TÉCNICO LISBOA

Supervised Learning Methodologies to Improve Customer Support

Development of a recommendation system to help diagnose telecommunication issues

Inês Ferreira Marques

Thesis to obtain the Master of Science Degree in

Mathematics and Applications

Supervisors: Prof. Isabel Maria Alves Rodrigues
Prof. Paulo Soares

Examination Committee

Chairperson: Prof. António Manuel Pacheco Pires
Supervisor: Prof. Isabel Maria Alves Rodrigues
Members of the Committee: Prof. Maria da Conceição Esperança Amado
Eng. Jorge Dias de Sousa

December 2020

Acknowledgments

First of all, I would like to express my sincere gratitude to ADIST, *Associação para o Desenvolvimento do Instituto Superior Técnico*, for letting me be part of this research project.

I would also like to acknowledge my dissertation supervisors, Prof. Paulo Soares and Prof. Isabel Rodrigues, for their valuable guidance, support and sharing of knowledge that has made this thesis possible.

Many thanks to Border Innovation for their collaborative effort during this research and for providing me with the tools that I needed to choose the right direction and successfully complete my dissertation.

I would also like to thank my parents for their unconditional love and support and for always being there for me. Last but not least, to all my friends and colleagues that helped me grow as a person and that were there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

Abstract

Within a specific framework of a telecommunications provider, this thesis is about classifying technical issues from customer support. The goal is to develop a predictive algorithm which the technical support assistant can use as an aid to diagnose the root cause of the user's problem. This is done by applying supervised learning methodologies as it gives us the possibility of predicting likely future outcomes based on previously collected data in order to save time and resources. The dataset consists of labeled technical issues, where the predictive variables are binary and the response variable is the cause. However, there are some properties of the dataset that can hamper the learning process such as sparsity and imbalance. We look into two approaches: first we attempt several popular classifiers by brute force and second we construct a multi-step model to mitigate the effects of imbalanced class distribution. We evaluated both of these approaches not only based on common metrics such as F1-score and accuracy but also on another metric: an accuracy score that indicates whether the algorithm is able to predict the root cause within the 3 most probable causes. The first approach produced the best results with Neural Networks achieving almost 70% of accuracy. The best results for the Top 3 accuracy were around 90% also for Neural Networks. Since we are looking for a way to help the assistant, between having one choice for the cause or having a set of 3 possible causes to choose from, and where one of them is correct with 90% confidence, the second option is better for the assistant. We conclude that the dataset needs to be improved in order to reach better results but the proposed model may already be a successful classification system.

Keywords

Telecommunication; Classification; Supervised learning; Class imbalance

Resumo

Dentro do âmbito específico de uma operadora de telecomunicações, esta tese trata a classificação de problemas de suporte técnico. O objetivo é desenvolver um algoritmo preditivo que sirva de apoio ao assistente técnico para chegar à causa correta do problema. Isto é alcançado através de metodologias de aprendizagem supervisionada, uma vez que nos dá a possibilidade de prever observações novas com base em dados recolhidos previamente de maneira a poupar tempo e recursos. O conjunto de dados consiste num conjunto de casos de suporte técnico classificados, onde cada instância se refere a uma chamada do cliente a reportar um problema. As covariáveis são binárias e a variável de resposta representa a causa do problema. Neste trabalho experimentámos duas abordagens: a primeira consiste em tentar vários classificadores conhecidos e avaliar o seu desempenho, e a segunda consiste em construir um modelo de múltiplos passos para mitigar os efeitos do desequilíbrio de classes. Ambas as abordagens foram avaliadas não só com base em métricas normalmente utilizadas como *F1-score* e precisão, mas também com base numa métrica personalizada: valor de precisão que indica se a causa correta está entre as 3 causas mais prováveis estimadas pelo algoritmo. A abordagem que produziu os melhores resultados foi a primeira, onde as Redes Neuronais alcançaram uma precisão de quase 70%. Quanto à precisão das três classes mais prováveis, também as Redes Neuronais produziram os melhores resultados com um valor de 90% nas classificações teste. Como o objetivo é ajudar o assistente, entre ter apenas uma escolha para a causa ou ter 3 e uma delas estar certa com 90% de confiança, a segunda opção é preferível para o assistente. Por fim, concluímos que o conjunto de dados necessita de ser melhorado, mas que o modelo proposto constitui já um classificador satisfatório.

Palavras Chave

Telecomunicações; Classificação; Aprendizagem supervisionada; Classes não balanceadas

Contents

List of Figures

List of Tables

Acronyms

ADSL	Asymmetric digital subscriber line
API	Application Programming Interface
CPE	Costumer premises equipment
DT	Decision Trees
DTMF	Dual-Tone Multi-Frequency
FTTH	Fiber to the Home
IPTV	Internet Protocol Television
IVR	Interactive Voice Response
KNN	k Nearest Neighbours
LBFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
LR	Logistic Regression
ML	Machine Learning
MLP	Multilayer Perceptron
NB	Naive Bayes
NN	Neural Networks
ONT	Optical Network Terminal
OVA	One versus All
OVO	One versus One
POTS	Plain old telephone service
ReLU	Rectified Linear Unit
RF	Random Forest
SAG	Stochastic Average Gradient
SGD	Stochastic Gradient Descent

STB Set-Top box

SR Softmax Regression

VoIP Voice over Internet Protocol

1

Introduction

Contents

Telecommunications and entertainment are widely embedded in our society. Due to the increased progress in technology, the access to this kind of services becomes easier and the need for efficient operations and quality performances becomes critical for more and more infrastructures in today's world. With such an important role in economic development, there is a high competitive environment among all service providers in the telecommunication sector.

