the total) on the gate used for the departure. The data also contained several columns dedicated to registering the different timestamps associated with the loading start and finish of the buses. For each of the buses used, it was recorded the time when the first and the last passengers boarded the bus. Therefore, if no buses were used, one could

assume that the specific flight movement did not require any bus, and if only the columns referring to the first bus have non-default values, it means that only one bus was used, and so on. Again, the data regarding arrivals was disrupted, and no information could be extracted, but regarding the departures, no information was missing, and the distribution in terms of the number of buses is as follows in Figure 3.9.

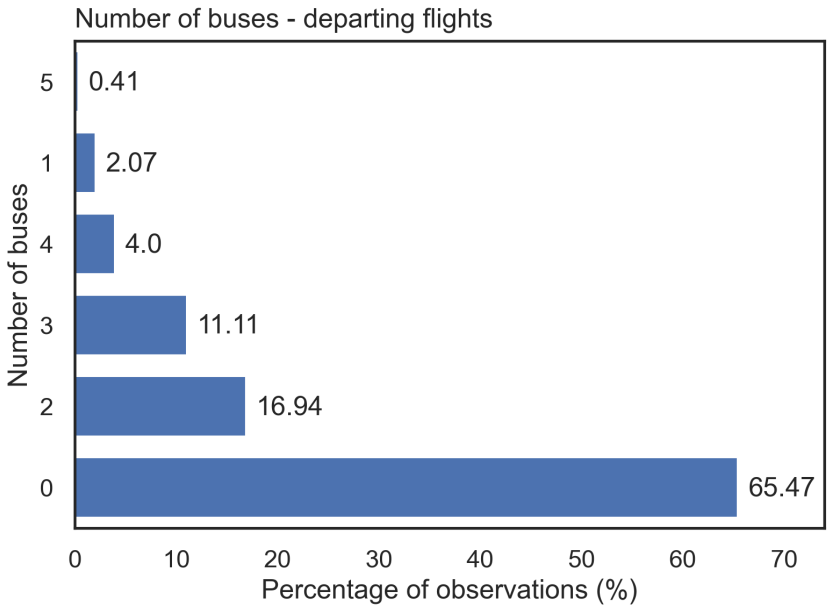


Figure 3.9: Distribution of bus usage among the departure flights. Most of the airplane boarding processes did not use any bus while only 0-41% of flights used 5 buses.

The most common way to board the aircraft is indeed to used the jet bridge and most of the flights that required bus-assisted boarding only needed two or three buses.

3.2 Data Enhancement

The goal of this step was to merge the three existing datasets into one that had the relevant information to feed good predictive models.

3.2.1 Data Cleaning

During the previous steps, only an initial and simple cleaning was performed, but the extent of the scope of this thesis required a more detailed cleaning.

Pax dataset

As stated before, the issue created by the mechanism that uses the default '9999-12-31' as the arrival date when passengers miss the connection needed to be addressed since this value is present in 467,969 rows, which is equivalent to 9.3% of the data, therefore disregarding this data was not a valid

option. It was also noticed that within all rows that had this error, 467,758 of them (or 99.95 %) were also missing the attribute indicating the travelling class on which the passenger arrived.

However, as disclosed before, this error is systematic and is related to the rescheduling of flights when passengers miss their original connection, so in theory is possible to retrieve some, if not all missing information. This issue represents the MNAR type of missing data presented in Section 2.3.3. One outlined a strategy to attempt to retrieve information that consisted of checking all the registered incoming flights for each distinct passenger ID.

Whenever some missing information was detected, it would be replaced by the correct information (in this particular case, the date and class of the arrival flight) contained within the row that presented the attribute passenger boarded equal to 1. Here is essential to notice that whenever a passenger has multiple incoming flights with the same code, it can represent multiple connections, and to account for this, it was set that the difference should be smaller than one day to be considered part of the same connection.

Hub dataset

In terms of attributes, this was the dataset with the most data. However, when accounting for the quality of the attributes and their usefulness, there was no necessity to perform a thorough cleansing of all 108 attributes because a high number of them had plenty of missing data or were not helpful for the goal model. Examples of such features were: the ones related to the cleaning and catering timestamps of the aircraft, the ones related to the crew, among others.

3.3 Creation of New Features

Based on all data provided, we did some feature engineering to extract the most valuable information from all data while keeping the total number of features as low as possible. This process is helpful because it allows for capturing information that was not explicit in the original data.

The new variables created were:

- **Scheduled Connection Time:** The idea behind this engineered feature was to record the time, in minutes, that the airline initially planned for the connection without considering any disruption or operation problems, i.e., the time interval between the schedule on-block time and the schedule off-block time. We extracted the information for this new feature from the 'Best time Hub Control' timestamps records.

- **Traffic Network:** There are two possible types of origin airports and two types of destination airports when it comes to the applicable legislation to transiting passengers. The outcome from these conditions is that there are in total four types of traffic passengers that can happen when one is dealing with connection flights: Schengen to Schengen (SS), Non-Schengen to Schengen (NS), Schengen to Non-Schengen (SN) and Non-Schengen to Non-Schengen (NN). To assign each of the origin/destination pairs to the correct type of traffic network, one performed a cross-check

between a list of all Schengen airports with a list of all Non-Schengen airports. This feature can have an excellent predictive power because all four types demand different requirements from the passengers regarding customs duties and even security issues.

- **travelling Class:** As stated earlier, the cardinality of the travelling class feature (both for arrivals and departures) was 16, which is a relatively high number. For that reason and because, as stated before, there are only five different types of classes in broad terms, the 16 classes were grouped in different categories: Economy, Allots, Groups, R1, and Business.

- **Label:** Depending on the connection status, several labels were created in an attempt to cover all possible scenarios:

  - **A:** This represents the scenario where the passenger can have a successful connection as it has been initially planned by the airline. The A label accounts for the majority of the connecting passengers of TAP.

  - **B (or b):** In this case, the passenger did not have a successful connection as planned initially, but the airline reassigns the passenger to another flight. Since as stated in Section 3.1.2 there are as many rows as flights reallocation, the connection record that saw the passenger board the plane is marked with the label *B*, the remaining one(s) with *b*.

  - **O (or o):** These labels were also attributed to passengers in the same conditions as the previous ones except that, in this case, the airline had to pay for the passengers' overnight expenses. The connection record that saw the passenger board the plane is marked with *O*, the remaining one(s) with *o*.

  - **X:** This label was assigned to all cases on which some sort of incoherent data was detected. Examples of such situations are when the same passenger is reassigned to multiple outbound flights.

  - **N:** Finally, the label *N* was given to all connections on which the lack of data might indicate that the passenger did not show up on the flight.

3.4 Data Selection

After the feature engineering described in Section 3.3 one added the new features to others previously selected, adding up to 11 in total. These features include much valuable information primarily due to the immense information the engineered features include. The chosen features are the incoming and outgoing flight code, gender, age, type of traveller (solo or group), incoming and outgoing travelling class, day of the week (Sunday through Saturday), day of the month, Scheduled Connection Time and Traffic Network.

After the merge of all information in the same dataset, it included a lot of data. However, some of that data was out of the scope of the current work or was unusable because it contained a large proportion of missing values. To start, only rows with labels A, b, and o are within the scope of this thesis because

labels X and N include corrupted or unreliable information, and labels B and O refer to passengers that board a flight other than the initially scheduled connection flight and are therefore also out of scope.

After this data cleaning, the dataset contained 3,592,004 samples, and from those, 3,381,353 were labeled with A, 154,760 with b, and 55,891 with o. However, the data still presented some missing values on some of the features, and a deeper cleaning was performed. The distribution of the missing values among the 11 features is shown in Figure 3.10 and in relative terms 0.79% of Class_FROM, 0.60% of SchConnectionTime and 0.41% of Gender values are missing. The remaining features did not include any missing values.

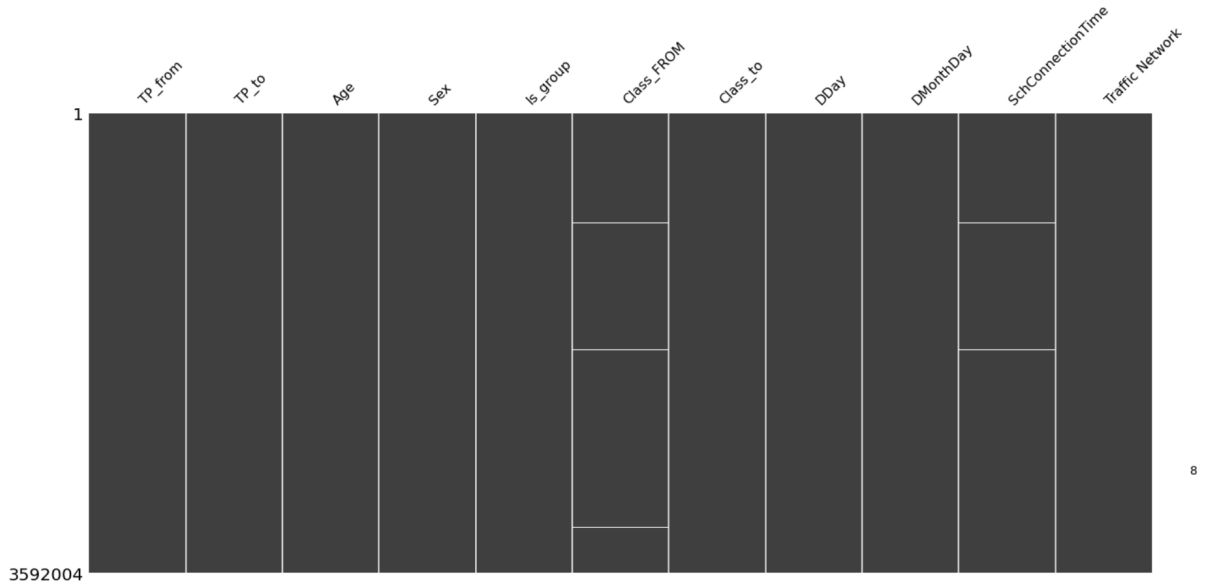


Figure 3.10: Distribution of Missing values across the features. White strips indicate the missing values in each of the features.[56]

Before moving to the next stage, one dropped all instances that included missing values in any of the features, and the dataset reduced the total size to 3,451,979 samples, including 3,261,690 A labels, 139,519 b and 50,770 o labels. Before the encoding phase, it was necessary to follow the best practices within these types of work and divide the data.

A stratified label-based train/test/validation split was performed and the resulting datasets corresponded to 80%, 10%, and 10% of the global data, respectively. In more concrete terms, the training dataset has 2,761,583 samples, and both the validation and test sets have 345,198 samples. In all four datasets discussed earlier, original clean dataset, train dataset, test dataset, and validation dataset, the proportion of samples was maintained and corresponded to around 94.49% of the samples belonging to the A label and the remaining 5.51%, including the b and o labels.

3.5 Data Transformation

3.5.1 Data Encoding

As stated in Section 2.3.2 there are plenty of methods to encode data for machine learning models. In this specific case, from the 11 features present, 3 of them, namely, Age, Gender, and Is_group are binary features and, in practice, are by default encoded using a one-hot-encoding technique. One out of the remaining eight features, the *SchConnectionTime*, is continuous and therefore does not require any encoding. The other seven features are non-binary categorical features, and all require some type of encoding.

As the general practice within this domain area, only the data in the training dataset was encoded, and afterwards, the same encoding information was applied to the test and validation datasets. After studying several scenarios to encode the seven features, one realized that the incoming and outgoing flight codes required an encoding solution that did not add any extra columns to the dataset since these features have a very high cardinality, standing at #332 in the case of *TP_from* and #311 in the case of *TP_to*.

The solution found for the *TP_FROM* and *TP_to* features was to encode them with target encoding. For the remaining five features, the cardinality was lower in all of them, ranging from #4 in the *Traffic Network*, #5 in the travelling classes, #7 on the day of the week, and #31 in the day of the month. In those cases, the same type of encoding was used as in the *TP_FROM* and *TP_to* features because using ordinal encoding would artificially feed the model with a nonexisting ordinal order, i.e., there is no reason why an SN transit would have a higher value than an NS transit or to impose ordering to weekdays.

3.5.2 Data Scaling

To avoid overflow, all 11 features were scaled to mitigate the issue. The choice of scaling made was to use the Standard scaling technique so that all the features would have a mean of 0 and a standard deviation of 1. By using this technique, we assured that no outliers would be suppressed, which might be a good approach since some extreme behaviors may be associated with missed connections.

However, feature scaling was only used to the DNN, and Logistic Regression models because in Decision Tree classification the data is split based on scores which are calculated using the homogeneity of the resultant data points.

3.6 Data Generation

From all the possible methods to tackle the data imbalance issue discussed in Section 2.2, the choice was made so that in the end, no data would be lost, the amount of added noise would be the lowest possible, and the chances of overfitting would remain as low as possible.