Telecommunication companies provide packages of services including TV, Internet, landline and mobile phone plan, etc. An important part of the user experience for any service is the assistance provided by the operator when the user needs help in what might be a stressful situation. Nowadays, the contact to support service has moved away from personal interactions with company representatives and has been replaced with an increased use of technology-based service. Therefore, it must be ensured that the customer support call center is effective and expeditious in finding a solution to the problem. As [Roos and Edvardsson, 2008] explains, telecommunication providers stopped perceiving the technical support as a separate service and started to see it as a complementary and essential service linked to the continuation of the relationships with the customers. However, for the generality of telecommunication operators, it is known that the customer support has its weaknesses and one can spend hours on the phone talking with the technical assistant on the other side of the line. This is why companies are focusing their energies to enhance the relationship with the customers by making their service quality more advanced and to stay competitive in the market by making the technical support service more efficient. Whenever current service providers do not satisfy their customers demand then their customers may start switching to other providers. Due to the highly competition between telecommunication companies, they are continually putting more efforts to improve and develop their operation of support call center in order to achieve customer's loyalty and satisfaction. So, how can the working method of the technical assistance be improved?

1.1 Context

This dissertation will be carried out under the scope of the KEEN project which is a collaboration between Border Innovation, an IT consulting firm, and CEMAT, Center of Computational and Stochastic Mathematics. We will work within a specific framework where the given data belongs to a certain telecommunications company and it is provided to us by Border Innovation. Within this framework, in order to obtain the root cause of a user's technical issue, it must be given background information, test results and a list of services and symptoms.

Whenever a user contacts the technical support call center, it will first be answered by an automatic attendant. This allows the system to set the *background* of the problem automatically by the Interactive Voice Response (IVR) system and the Dual-Tone Multi-Frequency (DTMF) signaling. The background

gives information about the client, the type of *service*, the type of technology and the equipment provider. The main services are Internet, Television and Voice, and each one encompasses a set of sub-services organized in a tree-like structure. There are also *tests* done automatically in result of a communication between the user's equipment of the service at issue and the company's servers. Their results are presented qualitatively in colors to the assistant through an interface. The tests results can come back as *error*, when something is wrong, or as *OK*, when everything is well, or sometimes they can be *inconclusive*. In conversation with the customer, the assistant records in the interface the occurring *symptoms* from which the customer is complaining. After collecting all this information, the assistant determines what he or she believes is the root cause of the problem. Then, the assistant decides if it is possible to provide a prompt solution and correction to the problem or if it is necessary an in-person service for further troubleshooting.

1.2 Problem

From all the steps described previously, the focus of this work is on the decision making part of the process, where the assistant has to find the root cause of the problem. Thus, the main question is: *How can we help the assistant properly diagnose a technical issue?*

The purpose of this Master's thesis is to develop a recommendation system that will allow the assistant to quickly find the source of the problem and consequently provide a better service to the customer. Creating a model that yields the cause of the problem with reasonable accuracy will not only improve the customer support response time - as it moves on to a faster resolution - but also increase the efficiency of the serviced. This is somehow a way of automating the decision making during a technical assistance call in order to optimize its efficiency.

1.3 Limitations

As mentioned above, this work does not generalize to every technical support line as we follow an example from a specific telecommunications operator. There are predefined sets of services, symptoms and causes from which the technical assistant is able to choose from. Additionally, in the scope of the KEEN project, we focus on supervised machine learning methodology as Border Innovation provides us with a labeled dataset.

In addition, regarding the provided dataset, each instance is classified based on the belief of the technical assistant and it is not necessarily the true cause of the problem. The human error present in the dataset can bring chaos to the model and the training data may not have the desired predictive power.

1.4 Related Work

Classification is one of the most popular topics of machine learning, with a broad array of applications. An observation by [Khondoker et al., 2013] states that recent advances in several fields and high-throughput technologies have led to an explosion of high-dimensional data requiring development of new methods or modification of existing statistical and machine learning techniques, in order to maximize the information gain from such data. The number of available methods has increased and logically there is a need for method comparisons to find the best one for different situations. Numerous publications focusing on comparative studies or applying statistical tests to compare machine learning algorithms are available in the literature, like [Chan et al., 2001], [Caruana and Niculescu-Mizil, 2005] and [Trawiński et al., 2012].

However, the focus on customer support data manipulation using machine learning techniques is not sufficiently addressed and neither is the comparison of different machine learning approaches to classify the causes of telecommunication technical issues. This work sets out to study these questions.

1.5 Outline

Following this introductory chapter is Chapter 2, that addresses the most important concepts and algorithms related to this thesis and any other relevant theoretical background. Then comes the preprocessing chapter (Chapter 3) where we describe the dataset and give the details on the changes it went through due to inconsistencies and non-compliance with certain rules specified by Border Innovation. We finish by presenting the final version of the dataset and by addressing the challenge of class imbalance. Note that the preprocessing code is private because the data collected is protected by client confidentiality. If interested, one should request to the author of this thesis to view it. The evaluation results of several machine learning algorithms trained on the final version of the dataset are presented in Chapter 4. In Chapter 5, we explain the designed multi-step model, an alternative to a single machine learning model aiming to mitigate the disparity of class frequencies, and we also comment on the results of its evaluation. For both approaches, we also evaluate the algorithms based on a customized performance metric requested by Border Innovation, that evaluates whether the models are able to output the true cause within the 3 most probable causes. Finally, in Chapter 6 we conclude this work with some final comments and an overview of the more important results, and also point to possible future work. The code related to Chapter 4 and Chapter 5 is available in my GitHub account at the following link: `https://github.com/inesfmarques/MasterThesis`.

2

Theoretical Background

Contents

This chapter gives an overview of the relevant findings from literature research on Machine Learning (ML) algorithms. The exponential growth of ML applications brings us to the question: *which method should be used for each application?* Exploring these techniques can give understanding into their usage for specific situations, and thus providing an appropriate algorithm for our case.