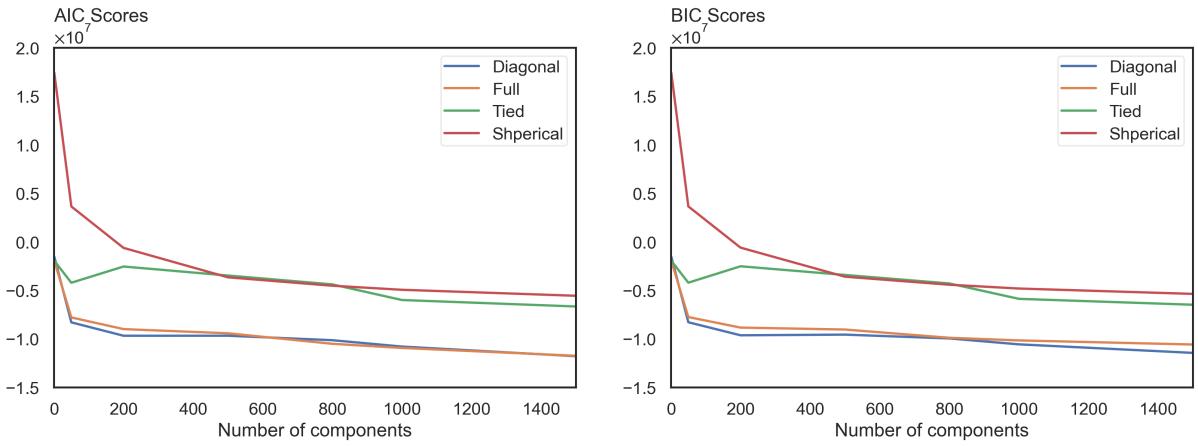
These criteria meant that the Undersampling technique would be immediately excluded because this would imply that the majority of the data of the majority class would be lost. Among the 2,761,583 training

samples, only 152,231 correspond to the minority class, which means that should this technique have been chosen, 2,457,121 samples from the majority class would have been lost. This would represent a significant loss, corresponding to approximately 90% of all data and therefore the Undersampling technique was not pursued any further.

The following available options were to use any of the Oversampling techniques presented in Section 2.2. Contrary to the Undersampling option, these options would not cause any loss of information but, in the case of the generic oversampling technique, would increase the likelihood of overfitting since the method only creates new copies of existing data and as stated earlier does not account for the fact that the minority class may not have an even distribution. The SMOTE method is know to be sensitive to outliers specially in high-dimensional data. Thus, the GMM was better suited for the generation of synthetic samples, and that is why we chose this method for the generation of new samples.

In order to define the parameters of the GMM that will be used, it was first necessary to select the type of covariance and the number of components used to train the model. To do so, and as explained before we used the AIC and BIC values that played an important role in the selection of the parameters. Figure 3.11 shows the results for all the available types of covariance shapes and a number of components ranging from 1 to 1500. Additionally, and as stated before, the criteria to select the number of components and the covariance shape is to search for the lowest values for AIC and/or BIC.

Looking at Figure 3.11 we observed that between the diagonal and full covariance shapes, the diagonal attains the better performance for all numbers of components tested in terms of the BIC score and has a lower or equal value to the full covariance shape in terms of the AIC score. This trend can be justified by the fact that BIC penalizes the model complexity more than AIC, which means that the diagonal covariance matrix returns a model less complex than the full shape. After a detailed analysis of the results, it was considered that the increase of components up to 1000 was worth the decrease in the AIC and BIC score values. After this point, the benefit of increasing the complexity of the model and consequently its cost was no longer worth the benefits in terms of AIC and BIC.



(a) AIC Score

(b) BIC score

Figure 3.11: AIC and BIC scores for the GMM parameters selection.

3.6.1 Data Re-balancing

After following the steps described in the previous sections and before feeding the training data to the models, an extra step was taken to achieve the best result from the data. The first stage was to re-balance the data and make the problem's classes balanced, and to do that, one divided the training dataset into two subsets, one corresponding to the majority class and another containing all samples labeled as belonging to the minority class. After this step, new artificial samples were generated from the minority class using the GMM technique as the oversampling mechanism, and the parameters founded earlier.

The number of newly created samples corresponds to the difference between the total number of samples of the majority class, 2,609,352, and the number of samples of the minority class, 152,231, which translates to 2,457,121 new artificial samples. The new samples were generated after fitting the GMM model to the data corresponding to the minority class, using 1000 components and a diagonal covariance matrix.

These new data points follow the distribution of the original data points, as can be seen in Figure 3.12 for the case of the Traffic Network attribute. However, some additional processing is necessary since the GMM generates continuous values, and some of the features present discrete values. To deal with this, the newly generated data points were grouped in a specific number of clusters using a K-means algorithm. The number of clusters in each feature corresponded to the cardinally of that feature in the original data.

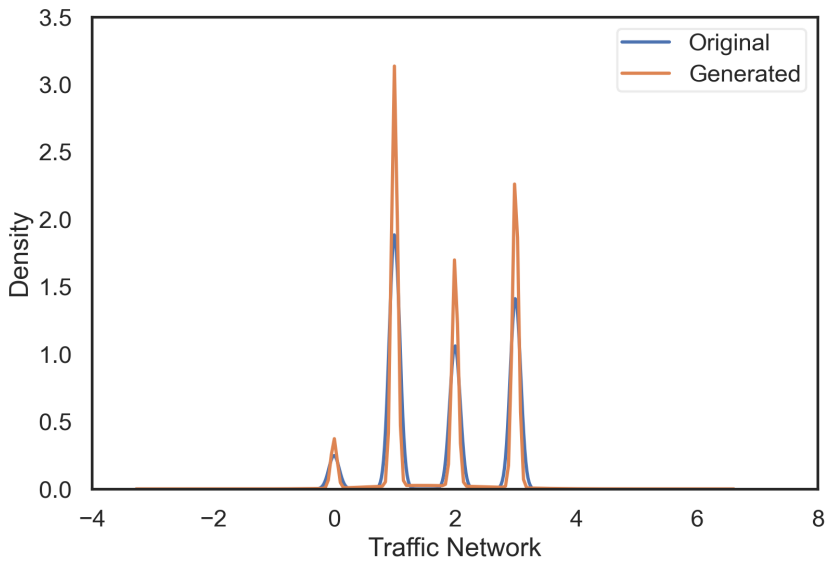


Figure 3.12: Fitting of the GMM over the traffic network feature.

One repeated this process to all features except for the ones concerning the flight numbers (both inbound and outbound). For these features, one chose another methodology since the K-means could have introduced new, previously unseen values with no real-world meaning, so one used the k-nearest neighbors (k-NN) algorithm to find the values for the two features that would make the most sense, knowing all other previously generated features.

After finishing all the clustering processes, we created a new dataset with 2,457,121 artificial samples all of which belonging to the minority class. This dataset was then merged with the original data (that included the majority class and the minority class elements used to generate the artificial samples), and a final dataset with 5,218,704 samples was reached. However, this dataset did not suffer from any data imbalance, and the data was evenly distributed among the two classes, meaning 2,609,352 samples belong to the majority class (successful connections) and the same number of 2,609,352 samples belong to the minority class (unsuccessful connections).

The final process to make the dataset ready to be fed to the model was to encode all the training data, both the original data points and the generated ones. This step followed the guidelines described in section 3.5.1.

Chapter 4

Modelling

The following chapter revolves around the experimental phase: the training and testing of the different predictive models, the assessment of their performance on unseen data, and finally, the results and the comparison of the experiments.

4.1 Model Baseline

As the baseline classifier, one considered the system used by the airline which assumes as the criterion to decide whether a passenger will successfully connect the Minimum Connecting Time (MCT) established by ANA - Aeroportos de Portugal, the airport authority of Portugal. This value corresponds to 60 minutes and means that the airline assumes that a passenger will miss the connection if the connection time is below 60 minutes.

The baseline model has an accuracy of 0.92, AUC_{ROC} equal to 0.61 and AUC_{PR} to 0.28. Table 4.1 summarizes the results from each one of the classes.

Table 4.1: Classification metrics of the Baseline algorithm for each of the classes.

	Precision	Recall	F1-score	Support
Class 0 (successful connection)	0.96	0.95	0.96	326169
Class 1 (unsuccessful connections)	0.25	0.27	0.26	19029
Micro average	0.92	0.92	0.92	345198
Macro average	0.61	0.61	0.61	345198
Weighted average	0.92	0.92	0.92	345198

Table 4.2 represents the confusion matrix of the Baseline model. The positive class corresponds to missing the connection and the negative class corresponds to have a successful connection.

Table 4.2: Normalized Confusion Matrix of the Baseline model.

		Predicted Class	
		Positive	Negative
True Class	Positive	27.46%	72.54%
	Negative	4.72%	95.28%

Tables 4.1 and 4.2 evidence a good performance on the majority class; however, the performance on the minority class is lower and the model miss-labeled around 72% of instances.

4.2 Model Proposed

The goal of the Decision-Making Models being designed is to ascertain whether a connecting passenger is likely to miss the second leg of the journey. With that in mind, the information available shall be the same as the information known by the airline after the flight schedule is made publicly available and the passengers booked their flight, but still before the flight date. The idea here is to create several models based on different machine learning algorithms and make predictions based on the input data that comes from the available information.

These models aim to understand how the airline can translate data about the passengers into valuable insights. The data includes the features already discussed in the previous chapter: incoming and outgoing flight numbers, the class on which the passenger is traveling, the week and month day of the flight, and some demographic features such as gender, sex, and whether the passenger is traveling in a group or not.

After carefully analyzing the models, the airline should be able to notice trends and clusters and understand what profile of passengers is at most risk of missing the connection and will do it knowing the relative importance of each feature for the final prediction.

4.3 XGBoost

4.3.1 Hyperparameter Tuning

The XGBoost algorithm presents some high-level parameters called hyperparameters that are the variables whose function is to determine the structure of the trees, e.g., the Depth of the tree and the variables that control the training, e.g., the Learning Rate. Since hyperparameters selection occurs before the final training process, it is necessary to find a way to efficiently compare the performance of different parameter settings, i.e., different hyperparameters configurations.

The process described above, designated model selection, demands extra care from the modeler to avoid overfitting. To avoid biases, we shall not use the test data in any part of the training process. The alternative is to either use the validation dataset, which is also unseen during training or use a k-fold cross-validation technique. In the latter, the training data is split, while maintaining class proportions,

into k different folds, and from that, k$-$1 are used to train the XGBoost model, and the remaining fold is used as a pseudo-validation set.

The described procedure is repeated as many times as there are folds to reach k different solutions. We chose the number of folds so that each fold had the same size as the testing and validation sets, which means that each fold corresponded to $1/8$ of the training dataset. Thus, the final data distribution among the pseudo-training, pseudo-validation, validation, and test sets is 70%, 10%, 10%, and 10%, respectively.

Each one of the predictions made is used to calculate a corresponding confusion matrix. Then all confusion matrices are added to yield a final confusion matrix, which is possible since the folds are created randomly without replacement. One repeated the process for all possible combinations of parameters, and the configuration that yields the best result becomes the chosen set of hyperparameters. This technique is known as grid search and is part of the XGBoost API, which facilitates its implementation.

Hyperparameter Space

The Gradient Boosting, Random Forest, and XGBoost methods are among the most popular tree-based ensemble algorithms for classification problems and the most effective at it. Their performance is substantially based on the depth of the trees they build and the learning rate. The XGBoost algorithm was the chosen algorithm on the tree-based ensemble category due to its versatility and also because this classifier presents system and algorithmic enhancements that increase the classification performance and training efficiency both in terms of memory and time that were presented in Section 2.1.3.

For the XGBoost algorithm, one set the hyperparameter space to a grid space spanning learning rates ranging from 0.3 up until 0.6 with steps of 0.1 and depths ranging from 30 to 50 with steps of 5. One chose this search space after a preliminary study suggested that this would not be a significant burden for the available computational tools at hand.

The grid search varied the above two parameters without limiting the number of estimators; however, one defined an early stopping criterion that interrupts the learning process if the results did not improve after a pre-determined number of boosting rounds. This parameter was set to 20. We used the validation dataset to evaluate the results and decide upon enforcing the early stopping criterion to the training process, and to do that the criterium chosen was the ROC-AUC.

Tables 4.3 to 4.6 show the results of the hyperparameter optimization process. Each one of the tables is dedicated to a different metric evaluated during the training process.

Table 4.3: Tuning the hyperparameters of the XGBoost classification model with cross-validation based on the Accuracy.

Accuracy		Tree Depth				
		30	35	40	45	50
Learning Rate	0.3	0.9325	0.9325	0.9326	0.9333	0.9330
	0.4	0.9332	0.9330	0.9328	0.9332	0.9335
	0.5	0.9336	0.9332	0.9328	0.9331	0.9329
	0.6	0.9337	0.9334	0.9331	0.9330	0.9327

Table 4.4: Tuning the hyperparameters of the XGBoost classification model with cross-validation based on the Precision.

Precision		Tree Depth				
		30	35	40	45	50
Learning Rate	0.3	0.9885	0.9884	0.9883	0.9883	0.9883
	0.4	0.9884	0.9882	0.9881	0.9882	0.9881
	0.5	0.9882	0.9881	0.9880	0.9880	0.9880
	0.6	0.9880	0.9879	0.9878	0.9878	0.9878

Table 4.5: Tuning the hyperparameters of the XGBoost classification model with cross-validation based on the Recall.

Recall		Tree Depth				
		30	35	40	45	50
Learning Rate	0.3	0.8753	0.8755	0.8758	0.8772	0.8767
	0.4	0.8769	0.8768	0.8764	0.8772	0.8778
	0.5	0.8778	0.8773	0.8765	0.8773	0.8767
	0.6	0.8783	0.8779	0.8772	0.8770	0.8766

Table 4.6: Tuning the hyperparameters of the XGBoost classification model with cross-validation based on the ROC-AUC.

ROC-AUC		Tree Depth				
		30	35	40	45	50
Learning Rate	0.3	0.9741	0.9745	0.9749	0.9752	0.9752
	0.4	0.9742	0.9742	0.9746	0.9752	0.9752
	0.5	0.9744	0.9743	0.9748	0.9748	0.9745
	0.6	0.9742	0.9742	0.9744	0.9744	0.9741

The above results show that the combination of hyperparameters that yield the highest scores across the different metrics varies. Precision and Recall present, as expected, opposite behaviors, and a

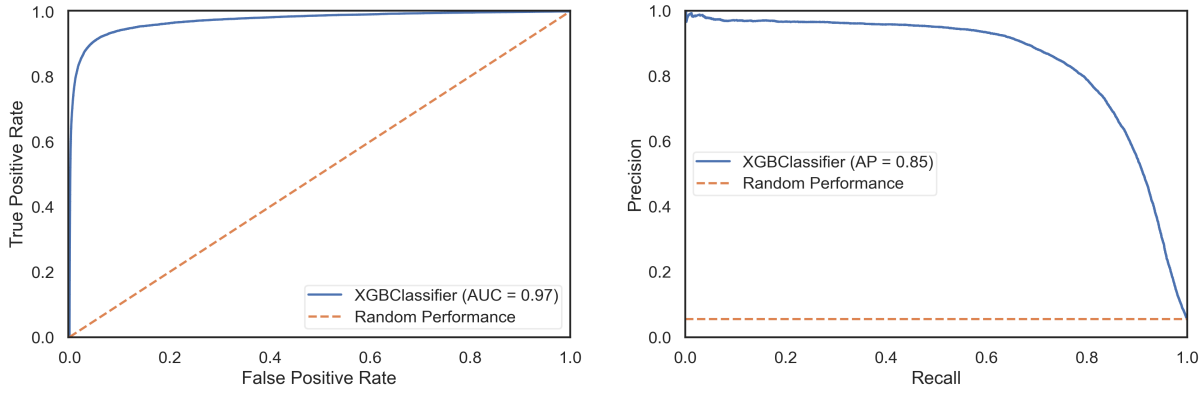
positive change in the Precision metric is reflected by an adverse change in the Recall metric reflecting the trade-off between them. One chose to select the hyperparameters based on the score obtained on the ROC-AUC metric, which corresponds to a learning rate equal to 0.3 and a maximum depth of the tree of 45. The values, in terms of both Precision and Recall, that correspond to this selection are not located in the extreme values of their ranges. The behavior of these metrics also indicate that extending the search space to values of learning rate below 0.3 or above 0.6, or extending the maximum depth below 30 or above 50, would not yield better results since the highest results for the ROC-AUC on validation set do not occur in the edges.

4.3.2 Results on the Test Set

After selecting the hyperparameters, we trained the final model using the entirety of the training data and the selected combination of hyperparameters.

XGBoost final predictions are obtained by weighting all ensemble learners' results since this is a boosting algorithm. The output from the model assigns, during the evaluation of each instance, its probability of belonging to the first class (successful connection) and the probability of belonging to the second class (unsuccessful connection). These two values add up to 1 in all instances. This way of predicting classes imposes another choice, the probability threshold on which we include each of the classes, and the first threshold used was 0.5, which corresponds to the default value. That means that the algorithm classifies an instance as belonging to the successful connection class if the probability on that class is higher than 0.5 (which would mean that the probability of belonging to the unsuccessful connections class is below 0.5) and classified as belonging to the unsuccessful connection class if the probability on that class is higher than 0.5 (which would mean that the probability of belonging to the successful connections class is below 0.5).

The plots for both the ROC and PR curves are shown in Figure 4.1. The AUC_{ROC} is equal to 0.97, the AP to 0.85 and the AUC_{PR} to 0.80.



(a) ROC-AUC curve

(b) PR curve

Figure 4.1: Performance results of the XGBoost Algorithm. The first plot shows, in blue, the ROC curve of the model and the second plot shows, also in blue, the PR curve of the model. In both cases the performance of a random classifier is shown in a dotted orange line.

The results can be further analyzed by looking at each class separately and at the confusion matrix. Table 4.7 presents the results from each one of the classes.

When dealing with imbalanced data, which in the present case has a distribution of 0.945/0.055 between the majority and the minority classes, respectively, only looking at the standard computation of the performance metrics might yield biased results. With that issue in mind, we tested the same metrics previously assessed during the hyperparameters selection, considering the imbalance between classes since the testing data is no longer balanced, contrarily to the training data.

It is essential to look at the Macro, Micro, and Weighted average of the different metrics because they all assess different behaviors. In macro averaging, one computes the performance for each class separately and then averages over classes, whereas, in micro averaging, one collects the decisions for all classes into a single confusion matrix and then computes the metrics from that data. Therefore, the majority class instances dominate the micro average metrics, and macro averaged metrics reflect the influence of the minority class. Finally, in weighted metrics, each class contribution is weighted by its size. Since we are dealing with a class-imbalanced dataset on which the correct prediction of the underrepresented class is more important than the correct prediction of the majority class, macro-averaging may be the better metric.

Overall, one can say that the model performance on what ends up being the majority class of the test dataset surpasses the model performance on the minority class of the dataset. As a whole, the accuracy is very high and stands at 0.98. Table 4.7 summarizes the remaining metrics.

Table 4.7: Classification metrics of the XGBoost algorithm for each of the classes.

	Precision	Recall	F1-score	Support
Class 0 (successful connection)	0.99	0.99	0.99	326169
Class 1 (unsuccessful connections)	0.80	0.75	0.78	19029
Micro average	0.98	0.98	0.98	345198
Macro average	0.89	0.87	0.88	345198
Weighted average	0.98	0.98	0.98	345198

To find what threshold different than the default value, if any, would lead to a better result, we conducted a study that checked the results of the fitted model on the validation data and found that a threshold value of 0.47 (instead of 0.50) increased the number of True Positives. However, the threshold tuning also reduced the number of True Negatives. This study used the G-mean as the criterium to choose the best threshold. The G-mean measures the balance between sensitivity (TPR) and specificity (1-FPR) and focus on finding the best trade-off located in the upper right corner of the ROC curve.

Looking more concretely at the predictions, one can see that the model correctly predicted 98.9% of the majority class instances for the default threshold value, missing only 1.1% of the unsuccessful connection instances. However, as stated earlier, the performance on the unsuccessful connection class was lower, and the model predicted the correct label on 79.1% of the instances and missed the label

20.9% of the time. For the tuned threshold, the model's performance on the majority class decreased to 98.7% of correctly predicted successful connections but increased on the minority class to correctly label 80.2% of instances. Both results are depicted on Table 4.8.

Table 4.8: XGBoost Normalized Confusion Matrix results for both thresholds.

Threshold	0.47	0.50
True Negatives	98.72%	98.86%
False Negatives	19.78%	20.91%
False Positives	1.28%	1.14%
True Positives	80.22%	79.09%

4.3.3 Model Explainability

An essential part of the scope of this thesis is to go beyond the model output and enter the domain of explainability and interpretability. The way that one chose to deal with this component of the post-analysis is to use the principles stated in Section 2.5. The SHAP tool provides both local and global interpretations. The following analysis will focus on more general interpretations that account for all instances used to train the model.

To start with, the idea behind the concept of feature importance defined by SHAP is straightforward: the larger the absolute Shapley value is, the more influential the feature is. Since the goal is to obtain the global importance, one must sum the absolute Shapley values per feature across the data and then sort the results by decreasing importance. The Figure 4.2 shows the SHAP feature importance for the XGBoost model.

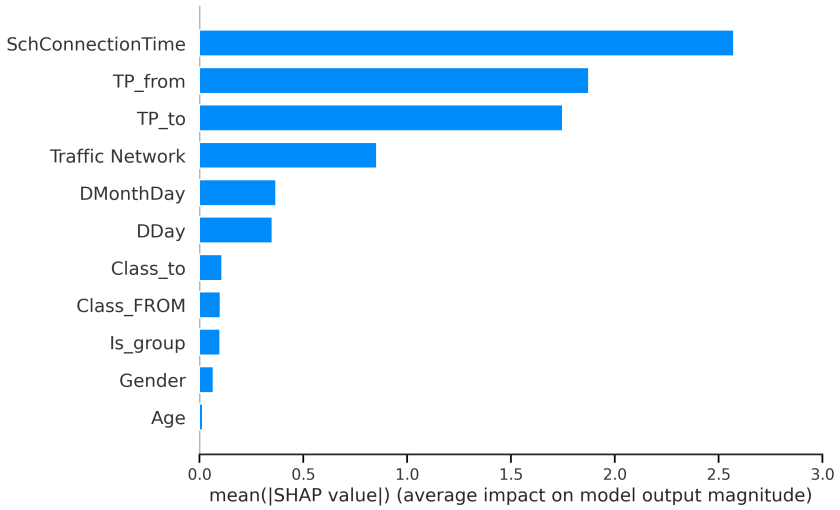


Figure 4.2: Average feature impact on XGBoost output. The chart shows, for each feature, the absolute value of the average impact on the final prediction.

The feature that has the highest impact on the final prediction is the Schedule Connection Time, followed by the incoming flight number closely trailed by the outbound flight number. Outside the top

3 most influential features, the sum of the impacts of the remaining eight features is lower than the combined impact of the first three. The three most minor impactful features are Age, Sex, and if the passenger is traveling within a group. It is also noticeable some importance clustering between similar features, e.g., the feature for the day of the week and the feature for the day of the month have a similar impact in the final prediction, which also happens with features that account for the classes on which the passenger traveled.

Next, the SHAP summary plot accounts for each feature's importance and their effects on the final model output and allows for a more thorough analysis of the specificity of each of the features. This representation is depicted in Figure 4.3. Each instance will be represented by the correspondent Shapley value on the summary plot. The y-axis is reserved for the features ordered by their relative importance, and the height of the clusters indicates the distribution of the Shapley values per feature. The x-axis represents the Shapley value. The color attributed to each point represents the value of the instance on the corresponding feature, ordered from low to high.

As said before, the feature with the highest importance is the Schedule Connection Time, and as shown in Figure 4.3 samples with low connection time are associated with positive SHAP values, which means that a low connection time value contributes to classify the connection as unsuccessful. This follows the expected behavior, and it is consistent with common sense. From the distribution of the instances' points throughout the feature span, it is noticeable that low and medium values for the Schedule Connection Time have a negative impact and contribute to classifying the output as a successful connection; however, most samples are located within the range from 0 to 4, contributing to the classification as unsuccessful. There is a clear separation between low values that contribute positively and low to medium values that contribute negatively.

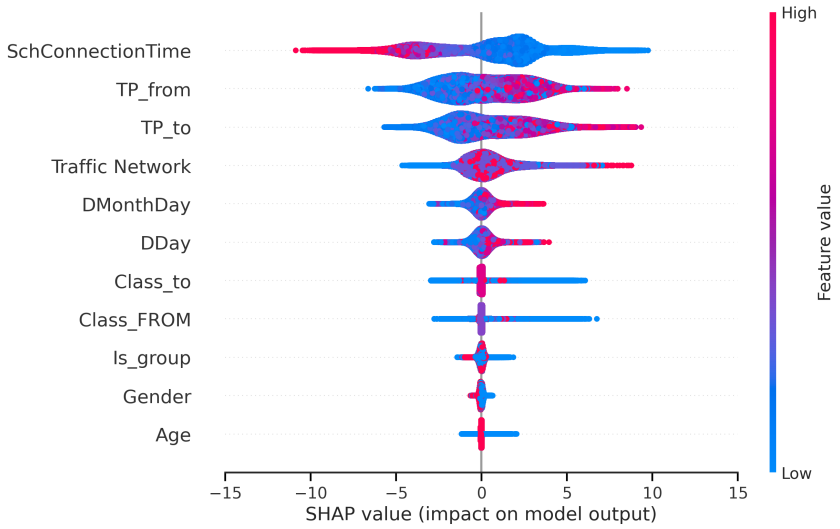


Figure 4.3: SHAP summary plot for the XGBoost algorithm. The plot shows, for each feature, what type of impact each instance had on the final prediction. Low values in the features span are associated with blue dots and high values with red dots.

Despite assuming an essential role in the prediction, the following two most important features, the

incoming flight and the outbound flight number cannot be analyzed entirely since the data is encoded and represents a categorical feature with no ordinal logic behind it. In terms of impacts on the model output, the span of these features is shorter than the one seen in the Schedule Connection Time, and unlike in the latter, there is no clear separation between high and low values.

Then, the traffic network variable has a mixed impact on the final prediction. High values can either have a high positive impact (classify the connection as unsuccessful) or a more moderate one. Low values can either have a positive impact, but not as high as the previous statement, or have a negative impact (classify the connection as successful). The encoding of the different combinations of transiting routes followed the order NN $<$ NS $<$ SN $<$ SS. It is interesting to notice that SS connections can have such different impacts on the final prediction and that most NN connections have a negative impact (classify the connection as successful) which perhaps comes from the fact that these types of connections relate with longer flights which tend to have more sparse schedules and higher connection times.