2.1 Machine Learning

Machine Learning is the study of computer algorithms that improve automatically through experience. The field provides automated methods that can detect patterns in data and use those uncovered patterns to predict future data or to perform other kinds of decision making under uncertainty. The term machine learning was coined in 1959 by Arthur Samuel in [Samuel, 1959], a pioneer in the field of computer gaming and artificial intelligence. There are two main types of learning approaches: *supervised* and *unsupervised*. A supervised learning algorithm learns from labeled training data and helps to predict outcomes for unforeseen data, while an unsupervised learning algorithm allows the model to work on its own to discover information or patterns and it deals with the unlabeled datasets. Here, we will look into the supervised, or predictive, algorithms.

According to [Bishop, 2007], the goal of supervised learning is to learn a mapping from inputs $\mathbf{x}$ to outputs $\mathbf{y}$, given a labeled set of input-output pairs $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{M}$ where M is the number of observations. Each input $\mathbf{x}$ is a vector of numbers called features, explanatory variables or covariates, which can consist of real-valued variables or even strings. On the other hand, the output, response or target variable can be an ordinal or a nominal variable from some finite set of possible outcomes such as gender, i.e. $y \in \{C_1, \ldots, C_K\}$ where K is the number of classes, or it can be a real-valued variable such as salary. In the first case, we have a classification problem and the latter is called a regression problem. Since the set of causes for a technical support issue is finite, we are dealing with a classification problem.

Classification problems are decision problems with a lot of real-world applications such as email spam filtering, handwriting recognition, face detection, etc. If $K = 2$ we have what it is called a binary classification. If $K > 2$ then it is a multiclass classification. We are assuming that the classes are disjoint and so each input is assigned to one and only one class.

If we assume that $y = f(\mathbf{x})$, the goal of learning, or training, is to estimate the function f given a labeled training set. Thus, the result of running a ML algorithm can be expressed as a function $\hat{f}$ which is an approximation of the unknown function f, and then it can be used to make predictions, $\hat{y} = \hat{f}(\mathbf{x})$. Generally, the set of solutions which the learning algorithm is able to choose from is called the hypothesis space. According to [Utgoff, 1986], the way a particular hypothesis $\hat{f}$ is chosen in favor of another is defined by the set of assumptions the algorithm makes, for example assuming the relationship between the features and the outcome is linear. This set of factors that influence the hypothesis selection is

called *inductive bias*. Without inductive bias, a learner cannot generalize from observed data to new observations better than random guessing. The algorithm is successful only when it is guided to make a satisfactory choice from among the available hypotheses. Not to be confused with the statistical term of bias, where the bias of an estimator is its average error for different training sets while the variance of an estimator indicates how sensitive its performance is to varying training sets. For more information on machine learning bias and statistical bias, please refer to [Dietterich and Kong,].

The ability to categorize correctly new observations that differ from those used for training is known as generalization. In practical applications, the training data comprises only a tiny fraction of all possible input vectors, and so generalization is a central goal in ML algorithms.

2.2 Regularization

Sometimes a model tries too hard to capture the noise of the training set and fails to give a good representation of the function f. This means that, while learning from such data points makes the model more flexible, the generalization error increases. According to [Goodfellow et al., 2016], "regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error". In other words, regularization can be used to train models that generalize better on unseen data, by preventing the algorithm from overfitting the training dataset. Regularization can come in many forms, for example pruning decision trees or having dropout layers in neural networks (we shall see later in more detail). In order to show how regularization can be integrated in the most simple algorithms, we will introduce the notion of regularization for regression functions, which is also called penalty.

The regression fitting procedure involves a cost function, or loss function, and $\hat{f}$ turns into $\hat{f}(\mathbf{x};\boldsymbol{\theta})$ which involves a linear combination of the input variables $\theta_0 + \theta_1 x_1 + \cdots + \theta_N x_N$ where the coefficients $\boldsymbol{\theta} \in \mathbb{R}^{N+1}$ are chosen such that they minimize the cost function. The regularization is responsible for shrinking the learned coefficient estimates towards zero, by adding a penalty term to the cost function. But why should we penalize high coefficients? If a feature occurs only in one class it may be assigned a very high coefficient by the algorithm. In this case the model will learn all details about the training set, probably too perfectly. Discouraging the coefficients from reaching large values will help to control the model's tendency to overfit or underfit. Suppose we want to approximate f as a linear combination of the features $\hat{f}(\mathbf{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_N x_N$ and that the cost function consists of the mean squared error. Then, the modified cost function takes the form:

$$J(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M}\left(y^{(i)} - \hat{f}(\mathbf{x}^{(i)};\boldsymbol{\theta})\right)^2 + \lambda R(\boldsymbol{\theta})$$

The penalty term consists of the parameter λ, that decides how much we want to penalize the flexibility of our model, and the regularization term $R(\boldsymbol{\theta})$. The purpose of the regularization term is to limit a potential growth of the coefficients, since it is usually a function of the vector space norm of the coefficients and the goal is to minimize $J(\boldsymbol{\theta})$. The parameter λ allows us to control the impact of the regularization term: when $\lambda = 0$ there is no regularization; for higher values we will have smaller coefficients, however too high values for λ can lead to underfitting. The two most common regularization terms are the ℓ_1-norm and the ℓ_2-norm (multiplied by $\frac{1}{2}$) of the coefficients, which motivates the names L1 and L2 regularization respectively. Note that it is essential that all the features are standardized because they should be put on equal footing so that regularization is the same all over the features.

According to [Bishop, 2007], the difference between the two forms of regularization comes from the fact that minimizing $J(\boldsymbol{\theta})$ consists in minimizing the mean squared error subject to the constraint $R(\boldsymbol{\theta}) \leq \eta$ where $R(\boldsymbol{\theta}) = \sum_{j=1}^{N} |\theta_j|$ for L1 regularization and $R(\boldsymbol{\theta}) = \sum_{j=1}^{N} \theta_j^2$ for L2 regularization, and for an appropriate value of η which depends on the parameter λ. The regularization coefficient estimates are the first point where the mean squared error, as a function of the coefficients, meets the constraint function. Given that the constraint in the case of L1 regularization has corners at each of the axis, the intersection may occur on an axis and when this happens many of the coefficients will equal 0 simultaneously. Therefore, the L2 regularization shrinks the coefficients for the least important covariates, very close to 0 but never actually reaching it. In other words, the final model will include all features. On the other hand, the L1 regularization may force some coefficient estimates to be exactly 0 when turning the parameter λ is sufficiently large. Thus, the L1 penalty also performs feature selection and is said to yield sparse models.