The following feature, DMonthDay, has a scattered distribution that comes naturally from its essence since it represents the encoded day of the month (from 1 until 31) when the connection occurred. Due to the moving nature of the placement of weekends and different weekdays as numbers, it is not easy to extract information on this behavior; however, this feature holds some relevant information since SHAP values are not zero. Instances with low values for the day of the week feature, DDay, tend to contribute to classifying the instances as successful, and higher values in this feature tend to classify them as unsuccessful.

For the traveling class, both inbound and outbound, the order on which the different categories were encoded is the same and is as follows: Business $<$ Groups $<$ Economy $<$ Allots $<$ R1. For the case of the outbound traveling class, high values have a marginal impact on the final predictions, which means that Business class instances can either have a high positive impact or a modest negative impact. Whereas in the case of the incoming traveling class, intermediary values corresponding to the Economy and Groups classes have a marginal impact, and the business class keeps the same behavior as in the incoming class feature.

The final three features have the most negligible impact on the predictions. Nevertheless, one noticed some general trends in those features. To start with, the three have most of the instances concentrated around zero, which indicates a null impact. Then for both the traveling within a group and sex features, it is possible to spot a shy partition of the data on which high values tend to have a positive impact and low values a negative impact. Both features are binary, which means that passengers traveling in a group tend to have a slightly positive impact (classify the connection as unsuccessful), whereas passengers traveling alone tend to classify the connection as successful, i.e., have a negative impact. When considering the Sex of the passengers, the impact is even less expressive than in the case of the group feature, and Men tend to have a positive impact, contrary to what happens with Women that have a negative impact. The last binary feature, Age, has no impact on high values (adults) and moderate or no impact on low values (infants).

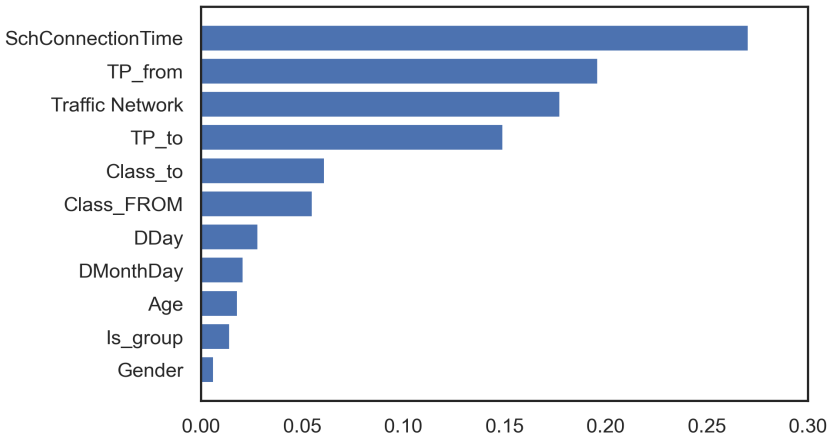


Figure 4.4: Intrinsic feature Importance of XGBoost model. This chart shows, for each feature, the relative importance measured in terms of the built-in tool of the model.

However, the results seen above for the feature importance assessed by the SHAP values are different from the feature importance results directly extracted from the XGBoost API. As shown in Figure 4.4, the top 2 contributors are the same as seen earlier, but the outgoing flight code and the Traffic Network features switch places in terms of importance. Then both the date and the traveling class-related features keep the same clustering seen above but in a different order. The XGBoost API assigns higher importance to the class-related features, contrary to the SHAP that assigned day-related features more importance than class-related ones. Finally, the last three features are the same and still present only a moderate impact. Contrary to the SHAP explanation, the least important feature is Sex.

4.4 Neural Networks

4.4.1 Hyperparameter Tuning

With more computation power and data available, neural networks are becoming very popular. To achieve the highest accuracy and faster convergence possible is essential to find the optimal architecture of the DNN. Besides, the training process of DNNs becomes tedious with the increasing of the depth of the network since they can be tweaked using several hyperparameters, such as the number of hidden layers, the number of units at each hidden layer and the dropout rate, among others.

The optimal architecture of DNNs is usually found using a trial-and-error process, which is an exponential combinatorial problem and a tedious task which the colossal amount of data of this thesis only exacerbates. To address this, an automatic hyperparameter optimization software framework was used. This framework was mainly designed for machine learning problems and allows for the dynamic construction of the search space for the hyperparameters. [57]

Hyperparameter Space

For the DNN, the tested hyperparameters were: the depth of the network, in the range from 1 to 10 hidden layers with multiple layer dimensions in the range of 4 to 256 units; dropout rates within the range from 0.01 to 0.8, batch size of the set [128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768] and the number of epochs in the set [20, 50, 80]. Additionally, on a higher level, the tuning process occurred over 20 trials, which is yet another hyperparameter that comes from the Optuna framework, and to select the activation function, one did two independent studies. There is no rule of thumb on addressing the optimal amount of trials. Thus, the choice was 20 so that the framework could have some freedom to test out different hyperparameters configurations but not too much freedom that would be a computational burden. Regarding the activation function, the studies focused on the ReLU, and PReLU types due to the reasons explained in Section 2.1.5. This was the choice for all layers except the final one that has the Sigmoid activation function given the problem's binary classification scope.

All values present on Table 4.9 refer to the value of the AUC_{ROC} on the validation set measured for each of the hyperparameters combinations set by Optuna. In that sense, they are not directly comparable because all hyperparameters combinations used different architectures, but since they measure the same metric across the same set of data, this value will allow the selection of the best hyperparameters. From the table, it is also possible to see that generally, the highest value is obtained for 50 epochs, indicating that training with only 20 epochs might result in an under-fitted model while training with 80 epochs might produce over-fitted models.

As for the best model, one chose the model corresponding to 50 epochs and batch size of 8192. This model obtained a AUC_{ROC} on the validation set of 0.854, the highest among all tested architectures. For these hyperparameters, the optimization framework outputted five layers in total: the input layer, three hidden layers, and the output layer.

Table 4.9: Tuning the hyperparameters of the DNN classification model based on the ROC-AUC of the validation dataset.

		ReLU			PReLU		
		Epochs					
		20	50	80	20	50	80
Batch Size	128	0.846	0.849	0.848	0.846	0.848	0.850
	256	0.848	0.849	0.849	0.850	0.850	0.847
	512	0.848	0.846	0.846	0.849	0.848	0.849
	1024	0.852	0.850	0.847	0.851	0.847	0.848
	2048	0.853	0.851	0.848	0.847	0.851	0.845
	4096	0.846	0.845	0.848	0.848	0.845	0.846
	8192	0.847	0.854	0.848	0.847	0.853	0.850
	16384	0.842	0.846	0.846	0.848	0.851	0.847
	32768	0.846	0.850	0.842	0.846	0.844	0.845

The input layer consists of 4 hidden units, a normalization layer, and a dropout rate of 0.3956. The activation function of this layer is the ReLU which is a characteristic shared with the hidden layers.

The following three layers were composed of 100, 139, and 244 units, respectively. Like in the previous layer, a normalization step was added before the activation function to avoid unexpected behaviors. The dropout rates for the hidden layers are 0.777, 0.591, and 0.713, respectively.

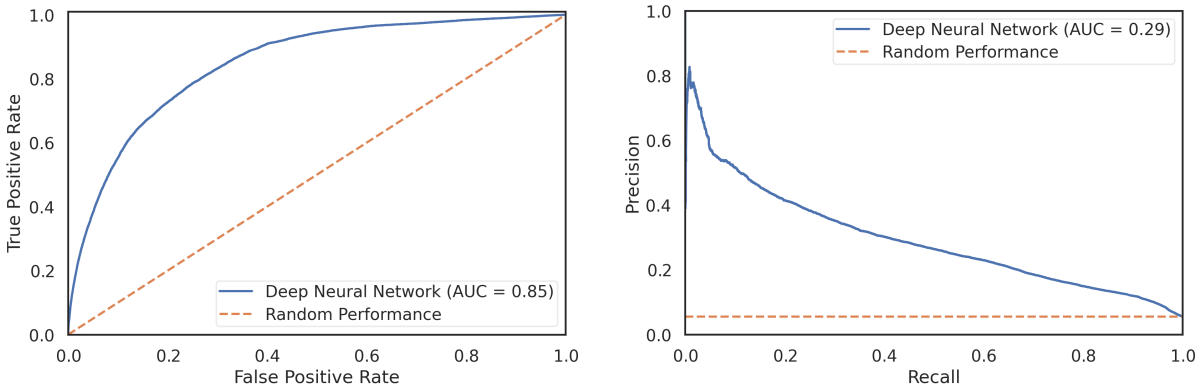
The final layer receives the information passed by the last hidden layer and outputs the prediction after applying the Sigmoid function. Here the prediction is a single value, and again the absolute value relative to the threshold will determine which class the prediction fits in.

4.4.2 Results on the Test Set

The procedure after the selection of the hyperparameters for the DNN is the same as the one followed when training the XGBoost. The whole data was fed to the architecture based on the chosen architecture. As stated before, DNN final predictions include a single value in the range from 0 to 1.

Figure 4.5 shows the plot for the ROC-AUC and PR curves. The AUC_{ROC} is equal to 0.85 and an the AUC_{PR} is equal to 0.29.

Overall, one can say that the model performance follows the same trend seen with the XGBoost algorithm, and the performance on the majority class surpasses the model's performance on the minority class of the test dataset. With the DNN the results on the majority class were below but close to the results seen with the XGBoost; however the results on the minority class were well below the ones seen with the XGBoost making the macro averaged metrics of the DNN algorithm between 0.64 and 0.66 contrary to the range from 0.87 to 0.89 seen in the XGBoost. As a whole, the accuracy in the test dataset is 0.92. Table 4.10 summarizes the remaining metrics.



(a) ROC-AUC curve

(b) PR curve

Figure 4.5: Performance results of the DNN Algorithm. The first plot shows, in blue, the ROC curve of the model and the second plot shows, also in blue, the PR curve of the model. In both cases the performance of a random classifier is shown in a dotted orange line.

Table 4.10: Classification report of the DNN algorithm.

	Precision	Recall	F1-Score	Support
Class 0 (successful connection)	0.96	0.96	0.96	326169
Class 1 (unsuccessful connection)	0.32	0.36	0.34	19029
Micro average	0.92	0.92	0.93	345198
Macro average	0.64	0.66	0.65	345198
Weighted average	0.93	0.92	0.92	345198

As in the case of the XGBoost a threshold is needed and this value was again found via the G-mean optimization on the validation data. When evaluating the fit of the validation data, the threshold that corresponds to the highest G-mean is 0.30.

Applying the default threshold of 0.50, the model performed fairly well in the majority class, missing only around 5% of the predictions. However, the case with the minority class is different, and the model can only predict 36% of the instances correctly, miss-labeling the remaining ones. If instead the default value one considers the tuned threshold, there is a big reduction in the number of correctly predicted successful connections, but an expressive increase follows this in the number of correctly predicted unsuccessful connections that go from only 36% to more than 75%. These results are depicted in Table 4.11. The same principle of the XGBoost analysis can be applied here, and, in theory, the gain from predicting a higher number of unsuccessful instances can offset the reduction in the number of correctly predicted successful classes. However, when factoring in the sheer difference in the absolute values, this might not be the best approach for the airline in terms of costs because all precautionary measures might become higher than all corrective costs.

Table 4.11: DNN Normalized Confusion Matrix results for both thresholds.

Threshold	0.30	0.50
True Negatives	77.88%	95.51%
False Negatives	24.79%	63.97%
False Positives	22.12%	4.49%
True Positives	75.21%	36.03%

4.4.3 Model Explainability

Figure 4.6 shows that the feature that has the highest impact in the final prediction is the Schedule Connection Time, following the behavior of the XGBoost model.

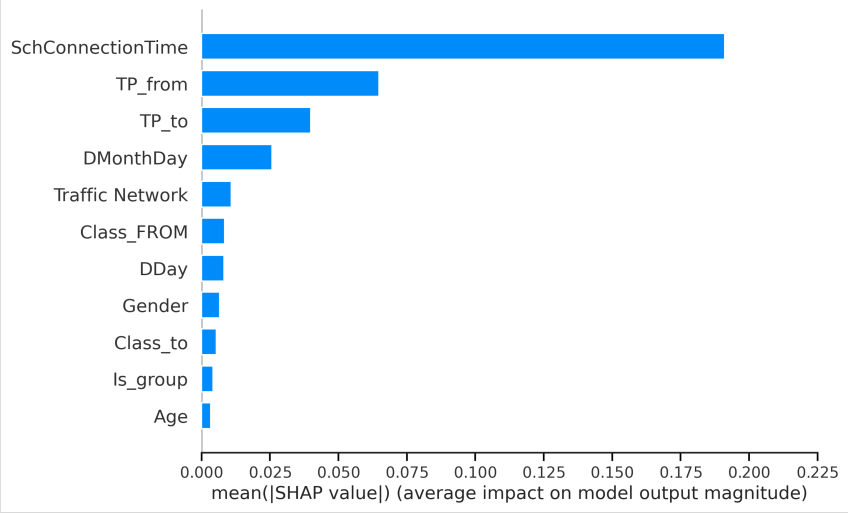


Figure 4.6: Average feature impact on the DNN output. The chart shows, for each feature, the absolute value of the average impact on the final prediction.

The following two most essential features also follow the trend seen in the interpretation of the XGBoost model and are the incoming and outgoing flight codes, respectively. Here, and imitating the previous behavior, the first three features account for most of the interpretative value of the predictions made by the model.

From the 4th most impactful feature on, it is possible to spot a clear from the previous model, and the remaining features, except for the day of the month, assume an equal and yet marginal contribution to the model's predictions. However, the traveling class (both inbound and outbound) and day of the week have a slightly higher impact than the other 3. The overall order is similar to XGBoost's explanation, with age and sex being again ranked at the bottom.

The bold number in the plot shown in Figure 4.7, 0.08, is the model's score under a specific observation. Higher scores lead the model to predict an unsuccessful connection, whereas lower scores lead the model to predict a successful connection. The features that were important to predict this specific observation are shown in red and blue colors, with red representing features that increased the model's score higher and blue representing features that did the opposite.

The relative position of each one of the features also indicates their relative impact on the final prediction. Features that had the most impact on the score are located closer to the dividing boundary between red and blue, and the length of each feature bar represents the size of that impact.

For the case of the first instance of the test set, the feature with the most impact on the final decision was the Schedule Connection Time, followed by the incoming and outgoing flight numbers. All these three features, among others less relevant, contributed negatively to the prediction and lowered the model's score. On the other hand, DMonthDay contributed the most to increase the model's score. These four features are the most relevant to the final prediction, and this result is coherent with the global average feature impact shown in Figure 4.6.

Figure 4.7: SHAP force plot for one DNN prediction. The plot shows the explanation for a single prediction by the model, showing in red, the the features that contributed the most positively and in blue the features that contributed the most to decrease the model score to this specific data point. The bold value represents the model's prediction.

4.5 Logistic Regression

4.5.1 Hyperparameter Tuning

The Logistic Regression algorithm also involves the crucial hyperparameters tuning process. Several parameters can be tuned, and among them there are five available solvers, all of each try to find the parameter weights that minimize the cost function. The available solvers are: the Newton method solver, *newton-cg*; the Limited-memory Broyden–Fletcher–Goldfarb–Shanno solver, *lbfgs*; the Stochastic Average Gradient descent, *sag*; and the *saga*.

However, it is important to notice that some solvers present limitations regarding the compatibility with penalty types, such as the case with the *sag* and *saga*, where the former did not allow for the L1 regularization, but the latter did. Overall there are four penalty types available: adding an L2 penalty term, which is the default choice of the Logistic Regression algorithm; adding an L1 penalty term; Elasticnet that adds both L1 and L2 penalties; and the case no penalty is added. Another critical parameter to tune is the inverse of the regularization strength, and here smaller values specify stronger regularization.

Hyperparameter Space

For the Logistic Regression, the tested hyperparameters were the 3 described earlier: the solver type, the penalty term, and the inverse of regularization strength value.

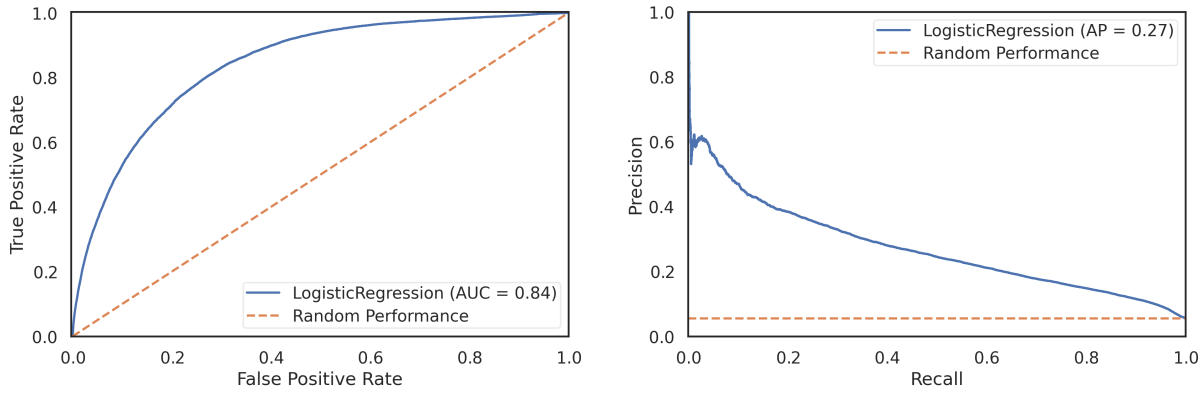
For the solver type, one tested all five options, even the *newton-cg* and the *lbfgs* even though the run time when using them could be longer than that with other options. For the penalty, the grid search included all three penalty functions available plus the no penalty option, and for the regularization parameter, the values included the set [100, 10, 1.0, 0.1, 0.01].

The performance, measured in terms of AUC_{ROC} on the validation set, was very similar across the different hyperparameters combinations. The absolute highest value was attained for the configuration that included *saga* as the solver, L2 as the regularization, and 1 as the regularization parameter.

4.5.2 Results on the Test Set

After the selection of the hyperparameters, the final model was trained using the whole data and the selected combination of hyperparameters.

Figure 4.8 shows the plot for the ROC-AUC and PR curves. The AUC_{ROC} is equal to 0.84 and an the AUC_{PR} is equal to 0.46.



(a) ROC-AUC curve

(b) PR curve

Figure 4.8: Performance results of the Logistic Regression Algorithm. The first plot shows, in blue, the ROC curve of the model and the second plot shows, also in blue, the PR curve of the model. In both cases the performance of a random classifier is shown in a dotted orange line.

We can say that the model performance follows the same trend seen with the Black-Box models, and the performance on the majority class surpasses the model's performance on the minority class of the test dataset, except for the Recall metric in which the model presented similar performance on both classes. However, the difference in performance between the two classes on the Precision metric is even greater than the difference seen in the former models, and the difference on the F1-score is close to the difference seen in the DNN. In terms of macro-averaged metrics, this model performance is worse than the XGBoost for all assessed metrics. When comparing with the DNN, the Precision and F1-score macro-averages are lower yet similar, but the Recall is higher and equal to the value seen in the XGBoost. As a whole, the accuracy is 0.77. Table 4.12 summarizes the remaining metrics.

As with the Black Box models, a threshold value is needed to analyze this algorithm performance at the confusion matrix level correctly. This threshold value was again found via the G-mean optimization on the validation data. When evaluating the fit of the validation data, the threshold that corresponds to the highest G-mean is 0.49.

Table 4.12: Classification report of the Logistic Regression algorithm.

	Precision	Recall	F1-Score	Support
Class 0 (successful connection)	0.98	0.78	0.87	326169
Class 1 (unsuccessful connection)	0.16	0.75	0.27	19029
Micro average	0.77	0.77	0.77	345198
Macro average	0.57	0.76	0.57	345198
Weighted average	0.94	0.77	0.83	345198

The model had a limited performance by applying the default threshold value since it missed around 22% of the instances belonging to the majority class and around 25% of the cases when the instance belonged to the minority class. These results, although modest, reached a performance in the minority class better than the ones seen with the DNN and only slightly worse than the XGBoost results. However, if one considers the tuned value instead, there is no big change regarding the predictions, and the number of correctly predicted unsuccessful connections goes up from 75% to 75.7%, while the number of correctly predicted successful connections goes down from 77.5% to 76.9%. These results are depicted in Table 4.13.

Table 4.13: Logistic Regression Normalized Confusion Matrix results for both thresholds.

Threshold	0.49	0.50
True Negatives	76.92%	77.53%
False Negatives	24.28%	25.04%
False Positives	23.08%	22.47%
True Positives	75.72%	74.96%

4.5.3 Model Explanability

The SHAP *post hoc* explanation of the Logistic Regression model shows that, once again, the feature with the highest importance is the Schedule Connection Time as shown in Figure 4.9. Instances with low connection times are associated with positive SHAP values, meaning that a low connection time value contributes to classifying the connection as unsuccessful, which is coherent with the behavior seen in the XGBoost model. Data points with low and medium Schedule Connection Time values have a negative impact and classify the connection as successful. In this feature, most samples are located within the range from 0 to 5, meaning that the contribution of low connection time values is more localized than the contribution of low connection times that span a larger influence region. There is a clear separation in this feature between low values that contribute positively and low to medium values that contribute negatively.

The following two most important features are the incoming flight and the outbound flight number which was also the case with both Black Box models. Despite assuming a vital role in the prediction,

these features cannot be completely analyzed since the data represents a categorical feature that is encoded, meaning that there is no ordinal logic behind it. The remaining eight features have only a moderate in the model's predictions.

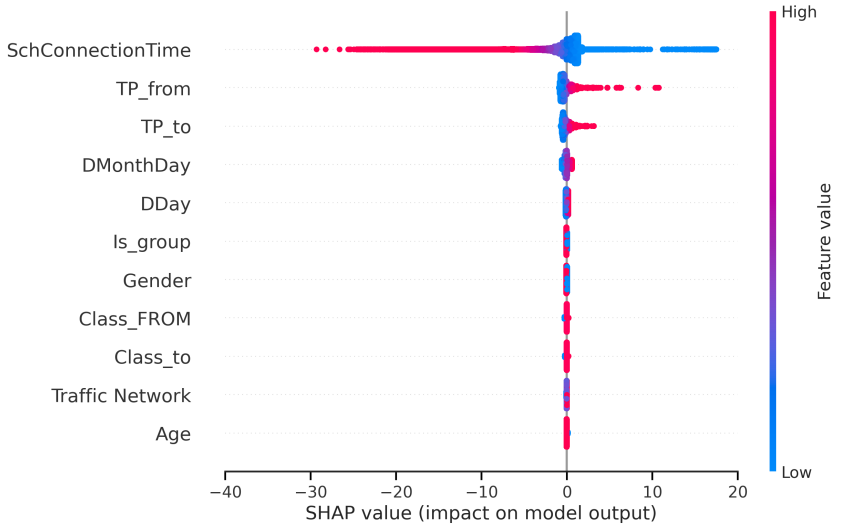


Figure 4.9: SHAP summary plot for the Logistic Regression algorithm. The plot shows, for each feature, what type of impact each instance had on the final prediction. Low values in the features span are associated with blue dots and high values with red dots.

As in the case of the XGBoost, the Logistic Regression algorithm also has a built-in feature importance rank. As shown in Figure 4.10 the most crucial feature is once again the Schedule Connection Time. Since this feature has negative coefficient, high Schedule Connection Time values push the classification more towards the negative class, i.e., the successful connection class.

The other relevant features are the incoming and outbound flight numbers, which contrarily to the most important feature, have positive importance. High values in these features push the classification towards the positive class, i.e., the unsuccessful connection class. Again, as earlier, this behavior cannot be analyzed entirely since the data is encoded and represents a categorical feature with no ordinal logic.

The interpretation extracted from the intrinsic feature importance is coherent with the explanations made by SHAP. The main differences happen after the 3rd most relevant feature because SHAP explanation attributed more modest importance to the remaining features. This is visible with, for example, the day of the month feature, DMonthDay, which has a higher impact on the intrinsic explanation than the one seen with SHAP. The top 5 features are the same for both sources, the remaining 6 have a slightly different order, and the least important feature changes from the Age feature to Traffic Network.

Although applicable, the interpretability explanation given in Section 2.1.1 is not straightforward since all data was previously encoded. Because of that, one needs first to understand what it means to increase each of the features by one unit. In the case of the most important feature, Schedule Connection Time, the data originally ranged from -2210 until 2825 minutes, and the encoded data ranges from -10.9858 to 12.4891, which means increasing the scaled data by 1 unit corresponds to increase the actual time by 214 minutes. In the case of binary features, they originally were either 0 or 1. After the

scaling process, they became either -1.025 or 0.975, meaning that, for example, a change between being an infant to being an adult corresponds to an increase of 2 units in the scaled version of the feature.

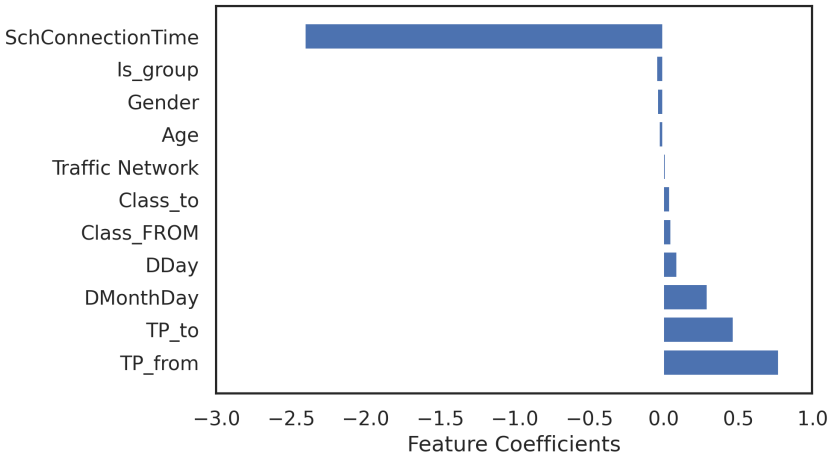


Figure 4.10: Feature Coefficients of Logistic Regression model. The coefficients outputted by The Logistic Regression algorithm show that the most important feature is once again the time scheduled for the connection followed by the incoming and outgoing flight codes, respectively.

An increase of 214 minutes in the Schedule Connection Time changes the odds of successful *versus* unsuccessful connection by a factor of -2.4 when all other features remain the same. This roughly means 90% reduced odds of missing the connection when having an extra 214 minutes connection time than a passenger who does not have the extra connection time.