2.3 Types of models

As described in [Murphy, 2012], there are two major approaches into solving classification problems:

1. **Generative Models**: This approach explicitly or implicitly models the distribution of inputs and outputs. By sampling from this kind of models, it is possible to generate synthetic data points in the input space. If modeling the joint distribution $p(\mathbf{x}, C_k)$ directly is not possible, we first determine the class-conditional densities $p(\mathbf{x}|C_k)$ for each class C_k individually, then infer also the prior class probabilities $p(C_k)$. The Bayes' theorem is then used in the form

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}$$

to find the posterior class probabilities $p(C_k|\mathbf{x})$. The denominator can be found in terms of the quantities appearing in the numerator, from the law of total probability $p(\mathbf{x}) = \sum_j p(\mathbf{x}|C_j)p(C_j)$.

2. **Discriminative Models**: This approach directly models the posterior probabilities $p(C_k|\mathbf{x})$ using a parametric model and optimizing the parameters. In other words, to get the form of the function $\hat{f}$, we tune in the coefficients of the adaptive model, learning from the training set.

For both approaches, after having found the posterior probabilities, one generally uses the maximum a posteriori estimator to determine class membership for each new input $\mathbf{x}$. That is, the class C_k that maximizes $p(C_j|\mathbf{x})$ for $j \in 1, \ldots, K$.

On the one hand, discriminative models divide the input space into decision regions whose boundaries are called decision boundaries. On the other hand, the generative methods model the actual joint distribution $p(\mathbf{x}, C_k)$ allowing to generate new data similar to existing data. One can argue that generative models solve a more general problem as an intermediate step, which is estimating the data distribution, and do not approach the problem of classification directly. Also, if the inputs have high dimensionality, we may need a large training set in order to be able to determine the class-conditional probabilities to the point of reasonable accuracy. The article [Ng and Jordan, 2001] suggests that discriminative classifiers generally outperform generative ones in classification tasks.

2.4 Review of Classifiers

This section discusses some of the most popular multiclass classification algorithms. These techniques can be categorized into *transformation to binary* and *extension from binary*.

2.4.1 Transformation to binary

One approach to solving a multiclass classification problem is to reuse existing algorithms for binary classification. The techniques presented in this section are well established in the literature, e.g. [Bishop, 2007], and show how to reduce multiclass classification into multiple binary classification problems. The strategies are *one versus all* and *one versus one*, which basically create an ensemble of individual binary models and then merge the results into a single model that predicts in the set of all classes. Any binary classifier can be used as the basis for these methods, for example, logistic regression or support vector machines.

2.4.1.A One versus All

The One versus All (OVA) heuristic method builds k binary models, one for each output class. To this end, K new datasets are created where one individual class is put against its complement (all other classes in the model), as a binary problem, and every model is trained with its respective dataset.

Then, predictions are made by running all k binary classifiers so that they learn the probability of a new observation belonging to each class. The chosen class is the one with the highest estimated probability.

Although this approach is simple, the results highly depend on the binary classifier used. Furthermore, the various classifiers learn from imbalanced training sets. For instance, if we have ten classes with equal frequency of training data points, then the individual classifiers are trained on data sets comprising 90% negative instances and only 10% positive instances, and the distribution of the original problem is lost. One way to mitigate this is to modify the response values so that the positive class is set as $+1$ and the negative class (complement) is set as $-1/(k-1)$.

2.4.1.B One versus One

The One versus One (OVO) heuristic method builds a binary model for each pair of classes and trains it on a subset of the data containing only those two classes. Given K classes, a total of $K(K-1)/2$ classifiers are trained. Then, predictions are made by having each model assign a class to a new instance. The chosen class is the mode, that is, the class with the most *votes*. This contrasts with OVA where each classifier estimated the probability of the new instance belonging to each class.

The greatest disadvantage of this approach is that for a large number of classes it requires significantly more training time than the previous approach. Similarly, to predict new observations, significantly more computation is required.

2.4.2 Extension from binary

This section discusses adaptations of binary classifiers that can solve multiclass classification problems. In these extensions, additional parameters and constraints are added to the optimization problem to handle the separation of the different classes.

2.4.2.A Naive Bayes Classifier

This classification technique is based on Bayes' theorem. It assumes conditional independence between every combination of features given the target variable. In simple terms, it assumes that the presence of a particular feature is unrelated to the presence of any other feature given the class. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability $p(\mathbf{x}, C_k)$, as [Mitchell, 1997] mentions. Therefore, it is a generative model since it computes the joint distribution of the data points. It is known to outperform even highly sophisticated classification methods. Assuming there are N features:

$$p(C_k|\mathbf{x}) = \frac{p(C_k)\prod_i^N p(x_i|C_k)}{p(\mathbf{x})}$$

The probability of each class $p(C_k)$ is estimated by the respective relative frequency of the class in the training set. Then, we apply the maximum *a posteriori* estimation to find the posterior class probabilities - note that $p(\mathbf{x})$ is constant given the input.

$$p(C_k|\mathbf{x}) \propto p(C_k) \prod^{N} p(x_i|C_k)$$

Different types of Naive Bayes (NB) classifiers assume different distributions for the class-conditional probability $p(\mathbf{x}|C_k)$. As [Metsis et al., 2006] states, the most used distributions are:

- **Gaussian NB**: when attribute values are continuous, an assumption is made that the values associated with each class are distributed according to a Normal distribution. The data is first segmented by class and then the mean and variance of each class are computed.
- **Multinomial NB**: it is preferably used when the distribution of the dataset models the probability of counts for each of k categories in n independent trials. That is, the multinomial distribution gives the probability of any particular combination of numbers of successes for the various classes, with each class having a given fixed success probability.
- **Bernoulli NB**: it is used on data that follows a multivariate Bernoulli distribution. That is, there can be multiple features but each one is assumed to be a Bernoulli variable, hence the features have to be binary valued.

Not only are Naive Bayes models fast and easy to build, but they are also practical on large data sets since the training phase is lazy depending just on some computations. However, as Naive Bayes classifiers consider all the features to be conditionally uncorrelated, they may not learn some relationships between them. Finally, the accuracy of this type of classifiers can be reduced in problems where categories may be overlapping.

2.4.2.B k Nearest Neighbours

The k Nearest Neighbours (KNN) is a simple algorithm that classifies new observations based on a similarity measure, e.g. distance function. KNN has been used in statistical estimation and pattern recognition and, as a non-parametric technique, it is often successful in classification situations where the decision boundary is nonlinear.

The principle behind this technique is to take a predefined number k of training observations which are closest in distance to the new observation, and predict the label by a majority vote. In other words, the label of a new data point is the most common class among its k nearest neighbors. One can assign weights to the neighbors so that the closest neighbors have *heavier* votes.

How to choose the value of k?

We can find in the literature, e.g. [Bishop, 2007], that in order to select the appropriate k for our data, the KNN algorithm is ran several times with different values of k and we choose the one that reduces the number of errors while maintaining the algorithm's ability to accurately make predictions when it is given unseen data. The training error rate and the validation error rate are two metrics that help to assess on different k-values.

If $k = 1$, then each observation is simply assigned to the class of its nearest neighbor and the error rate is always zero for the training observations. However, the validation error curve starts high with $k = 1$ because there is overfitting on the boundaries. So, as we decrease the value of k to 1, our predictions become less stable.

Inversely, as we increase the value of k, the classifier is able to make more accurate predictions as it becomes more stable due to majority voting or averaging, but up to a certain point. As we can see in Figure 2.1, the boundary becomes smoother with increasing value of k. The validation error rate initially decreases and reaches a minimum. After the minimum point, it then increases with increasing k and we begin to witness an increasing number of generalization errors. It is at this point that we know we have pushed the value of k too far. Since we are taking a majority vote among classes (picking the mode in a classification problem), we usually choose to have k as an odd number to have a tiebreaker.

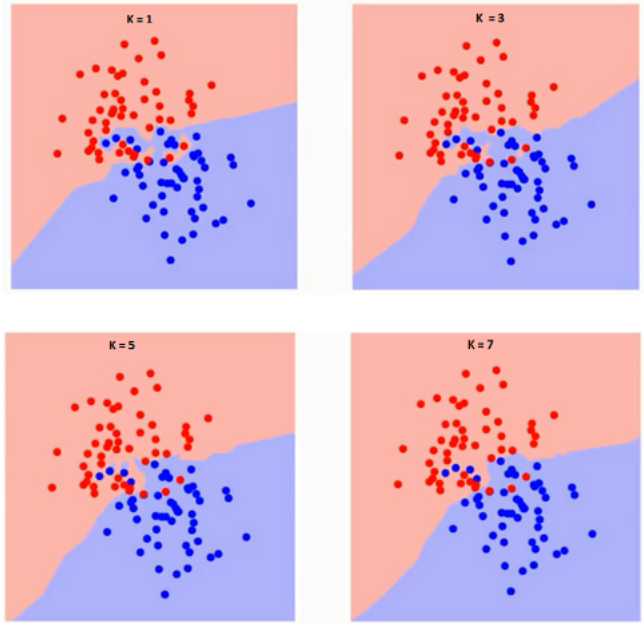


Figure 2.1: Different boundaries separating two classes with different values of k.

The algorithm is simple and easy to implement: there is no need to build a model, tune several parameters, or make additional assumptions. On the other hand, there is a curse of dimensionality. The algorithm gets significantly slower as the number of observations or predictor variables increase.

Moreover, according to [Mitchell, 1997], one practical issue in applying KNN is that the distance between instances is calculated based on all features of the instance (i.e., on all axes in the Euclidean space containing the data points). This contrasts with methods such as decision tree learning structures that select only a subset of the instance features. Therefore, if the distance between neighbors is dominated by the large number of irrelevant features it can lead to a misleading similarity metric used by KNN.

2.4.2.C Softmax Regression

Logistic Regression (LR) is a discriminative linear model for binary classification. Assuming $y \in \{0, 1\}$, the hypothesis is the probability of y being 1 given $\mathbf{x}$, also called the sigmoid function:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = P(C_1|\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

According to [Starkweather and Moske, 2011], LR assumes there are no high correlations (multicollinearity) among the covariates, which can be assessed by a correlation matrix. In addition, when selecting the relevant features to the logistic regression model, adding more covariates to the model increases the amount of variance explained in $\boldsymbol{\theta}^T \mathbf{x}$. Conversely, adding more and more covariates to the model can result in overfitting, which reduces the generalization ability of the model beyond the data on which the model is fit.

Softmax Regression (SR) is an extension of logistic regression, a type of multinomial logistic regression, used for solving multiclass classification. As the name suggests, the sigmoid logistic function is replaced by the so-called softmax function. Assuming we have K classes, our hypothesis takes the form:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \begin{bmatrix} P(C_1|\mathbf{x}; \boldsymbol{\theta}) \\ P(C_2|\mathbf{x}; \boldsymbol{\theta}) \\ \vdots \\ P(C_K|\mathbf{x}; \boldsymbol{\theta}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} exp((\boldsymbol{\theta}^{(j)})^T \mathbf{x})} \begin{bmatrix} exp((\boldsymbol{\theta}^{(1)})^T \mathbf{x}) \\ exp((\boldsymbol{\theta}^{(2)})^T \mathbf{x}) \\ \vdots \\ exp((\boldsymbol{\theta}^{(K)})^T \mathbf{x}) \end{bmatrix}$$

Here, $\boldsymbol{\theta} = \begin{bmatrix} | & | & | & | \\ \boldsymbol{\theta}^{(1)} & \boldsymbol{\theta}^{(2)} & \cdots & \boldsymbol{\theta}^{(K)} \\ | & | & | & | \end{bmatrix}$ where each column vector $\boldsymbol{\theta}^{(j)} \in \mathbb{R}^{N+1}$ denotes the parameters of our model for each estimated probability.