The remaining non-binary categorical features do not have a meaningful interpretation since there is no ordinal logic behind their nature. Despite being easily interpretable, the binary features have a minimal impact, and their interpretability is not relevant. All binary features have a negative factor, meaning that the odds for successful *versus* unsuccessful connection are by a factor of 0.97, 0.96, and 0.95 lower in the case of Age, Sex, and Is_group, respectively. Since two units separate the scaled versions of the binary features, this means that being an adult (Age equals 1) translates to a reduction in terms of odds of missing the connection of 6%, or that being a male (Gender equals 1) translates to a reduction in terms of odds of missing the connection of 7% or even that traveling in a group (Is_group equals 1) translates to a reduction in terms of odds of missing the connection of 8%.

4.6 Decision Tree Classifier

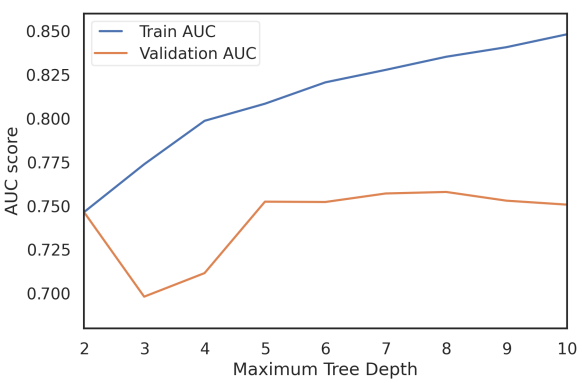
4.6.1 Hyperparameter Tuning

Before training the Decision Tree Classifier algorithm, one carefully considered the selection of hyperparameters since this model's intrinsic interpretable nature might be lost if the tree grows in depth beyond reasonable. As stated in Section 2.1.2 the width of the tree grows exponentially with the depth,

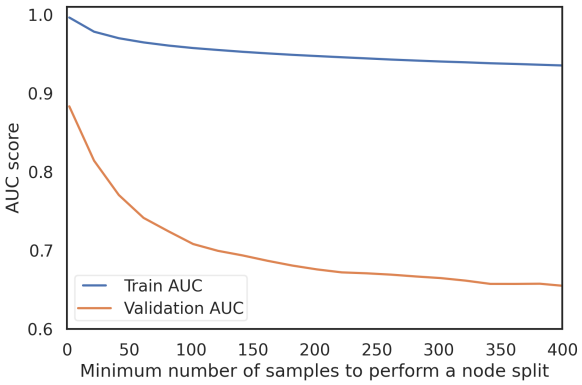
so this is an essential parameter to tune. Other relevant hyperparameters include the minimum number of samples required to split an internal node, the minimum number of samples required at leaf nodes, the number of features to consider when looking for the best split and the criteria for the split, either Gini or Entropy.

Hyperparameter Space

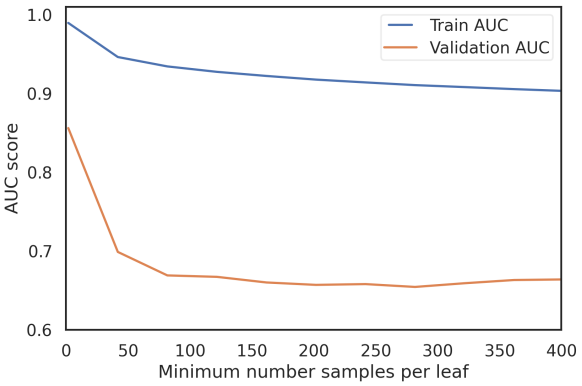
For the Decision Tree Classifier, the tested hyperparameters were: the depth of the tree ranging from 2 until 10, the minimum number of samples to split an internal node as well as the minimum number of samples required at leaf nodes between 2 and 400 and the the number of features to consider when looking for the best split corresponding to the range of available features, therefore from 1 until 11. A preliminary study found that the Gini criterion results are very similar to the results using the Entropy criterion. Therefore the choice was to use the less computationally expensive method, i.e., the Gini criterion.



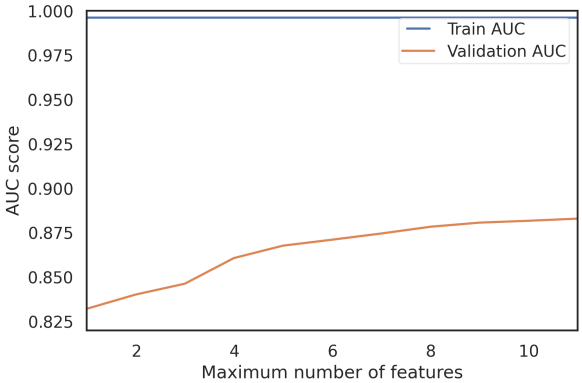
(a) Maximum depth



(b) Minimum number of samples required to split an internal node.



(c) Minimum number of samples required at leaf nodes.



(d) Number of features to consider when looking for the best split.

Figure 4.11: Decision Tree Classifier hyperparameter tuning. For each of the four hyperparameters the performance of the model was assessed both on the training and on the validations set.

Figure 4.11 shows the results for the AUC_{ROC} score both in terms of the training and validation data. Each plot shows the evolution of the two metrics by studying the influence of varying a single hyperparameter while keeping all other parameters with the default values. As already stated, the most

critical feature is the depth of the tree, and as seen in Figure 4.11 the slight gain in the performance after a depth of 5 is not worth the loss in interpretability, therefore one chose a depth of 5 to build the tree. Values lower than 5 are not advisable since the jump AUC_{ROC} from depth 3 or 4 is considerable at the expense of augmenting the tree depth of 2 or 1 units, respectively.

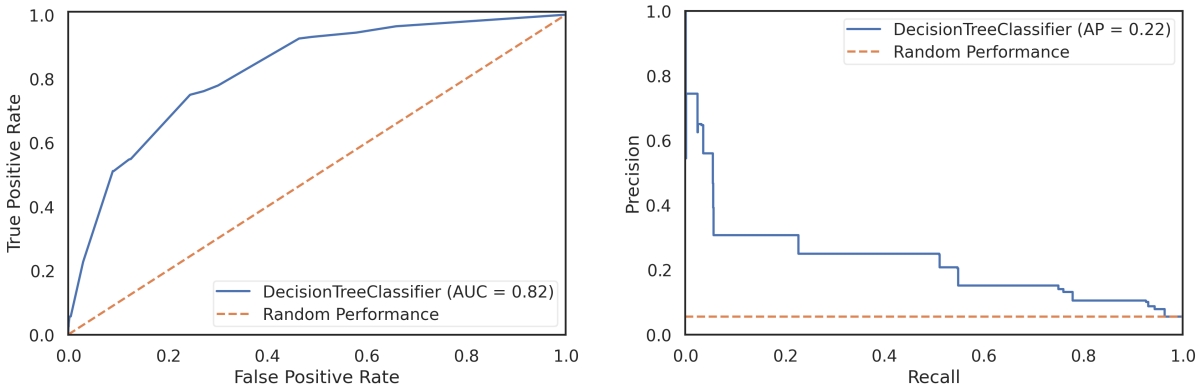
The results for the minimum number of samples to split an internal node as well as the minimum number of samples required at leaf nodes follow a similar behavior to each other and the maximum performance in terms of AUC_{ROC} score on the validation dataset is attained for the smallest value allowed for these parameters, i.e., 2. Finally, for the number of features to consider when looking for the best split, the maximum performance in terms of AUC_{ROC} score on the validation dataset is attained for the total number of features, i.e., 11.

4.6.2 Results on the Test Set

After the selection of the hyperparameters, one trained the final model using the whole data and the selected combination of hyperparameters.

Figure 4.12 shows the plot for the ROC-AUC and PR curves. The AUC_{ROC} is equal to 0.82 and an the AUC_{PR} is equal to 0.46.

Once again, the model performance follows the same trend seen with the Black-Box models and with the Logistic Regression, and the performance on the majority class is better than the model's performance on the minority class except for the Recall that has similar values on both classes. The difference in performance between the two classes among the remaining metrics is similar to the one seen in the Logistic Regression model. The macro-averages scores are close to the ones seen with the Logistic Regression, and therefore lower yet similar to the ones seen with the DNN except for the Recall, which is higher in the interpretable models. As a whole, the accuracy is 0.75 and Table 4.14 summarizes the remaining metrics.



(a) ROC-AUC curve

(b) PR curve

Figure 4.12: Performance results of the DNN Algorithm. The first plot shows, in blue, the ROC curve of the model and the second plot shows, also in blue, the PR curve of the model. In both cases the performance of a random classifier is shown in a dotted orange line.

Table 4.14: Classification report of the Decision Tree Classifier algorithm.

	Precision	Recall	F1-Score	Support
Class 0 (successful connection)	0.98	0.76	0.85	326169
Class 1 (unsuccessful connection)	0.15	0.75	0.25	19029
Micro average	0.75	0.75	0.75	345198
Macro average	0.57	0.75	0.55	345198
Weighted average	0.94	0.75	0.82	345198

For the decision tree classifier, the search for the adequate threshold showed that there are no predictions made by the model on the validation dataset whose probabilities fall around the default threshold value. Therefore both the default threshold value and the value yielded from the G-mean analysis present the same distribution in terms of classification, i.e., TP, TN, FP and FN. Table 4.15 summaries the results.

Table 4.15: Decision Tree Classifier Normalized Confusion Matrix results for both thresholds.

Threshold	0.53	0.50
True Negatives	75.50%	75.50%
False Negatives	25.04%	25.04%
False Positives	24.50%	24.50%
True Positives	74.96%	74.96%

4.6.3 Model Explainability

Figure 4.13 display the only four relevant features to the model explanation. Besides the Schedule Connection Time, only the incoming and outgoing flight numbers and the Traffic Network influence the output. Despite assuming an important role in the prediction, the second and third most important features cannot be completely analyzed since the data represents a categorical encoded feature; therefore, there is no ordinal logic behind it. Finally, the Traffic Network feature has almost no impact on the final predictions.

In terms of the distribution among the classes, all features influence predicting Class 0 in similar terms as to predict Class 1. We looked at the instances' contribution to the final prediction of data belonging to the majority class. Figure 4.14 represents these contributions and the equivalent plot for data points belonging to the minority class, is the symmetric of the graph in Figure 4.14 in relation to the y-axis.

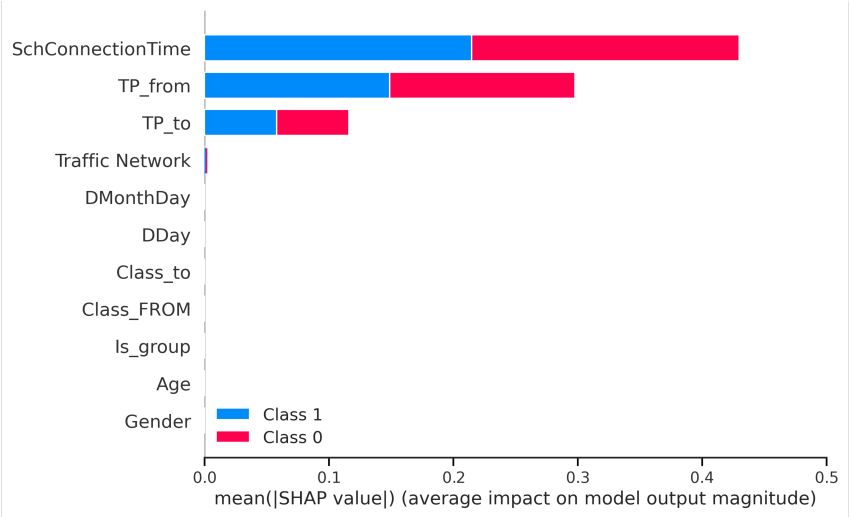


Figure 4.13: Average feature impact on Decision Tree Classifier output per class. The chart shows, for each feature, the absolute value of the average impact on the final prediction separated between the two classes.

The following two most important features, the incoming flight and the outbound flight numbers, do not have, once again, any ordinal logic behind them. The span of these features, in terms of impact on the model output, differs, and in the case of the outbound flight number, it is shorter than the one seen in the Schedule Connection Time.

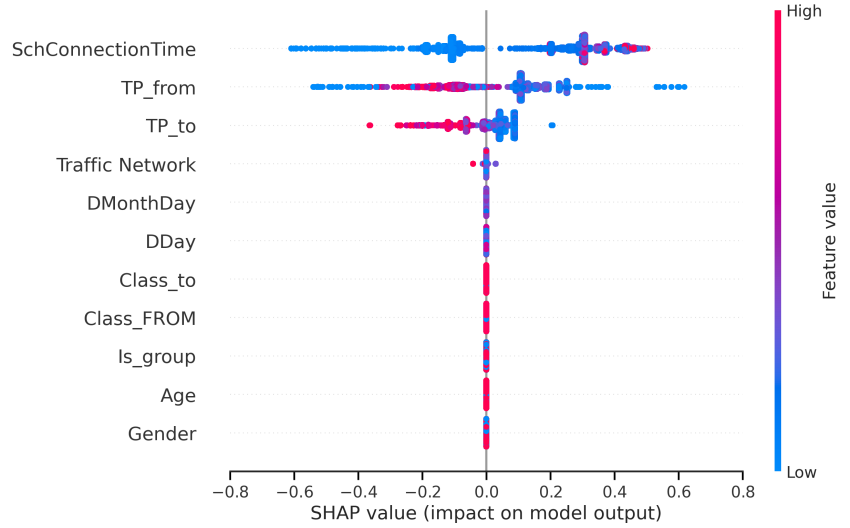


Figure 4.14: SHAP summary plot for the successful class of Decision Tree Classifier algorithm. The plot shows, for each feature, what type of impact each instance belonging to the majority class had on the final prediction. Low values in the features span are associated with blue dots and high values with red dots.