The model is trained via an optimization algorithm[1], e.g. gradient descent. For that we need to define a cost function that will be minimized. As [Bishop, 2007] suggests, we use the cross-entropy error function for the multiclass classification problem:

$$J(\boldsymbol{\theta}) = -\frac{1}{M} \sum_{i=1}^{M} \sum_{k=1}^{K} 1_{\{y^{(i)}=C_k\}} \log \frac{exp((\boldsymbol{\theta}^{(k)})^T \mathbf{x}^{(i)})}{\sum_{j=1}^{K} exp((\boldsymbol{\theta}^{(j)})^T \mathbf{x}^{(i)})}$$

[1] Optimization schemes come in handy when the parameters cannot be calculated analytically.

The function $1_{\{y^{(i)}=C_k\}}$ is an indicator function that it is equal to 1 if $y^{(i)} = C_k$ and 0 otherwise.

$J(\boldsymbol{\theta})$ is the average of all cross-entropy over the M training observations. Cross-entropy measure is a widely used alternative for the mean squared error when the output of the algorithm is a probability distribution. Taking the derivatives, the gradient is:

$$\nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\theta}) = -\frac{1}{M} \sum_{i=1}^{M} \mathbf{x}^{(i)} \left(1_{\{y^{(i)}=C_k\}} - \frac{exp((\boldsymbol{\theta}^{(k)})^T \mathbf{x}^{(i)})}{\sum_{j=1}^{K} exp((\boldsymbol{\theta}^{(j)})^T \mathbf{x}^{(i)})} \right)$$

Note that $\nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\theta})$ is itself a vector, so that its j^{th} element is $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_j^{(k)}}$ the partial derivative of $J(\boldsymbol{\theta})$ with respect to the j^{th} element of $\boldsymbol{\theta}^{(k)}$.

The multinomial logistic regression tends to be less effective when there are complex relationships (nonlinear) between the input variables.

Stochastic Gradient Descent

In this section, we introduce the basics of how an optimization scheme like gradient descent works. According to [Brownlee, 2014], gradient descent finds the coefficients of a function that minimize a certain cost function, using an iterative procedure. In order to find the minimum of the cost function, we compute the value of its derivative on the current coefficients. The sign of the derivative value will dictate the direction in which to move the coefficients, in order to get a lower cost in the next iteration. This technique is repeated until the cost function valued at the current coefficients is 0 or no further improvement to the cost function can be achieved. Nonetheless, gradient descent can be slow to run on very large datasets since one iteration of the algorithm requires a computation of the derivative for all the training observations. In these situations, a variation of gradient descent called Stochastic Gradient Descent (SGD) can be used. The only difference lies in the update to the coefficients which is performed based on one random data point at a time, rather than at the end of the whole training set.

2.4.2.D Decision Trees

Decision Trees (DT) are a non-parametric model that predicts the value of a target variable by learning simple decision rules inferred from the data features. As explained in [Gupta, 2017], the key idea is that the data set is broken down into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The tree tries to infer a split of the training data based on the values of the available features to produce a good generalization.

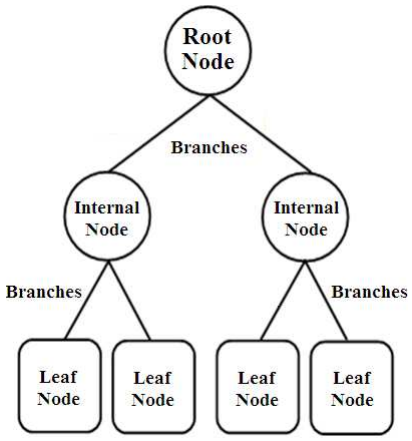


Figure 2.2: Components of a decision tree. Image from [Sá et al., 2016].

In a decision tree, each leaf node is assigned a label. The non-terminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate observations that have different characteristics. The split at each node is based on the feature that gives the maximum information gain. A new observation is classified by following a path from the root node to a leaf node, where at each node a test is performed on some feature of that instance.

How to build a Decision Tree?

In principle, the number of decision trees that can be constructed from a given set of attributes is exponential in the number features. Nevertheless, according to [Tan et al., 2005], efficient algorithms have been developed to induce a reasonably accurate, albeit sub-optimal, decision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy that grows a decision tree by making a series of locally optimal decisions about which attribute to use for partitioning the data. One such algorithm is Hunt's algorithm which is based on binary recursive partitioning the training data into successively purer subsets, that is, subsets with instances from less different classes. Suppose we run all the training points through the tree. Let D_t be the set of training observations which pass by node t. The following is a recursive definition of Hunt's algorithm:

Step 1. If all the observations in D_t belong to the same class C_t, then t is a leaf node labeled as C_t.

Step 2. If D_t contains observations that belong to more than one class, an attribute test condition is selected to partition the observations into smaller subsets. A child node is created for each outcome of the test condition and the observations in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

It should be noted that if none of the training observations have the combination of attribute values associated with a node, it is possible for some of the child nodes created in Step 2 to be empty. In this case, the node is declared a leaf node with the same class label as the majority class of training

observations associated with its parent node. In addition, if all the observations associated with D_t have identical attribute values (except for the class label), then it is not possible to split these records any further. In this case, the node is declared a leaf node with the same class label as the majority class of the training observations associated with the node t.

How should the training records be split?

The algorithm must provide a measure for evaluating the goodness of each test condition. The measures developed for selecting the best split are often based on the degree of impurity of the child nodes. The smaller the degree of impurity, the more skewed the class distribution. For example, suppose we are in a binary classification problem. If a node has class distribution $(0, 1)$ then it has zero impurity. If the node has a uniform class distribution $(0.5, 0.5)$ then it has the highest impurity. Some examples are entropy, Gini and classification error.

How should the splitting procedure stop?

A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the instances belong to the same class or all the instances have identical attribute values.

In the literature [Tan et al., 2005], we can find some **advantages** of decision trees:

- Simple to visualize and to interpret.
- Classifying a new observation is extremely fast, with a worst-case complexity of $O(w)$, where w is the depth of the tree.
- Able to handle categorical data and multiclass problems.
- Possible to validate a model using statistical tests.
- Performs well even if its assumptions are somewhat violated by the true model from which the data was generated.

On the other hand, [Gupta, 2017] enumerates some **disadvantages** of decision trees:

- It is unable to extract linear combinations of features.
- An over-complex tree can be created that does not generalize the data well (overfitting). Setting the minimum number of observations required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- If some classes dominate, the generated tree can predict poorly the minority classes. It is therefore recommended to balance the data set prior to fitting the decision tree.

- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble as we will see next.

2.4.2.E Random Forest

Random Forest (RF), as the article [Yiu, 2019] explains, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest outputs a class prediction and the most voted class (the mode) is the model's prediction.

Typically, RF corrects for decision trees' habit of overfitting to their training set because a large number of relatively uncorrelated models (decision trees) operating as a committee will outperform any of the individual constituent models. Therefore, low correlation between models is the key and the chances of making correct predictions increase with the number of uncorrelated trees in the RF, in contrast to just one decision tree. In addition, we assume that there is some predictive power in the features so that models built using those features do better than random guessing.

How to ensure that the models diversify each other? Two methods are used so that the behavior of each individual tree is not too correlated with the behavior of any of the other trees:

1. **Bootstrap Aggregation:** We know that decision trees are very sensitive to the data they are trained on - small changes to the training set can result in significantly different tree structure. RF takes advantage of this by allowing each individual tree to randomly sample from the data set with replacement, resulting in different trees.

2. **Feature Randomness:** When splitting a node, a normal decision tree selects the feature that produces the most separation between the observations in the child nodes. In contrast, each tree in a RF can pick only from a random subset of features. This forces even more variation among the trees in the model and ultimately results in lower correlation across trees and more diversification.

In conclusion, random forest leverages the power of multiple decision trees and it does not rely on the feature importance given by a single decision tree. However, not only DT are much easier to interpret and visualize but also they take less training time.

2.4.2.F Neural Networks

So far we have seen linear classifiers; bayesian and instance-based algorithms; hierarchical and ensemble methods. Now, as seen in [Bishop, 2007], we are going to look into another type of probabilistic model, the Neural Networks (NN), which is capable of expressing a rich variety of nonlinear decision surfaces. We shall restrict our attention to a specific class of neural networks that have proven to be of

greatest practical value: the Multilayer Perceptron (MLP). The functional form of a MLP is based on a linear combination of fixed nonlinear functions $\phi_j(\mathbf{x})$:

$$y(\mathbf{x}, \mathbf{w}) = h\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

where h is a nonlinear function called an activation function and the coefficients w_j in the linear combination are called *weights*. In addition, each function $\phi_j(\mathbf{x})$ is itself a nonlinear function of a linear combination of the inputs, and that is why these functions are parametric and the parameters values are updated during training.

This type of NN has a feed-forward architecture which can be described as a series of transformations between layers. There are three types of layers; input layer, hidden layers and output layer. Each hidden layer consists of units/neurons which take an input, apply the function above and return the output. The input layer consists only of one neuron for each input variable and the output layer is terminal where the number of neurons is equal to the numbers of classes. By constructing multiple layers of neurons, each of which passes on its results to the next layer, the network can learn for example nonlinear functions.

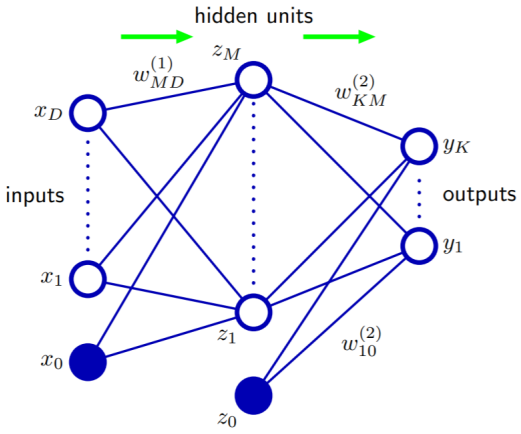


Figure 2.3: Scheme of a feed-forward neural network. Image from [Bishop, 2007].

In the case of a neural network with only one hidden layer as pictured in Figure 2.3, we first take the D input variables of an observation and construct linear combinations of the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

where j ranges from 1 to M - the number of units/outputs in the first layer - and the superscript (1) indicates the number of the layer. The term w_{j0} is called *bias*. The result a_j is transformed using a

differentiable nonlinear activation function h:

$$z_j = h(a_j)$$

The most common activation functions are the sigmoid functions, hyperbolic tangent ($tanh$) and the Rectified Linear Unit (ReLU). The outputs z_j are again used as inputs for the output layer:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

where k ranges from 1 to the number of units the output layer. Finally, the output units are transformed using an appropriate activation function to give a set of network outputs y_k. The choice of activation function is determined by the nature of the data. In the case of multiclass classification, the softmax function, σ, is used. The overall network function, assuming there is one hidden layer, is:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

In this type of networks the layers are dense meaning they are fully connected. Moreover, since MLP uses continuous nonlinearities in the hidden units (differentiable activation functions), the network function is differentiable with respect to the parameters.