However, the span of the range of impact for the incoming flight number is similar to the one seen in the Schedule Connection Time. Unlike the Schedule Connection Time and the outbound flight number, the incoming flight number does not separate the impacts of high and low values. The remaining fea-

ture with any impact on the predictions, the Traffic Network variable, has almost no impact on the final predictions since most of the data points are concentrated on the origin.

Like in the case of the XGBoost and the Logistic Regression models, the Decision Tree Classifier also presents a built-in feature importance rank which is depicted in Figure 4.15. The results show that although the ranking of features' importance is the same, their relative values are not, and the Schedule Connection Time assumes a more preponderant position relative to the remaining three features.

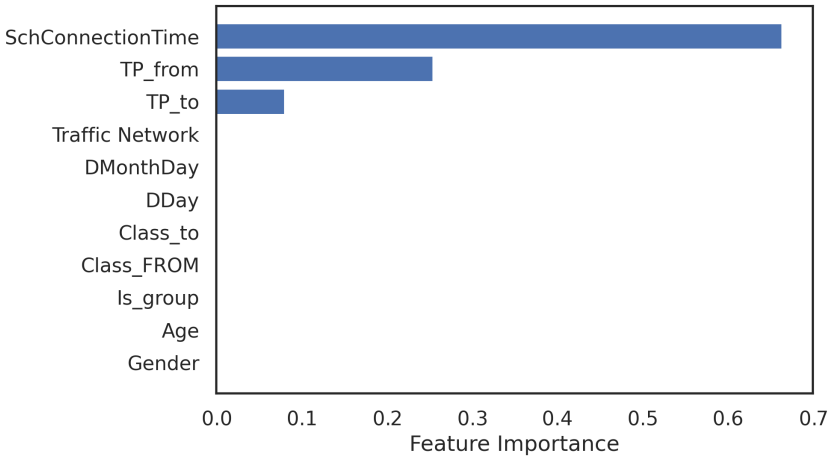


Figure 4.15: Intrinsic feature Importance of Decision Tree Classifier model. This chart shows, for each feature, the relative importance measured in terms of the built-in tool of the model.

4.7 RuleFit

As stated in Section 2.1.4, RuleFit results are not always promising regarding Classification Tasks. Even with this in mind, one decided to train a rule-based algorithm as the third type of intrinsically interpretable model.

The first step was, as always, to tune the hyperparameters. However, before jumping to the tuning process, one did a preliminary inspection to find the acceptable range for the hyperparameters. The entirety of the training dataset was fed to the algorithm to find how different sets of hyperparameters influenced the results on the validation set. Keeping in mind that this is an intrinsic interpretable model, the goal is to reach a good performance while keeping the number of rules and the maximum depth of the auxiliary trees to the bare minimum.

However, the preliminary study showed poor model performance, even with great flexibility regarding the number of rules and maximum tree depth. Even though the model was trained with the balanced dataset, it was incapable of having a good classification performance on both classes and assigned the majority of instances to the dominant class in all hyperparameters combinations that one tried. The preliminary study was a grid search with the maximum depth of the tree in the set [3, 10] and the maximum number of rules in the set [50, 250, 500]. In all trials, the accuracy was high, around 0.95. However, the performance in the minority class was terrible across the board. The F1-score of the minority class ranged from a minimum 0.11 in the smallest hyperparameters combination, a maximum

depth of 3 and a maximum of 50 rules, to a maximum of 0.21 in the combination of hyperparameters corresponding to a maximum depth of 10 and a maximum of 500 rules.

Table 4.16 summarizes the results in terms of ROC_{AUC} of all hyperparameters combinations tested during the preliminary study.

Table 4.16: Preliminary study of the ROC-AUC score on the validation dataset for the hyperparameters of the RuleFit classification model.

ROC-AUC		Maximum Number of Rules		
		50	250	500
Maximum	3	0.52	0.53	0.55
Depth	10	0.54	0.55	0.56

The highest value in terms of the ROC_{AUC} score on the validation set corresponds to the edge of the grid. However, no further search was performed because the correspondent hyperparameters already make the model uninterpretable, and besides, the memory requirements would make this search computationally expensive. The model corresponding to a maximum depth of 10 and a maximum of 500 rules had 277 rules with non-zero coefficients, making this model already non-intelligible for humans.

In light of the findings, the decision was made not to continue pursuing a rule-based interpretable algorithm.

4.8 Costs Assessment

The analysis presented in this Section is based on the work developed by Guimarães [43]. The original work presents the cost analysis of which we replicate the equations and approximations. The present work is, therefore an extension of the original paper that includes other models besides the XGBoost. All four strategies presented so far have different results than the values obtained by the TAP baseline model described in Section 4.1.

As stated in the original work, the airline will incur in two types of expenses: Precautionary Costs, C_{Pre} and Corrective Costs, C_{Cor}. The current system in place, using the Baseline model, does not consider Precautionary costs and assumes a corrective only approach.

When the model correctly predicts a TP, the airline will have helpful information to minimize the disruption and will incur precautionary costs. Some possible actions include, but are not limited to, the delaying of the second leg of the journey, assigning some airport personnel to escort the passengers to their next flight, or even assigning a seat closer to the exit door and giving those passengers priority. When the model predicts an FP, the airline will incur the exact precautionary costs that we saw in the TP case. Moreover, whenever the model predicts an FN, the airline will incur unexpected corrective costs to solve the missed connection. These costs may include assigning the passenger onto a new flight and making the arrangements necessary for the extended layover, including overnight accommodation or meal vouchers.

Table 4.17 summarizes the information gathered by Guimarães:

Table 4.17: Cost structure of the different approaches.

	Basline Model	Developed Frameworks
Number of Precautionary actions	No Actions	TP & FP
Number of Corrective actions	TP & FN	FN

The following step, performed in the original research, was to consider the difference in costs between the implementation *versus* no implementation of the XGBoost model. We followed the same approach but in the case of the current scope the difference will be computed to each one of the models independently. Guimarães derived the following equations:

$$\Delta C = [C_{Pre} \times (FP + TP) + C_{Cor} \times FN] - C_{Cor} \times (TP + FN) \tag{4.1}$$

assuming a relationship between the costs of p, $C_{Cor} = pC_{Pre}$,

$$\Delta C = C_{Pre} \times (FP + TP) - p \times C_{Pre} \times TP \tag{4.2}$$

Since, as pointed in the original research, the models are intended to lower the costs with missed connections for the airline, we aim at a negative change of ΔC,

$$\Delta C < 0 \Leftrightarrow p > \frac{FP + TP}{TP} \tag{4.3}$$

Table 4.18 shows the computed results for each one of the models using Equation (4.3).

Table 4.18: Relationship between Precautionary and Corrective costs for each model.

	p_{min} threshold not tuned	p_{min} threshold tuned
XGBoost	1.25	1.27
DNN	3.14	6.04
Logistic Regression	6.14	6.22
Decision Tree Classifier	6.60	6.60

Table 4.18 indicate that the solution outputted by each one of the models is worth pursuing if C_{Cor} are at least p times greater than C_{Pre}. There is no concrete data on TAP costs, but it is reasonable to assume that corrective costs are more expensive than precautionary costs.

Overall, the results show that the tuned threshold approach produced worse results compared to the default threshold. The results worsened because the threshold tuning focused on the G-mean, meaning that the threshold value choice was based on a trade-off between classification performances on both the majority and minority classes, i.e., it focused on precision and recall. When looking to Equation (4.3), Guimarães noticed that $(FP + TP)/TP$ is equivalent to $1/precision$ as defined in Section 2.4 meaning that the ratio between costs is inversely proportional to the recall, which was not the strategy used to

tune the threshold value.

The XGBoost had the best results in terms of the easiness of possible savings for the airline since it only requires C_{Cor} to be 1.25 times greater than C_{Pre}. The remaining algorithms had worse performance than the XGBoost, but the use of the Logistic Regression or Decision Tree Classifier might still be justifiable due to their interpretability.

4.9 Results Assessment

All models were trained in similar conditions and assessed on the same test data; therefore, a qualitative and quantitative comparison is possible. In terms of accuracy, and considering the case of the default threshold value for all algorithms, the best model was, by far, the XGBoost with an accuracy score of 0.98. The DDN had an accuracy of 0.92, followed by the Logistic Regression and the Decision Tree Classifier with accuracies of 0.77 and 0.75, respectively. This accuracy rank shows that the Black Box models had a higher accuracy than the White Box models in this classification task.

However, and as stated earlier, given that this is a highly imbalanced classification problem, the accuracy is not the best metric since the algorithm might be biased towards the dominant class. A simple model that only outputs predictions belonging to the majority class would present, in terms of accuracy, a score of 0.94, which would make it better than all trained models except for the XGBoost however, this simple model would have miss-labeled all unsuccessful connections and would have had no skill whatsoever. Therefore, looking at either the Precision, Recall, or F1-score macro-average values or the ROC-AUC or PR-AUC scores is a better indicator of the model performance. Multiple metrics are essential in the current classification task because, for example, in the above-mentioned example, although accuracy was high, the ROC-AUC was low, standing at 0.5, which would have indicated that the model had no skill.

The optimal point between Precision and Recall depends on the severity of the issue in hand, in this case the costs incurred by the airline. Nonetheless, for cases with missed connections, it is advised to minimize the risk of not alerting the airline about a person that may be at risk of missing the connection by minimizing the Miss Rate or False Negative Rate:

$$FNR = \frac{FN}{P} = 1 - Recall \tag{4.4}$$

Since the model is predicting if a passenger will make a successful connection or not, the model aims to have a high Recall value, meaning that the model predicts only a smaller number of FN.

In light of this, the rank of the models by their respective macro-averaged Recall is XGBoost with 0.87; Logistic Regression with 0.76; Decision Tree Classifier with 0.75; and DNN with 0.66. When ranked according to the Recall, the initial order set by the accuracy is no longer in place, and the White Box models surpass the state-of-the-art Black Box model, the Neural Network, and shorten the distance, in terms of scores, to the best model, the XGBoost. Looking at the Recall in the minority class yields even better results in terms of the performance of White Box models since they achieve the best score

ex aequo with the XGBoost model. The results were XGBoost, Logistic Regression, and Decision Tree Classifier with Recall on the minority class of 0.75, the DNN with a Recall on the minority class of 0.36.

Regarding the models' interpretability/explainability, all approaches presented different results. In the case of the XGBoost model, the Black Box tree ensemble algorithm, the explanation results that SHAP gave differ from the feature importance results given by the built-in tool. In those cases, not only the less important features saw a change in the rank but also among the top ranks the two approaches showed different results.

In the second Black Box model, the DNN, the explanation given by SHAP was mostly coherent with the explanation given by SHAP on the XGBoost algorithm predictions, except for the importance of day-related features. The DNN was the only model among the four that did not have any form of built-in feature importance tool.

In the cases of the White Box models, the results from the models' intrinsic interpretability were mostly coherent with the explanations by SHAP. The Decision Tree based its predictions on four features, which are the same features to which SHAP outputted any level of importance, and the Logistic Regression coefficients, when ordered by their absolute value, corresponding to the order of feature importance outputted by SHAP. Only a slight deviation in the behavior was noticed in terms of the DMonthDay feature to which SHAP outputted an impact well below the incoming flight number, although the two features coefficients are not that different.

Chapter 5

Conclusions

The steady growing demand seen in the air transportation industry over the recent years has naturally been followed by a growth in aviation traffic and overcrowded airports. More investment is required in improving airline operations and passenger satisfaction in the aviation industry since it is a multi-billion dollar industry with razor-thin profit margins. For stakeholders, this means that all improvements in the business are welcomed, but at the same time, they might be unwilling to make decisions with no knowledge of the reasoning behind it whatsoever.

One of the main problems in this work was the volume of data. This leads to some issues in terms of memory and the computational power needed to train the models. Another problem was to find suitable ways to compare models since there are many different requirements in terms of performance goals. Ideally, the model should get as many correct predictions as possible while not hurting its interpretability or the associated costs. However, these three conditions have different and contradicting behaviours.

Our analysis found that getting as many correct predictions as possible might hurt the interpretability of the model, and putting too much emphasis on the interpretability might hurt the costs for the airline. As shown in Section 4.8, focusing too much on, for example, minimizing the Miss Rate on the minority class might come at the expense of degrading the performance on the majority class. While minimizing the Miss Rate on the minority class may be the correct way to proceed in terms of passenger satisfaction, ultimately, the airline profitability is affected, which is more desirable.

Section 4.9, on the other hand, showed that trying to explain a model after the training process has occurred might lead the modeller to believe in some rank of importance that can be refuted by using another tool to assess feature importance. This meets the original idea that the safest solution to certify interpretability is to use inherently interpretable models.

All models attributed the highest importance to the feature corresponding to the time planned for the connection, which is coherent with prior beliefs. Other important features common to all models were the incoming and outgoing flight codes. These two features encode a lot of information given the uniqueness nature of the flight identifiers. These features include information regarding the origin or destination airport providing their refer to the incoming or outgoing flight codes, respectively. The remaining features had mixed impacts across the different models. The feature indicating the type

of transit passenger, for example, had a considerable impact on the XGBoost and shy impact in the Decision Tree Classifier but had only a marginal impact on the remaining algorithms.

The other features, namely the ones related to the day of the connection and the class on which the passenger was traveling, had only marginal to no impact across all the models. Furthermore, the demographic features such age and gender generally had the least impact on the final predictions, and were not even considered on the Decision Tree Classifier.

5.1 Achievements

The major achievement of the present work was the introduction of model interpretability in the airline connections optimization domain. The model and explanations developed allow for predicting the likelihood of missed connections that might result in capital savings to the airline if it acts beforehand and does this while presenting the reasons for such decisions allowing for the ownership by responsible entities. We showed that despite not being globally as accurate as Black-Box models, White-Box models can still have a good performance in the minority class of an imbalance dataset classification task.

5.2 Future Work

As this is a domain of knowledge experiencing a lot of research and interest from investigators, new approaches and methods are coming up fast. As a possible extension of the current work, one could apply new state-of-the-art algorithms. One of such new approaches is called Generalized Optimal Sparse Decision Tree, an algorithm that intends to provide a general framework for optimizing Decision Trees with a guarantee of optimality.

As a compliment, one could also try to implement a rule-based algorithm other than RuleFit to assess its performance on the task and compare it with the remaining algorithms. One example could be studying the performance of the Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algorithm.

As a means to increase the performance on the different models one could also try to train independent models to each class and then make the predictions based on the output from both models but this approach has the potential to degrade the *post hoc* model explanations.

Bibliography

[1] C. Rudin and J. Radin. Why are we using black box models in ai when we don't need to? a lesson from an explainable ai competition. *Harvard Data Science Review*, 1, 10 2019. doi: 10.1162/99608f92.5a8a3a3d.

[2] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin. Learning certifiably optimal rule lists. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017. doi: 10.1145/3097983. URL https://doi.org/10.1145/3097983.3098047.

[3] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv*, 51, 2018. doi: 10.1145/3236009. URL https://doi.org/10.1145/3236009.

[4] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8), 2019. ISSN 2079-9292. doi: 10.3390/electronics8080832. URL https://www.mdpi.com/2079-9292/8/8/832.

[5] J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2:916–954, 2008. doi: 10.1214/07-AOAS148.

[6] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1:206–215, 10 2018. URL https://arxiv.org/abs/1811.10154v3.

[7] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, Oct 2019. ISSN 1091-6490. doi: 10.1073/pnas.1900654116. URL http://dx.doi.org/10.1073/pnas.1900654116.

[8] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning, 2017. arXiv: 1702.08608.

[9] C. Molnar, G. Casalicchio, and B. Bischl. Interpretable machine learning – a brief history, state-of-the-art and challenges, 2020. arXiv: 2010.09337.

[10] T. Hastie and R. Tibshirani. Generalized Additive Models. *Statistical Science*, 1(3):297 – 310, 1986. doi: 10.1214/ss/1177013604. URL https://doi.org/10.1214/ss/1177013604.

[11] M. Fasiolo, R. Nedellec, Y. Goude, and S. N. Wood. Scalable visualization methods for modern generalized additive models. *Journal of Computational and Graphical Statistics*, 29(1):78–86, 2020. doi: 10.1080/10618600.2019.1629942. URL https://doi.org/10.1080/10618600.2019.1629942.

[12] A. Zeileis, T. Hothorn, and K. Hornik. Model-based recursive partitioning. *Journal of Computational and Graphical Statistics*, 17(2):492–514, 2008. doi: 10.1198/106186008X319331. URL https://doi.org/10.1198/106186008X319331.

[13] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018. doi: 10.1109/ACCESS.2018.2870052.

[14] K. Sokol and P. Flach. Explainability fact sheets. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, Jan 2020. doi: 10.1145/3351095.3372870. URL http://dx.doi.org/10.1145/3351095.3372870.

[15] S. Mohseni, N. Zarei, and E. D. Ragan. A multidisciplinary survey and framework for design and evaluation of explainable ai systems. *ACM Trans. Interact. Intell. Syst. 1, 1, Article*, 1:46, 2020. doi: 10.1145/3387166.

[16] A. E. Eltoukhy, F. T. Chan, and S. H. Chung. Airline schedule planning: A review and future directions. *Industrial Management and Data Systems*, 117:1201–1243, 2017. doi: 10.1108/IMDS-09-2016-0358.

[17] M. M. Etschmaier and D. F. X. Mathaisel. Airline scheduling: An overview. *Transportation Science*, 19:127–138, 1985. ISSN 00411655. doi: 10.1287/trsc.19.2.127.

[18] H. D. Sherali, E. K. Bish, and X. Zhu. Airline fleet assignment concepts, models, and algorithms. *European Journal of Operational Research*, 172:1–30, 10 2006. ISSN 0377-2217. doi: 10.1016/J.EJOR.2005.01.056.

[19] B. Gopalakrishnan and E. L. Johnson. Airline crew scheduling: State-of-the-art. *Annals of Operations Research 2005 140:1*, 140:305–337, 10 2005. ISSN 1572-9338. doi: 10.1007/S10479-005-3975-3. URL https://link.springer.com/article/10.1007/s10479-005-3975-3.

[20] L. H. Lee, C. U. Lee, and Y. P. Tan. A multi-objective genetic algorithm for robust flight scheduling using simulation. *European Journal of Operational Research*, 177:1948–1968, 10 2007. doi: 10.1016/J.EJOR.2005.12.014.

[21] S. Yan and H. F. Young. A decision support framework for multi-fleet routing and multi-stop flight scheduling. *Transportation Research Part A: Policy and Practice*, 30:379–398, 1996. doi: 10.1016/0965-8564(95)00029-1.

[22] S. Yan and C. H. Tseng. A passenger demand model for airline flight scheduling and fleet routing. *Computers & Operations Research*, 29:1559–1581, 10 2002. ISSN 0305-0548. doi: 10.1016/S0305-0548(01)00046-6.

[23] S. Yan, C. H. Tang, and M. C. Lee. A flight scheduling model for taiwan airlines under market competitions. *Omega*, 35:61–74, 10 2007. doi: 10.1016/J.OMEGA.2005.03.002.

[24] H. Jiang and C. Barnhart. Dynamic airline scheduling. *Transportation Science*, 43(3):336–354, 2009. doi: 10.1287/trsc.1090.0269. URL https://doi.org/10.1287/trsc.1090.0269.

[25] S. Lan, J.-P. Clarke, and C. Barnhart. Planning for robust airline operations: Optimizing aircraft routings and flight departure times to minimize passenger disruptions. *Transportation Science*, 40 (1):15–28, 2006. doi: 10.1287/trsc.1050.0134. URL https://pubsonline.informs.org/doi/abs/10.1287/trsc.1050.0134.

[26] J. Abara. Applying integer linear programming to the fleet assignment problem. *Interfaces*, 19(4): 20–28, 1989. ISSN 00922102, 1526551X. URL http://www.jstor.org/stable/25061245.

[27] C. A. Hane, C. Barnhart, E. L. Johnson, R. E. Marsten, G. L. Nemhauser, and G. Sigismondi. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming 1995 70:1*, 70:211–232, 10 1995. ISSN 1436-4646. doi: 10.1007/BF01585938. URL https://link.springer.com/article/10.1007/BF01585938.

[28] R. A. Rushmeier and S. A. Kontogiorgis. Advances in the optimization of airline fleet assignment. *Transportation Science*, 31(2):159–169, 1997. doi: 10.1287/trsc.31.2.159. URL https://doi.org/10.1287/trsc.31.2.159.

[29] Z. Liang and W. A. Chaovalitwongse. The aircraft maintenance routing problem. *Springer Optimization and Its Applications*, 30:327–348, 2009. doi: 10.1007/978-0-387-88617-6_12.

[30] K. T. Talluri. The four-day aircraft maintenance routing problem. *Transportation Science*, 32(1): 43–53, 1998. doi: 10.1287/trsc.32.1.43. URL https://doi.org/10.1287/trsc.32.1.43.

[31] C. Sriram and A. Haghani. An optimization model for aircraft maintenance scheduling and reassignment. *Transportation Research Part A: Policy and Practice*, 37:29–48, 10 2003. ISSN 0965-8564. doi: 10.1016/S0965-8564(02)00004-6.

[32] M. Başdere and Ümit Bilge. Operational aircraft maintenance routing problem with remaining time consideration. *European Journal of Operational Research*, 235:315–328, 10 2014. ISSN 0377-2217. doi: 10.1016/J.EJOR.2013.10.066.

[33] K. L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39:657–682, 1993. doi: 10.1287/MNSC.39.6.657.

[34] I. Muter, I. Birbil, K. Bülbül, G. Şahin, H. Yenigün, D. Taş, and D. Tüzün. Solving a robust airline crew pairing problem with column generation. *Computers & Operations Research*, 40:815–830, 10 2013. ISSN 0305-0548. doi: 10.1016/J.COR.2010.11.005.

[35] M. Lohatepanont and C. Barnhart. Airline schedule planning: Integrated models and algorithms for schedule design and fleet assignment. *Transportation Science*, 38:19–32, 2004. doi: 10.1287/TRSC.1030.0026.

[36] J. J. Rebollo and H. Balakrishnan. Characterization and prediction of air traffic delays. *Transportation Research Part C: Emerging Technologies*, 44:231–241, 10 2014. ISSN 0968-090X. doi: 10.1016/J.TRC.2014.04.007.

[37] C. L. Wu and K. Law. Modelling the delay propagation effects of multiple resource connections in an airline network using a bayesian network model. *Transportation Research Part E: Logistics and Transportation Review*, 122:62–77, 10 2019. ISSN 1366-5545. doi: 10.1016/J.TRE.2018.11.004.

[38] N. Kafle and B. Zou. Modeling flight delay propagation: A new analytical-econometric approach. *Transportation Research Part B: Methodological*, 93:520–542, 10 2016. ISSN 0191-2615. doi: 10.1016/J.TRB.2016.08.012.

[39] Q. Li and R. Jing. Characterization of delay propagation in the air traffic network. *Journal of Air Transport Management*, 94:102075, 10 2021. ISSN 0969-6997. doi: 10.1016/J.JAIRTRAMAN.2021.102075.

[40] X. Fageda and R. Flores-Fillol. Airport congestion and airline network structure. *Advances in Airline Economics*, 6:335–359, 2017. ISSN 2212-1609. doi: 10.1108/S2212-160920170000006013.

[41] B. Yu, Z. Guo, S. Asian, H. Wang, and G. Chen. Flight delay prediction for commercial air transport: A deep learning approach. *Transportation Research Part E: Logistics and Transportation Review*, 125:203–221, 10 2019. ISSN 1366-5545. doi: 10.1016/J.TRE.2019.03.013.

[42] S. Bratu and C. Barnhart. An analysis of passenger delays using flight operations and passenger booking data. *Air Traffic Control Quarterly*, 13(1):1–27, 2005. doi: 10.2514/atcq.13.1.1. URL `https://doi.org/10.2514/atcq.13.1.1`.

[43] M. Guimarães. Predicting passenger connectivity in an airline's hub airport. Master's thesis, Instituto Superior Técnico – Universidade de Lisboa, Av. Rovisco Pais. 1049-001 Lisboa, 1 2021.

[44] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning From Data*. AMLBook, 2012. ISBN 1600490069.

[45] V. Bewick, L. Cheek, and J. Ball. Statistics review 14: Logistic regression. *Critical Care*, 9:112–118, 2 2005. doi: 10.1186/CC3045.

[46] J. Gentle, W. K. Härdle, and Y. Mori. *Handbook of Computational Statistics: Concepts and Methods*. Springer, Berlin, Heidelberg, 01 2012. ISBN 978-3-642-21550-6. doi: 10.1007/978-3-642-21551-3.

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

[48] C. Molnar. *Interpretable Machine Learning*. 2019.

[49] L. Breiman. Bagging predictors. *Machine Learning 1996 24:2*, 24:123–140, 1996. ISSN 1573-0565. doi: 10.1007/BF00058655. URL `https://link.springer.com/article/10.1007/BF00058655`.

[50] P. Buhlmann. *Bagging, Boosting and Ensemble Methods*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-21551-3. doi: 10.1007/978-3-642-21551-3_33. URL https://doi.org/10.1007/978-3-642-21551-3_33.

[51] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-August-2016: 785–794, 3 2016. doi: 10.1145/2939672.2939785. URL https://arxiv.org/abs/1603.02754v3.

[52] H. W. Sorenson and D. L. Alspach. Recursive bayesian estimation using gaussian sums. *Automatica*, 7:465–479, 7 1971. ISSN 0005-1098. doi: 10.1016/0005-1098(71)90097-5.

[53] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. doi: 10.1109/TAC.1974.1100705.

[54] M. Stone. Comments on model selection criteria of akaike and schwarz. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):276–278, 1979. ISSN 00359246. URL http://www.jstor.org/stable/2985044.

[55] S. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 2017-December:4766–4775, 10 2017. URL https://arxiv.org/abs/1705.07874v2.

[56] A. Bilogur. Missingno: a missing data visualization suite. *Journal of Open Source Software*, 3(22): 547, 2018. doi: 10.21105/joss.00547. URL https://doi.org/10.21105/joss.00547.

[57] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.