The difficulty lies in determining the network parameters that fit best in the training data. To that end, we want to measure the network's output error: the difference between the desired output and the actual output of the network. As in the softmax regression, our cost function will be the multiclass cross-entropy error function and the goal is to find the vector $\mathbf{w}$ that minimizes $J(\mathbf{w})$. However, the error function will have a highly nonlinear dependence on the weights and bias parameters and therefore difficult to compute analytically. Nevertheless, it is possible to evaluate the gradient of an error function efficiently by means of the *backpropagation procedure*.

The **backward propagation** algorithm is a computationally efficient method for evaluating the derivatives of the cost function. The procedure for minimizing J is iterative with adjustments being made to the weights in a sequence of steps[2]. There are two stages in each step: first we evaluate the derivatives and then we use those derivatives to compute the appropriate adjustments to the weights. The algorithm computes how much each neuron in the last hidden layer contributed to each output's neuron error. Then, it proceeds to measure how much of these error contributions came from each neuron in the previous hidden layer - and so on until we reach the input layer. This passing scheme where information is sent backwards efficiently measures the error gradient across all the connection weights in the

[2]Note that the error function is defined with respect to a training set, and so each step requires that the entire training set be processed in order to evaluate ∇J.

network by propagating the error gradient backward in the network. The final step consists of using an optimization scheme, like gradient descent, to tune the network parameters to best fit a training set of input-output pairs.

Neural networks are very effective for high dimensionality problems and they are able to deal with complex relations between input variables or between input and output variables. There are also powerful tuning options to prevent over and underfitting. On the other hand, they are theoretically complex, difficult to implement and require some degree of expertise to tune. [Géron, 2017, Erb, 1993]

3

Data Preprocessing

Contents

This chapter presents the exploratory analysis performed on the data and it also aims to give a better understanding of the features and their relationship with the causes. The preprocessing phase is essential for model building in order to minimize the error originated by the dataset itself and not from the ML algorithm.

During the data preprocessing phase, multiple versions of the dataset were given by Border Innovation due to errors and inconsistencies, which will be given in more detail in this chapter. Moreover, all the data analysis in this work was done in R software environment for statistical [R Core Team, 2020]. In order to maintain data confidentiality, the R markdown notebook designed for the data preprocessing cannot be made public. Therefore, if the reader is interested in taking a look at the code, it should be requested to the author of this thesis.

We first give an insight on the general format of the dataset, which stayed the same throughout all the updates. Next, we present the steps for exploratory analysis and data cleaning and also expose the many difficulties we faced along the way until we reached the final version of the dataset. Then, we present what will be the input for the machine learning algorithms. Finally, we address an important issue with the dataset: class imbalance.

3.1 Description of the dataset

The dataset provided by Border Innovation contains real cases from a telecommunications company technical support. It consists of a `csv` file that can be converted to a table of the form *observations* $\times$ *features*. The columns are divided into the response variable and the predictor variables. As mentioned in the first chapter, the features are divided into Background, Tests, Services and Symptoms - which together make up the information needed to solve the technical issue. The response variable is an integer that encodes a specific cause and the predictor variables are binary where each entry indicates the presence or absence of the feature.

The input data was originally categorical but the covariates are represented in dummy encoding, which is a process of converting categories into numbers. According to [Géron, 2017], there are two main approaches in handling categorical attributes: label encoding and *OneHot* encoding. While the first assigns each category an integer value, the second creates an additional dummy binary variable for every unique value in the categorical variable. When label encoding is performed, an ordinal relationship for the variable is imposed, meaning one assumes that two nearby values are more similar than two distant values. In cases where that relationship makes no sense, *OneHot* encoding is used and the result will be a sparse matrix of *OneHot* vectors where only one entry will be equal to 1, while the others will be 0. However, this approach leads to a dependency between the independent features (multicollinearity). This problem is known as the dummy variable trap and the solution is to drop one

of the dummy variables so that the respective category will become the one that is assigned all zero values, called the *baseline*. This way we remove the extra degree of freedom and it results in what is called in [Zheng and Casari, 2018] as dummy encoding.

The background variables describe the characteristics of the service in question and of the client. The tests can have binary or categorical results, which sets how many columns are associated with a test. In practice, the binary tests are categorized into `error` and `ok` and they triggered in accordance with the type of problem. That is, they are only done when something related to the test is wrong. However, there is a level of ambiguity in having the value 0 on both columns: it can mean the test result is inconclusive or it can mean that the test wasn't even done. Motivated by this, some binary tests were converted to categorical tests, and so the tests are mostly categorical. They are associated to a triplet of columns that encode the categories `unknown`, `error` and `ok`. The baseline category, when all three variables are 0, means that the test was not done. Moreover, the tests variables are named as Dx with x being a positive integer, for example D83, with sequential numbering.

Finally, the main services are Internet, TV and Voice. The relations between services and symptoms are described by a tree with 4 levels. The first two levels refer to the services (main and secondary) and the other two levels refer to the possible symptoms for each combination of services. Each branch can have at most two services and two symptoms. The combination of services and symptoms variables in each observation in the dataset is associated to a branch of that tree. That is, there is a predefined set of combinations for this part of the predictor variables. In addition, the services and symptoms variables are named as SERx and SYMx respectively, with x being a positive integer with sequential numbering.

3.2 Data Cleaning

Data cleaning routines make the data "clean" by handling missing values, identifying or removing outliers, and resolving inconsistencies. Any model learned from a "dirty" dataset can result in unreliable output.

The initial dataset had 192684 rows and 582 columns. After loading the data, the first step was to look for missing values and one service covariate had 17056 missing values. We were told by Border Innovation that the missing values should be treated as the absence of the service in question since it was a bug in the system that keeps all of the information, and they were replaced with 0.

The next step was to check whether each variable had the correct domain type and range. In the same variable where missing values were found, instead of 1 and 0, there were 1.0 and 0.0 - meaning that instead of binary values there were doubles. The solution is to convert everything to integers.

Since we are dealing with binary-valued data, we also looked for constant columns and observed that there were 163 variables always equal to 0 but none always equal to 1. In the process of understanding the reason behind this, we were told that some variables were outdated and others were missing due to

the dataset not being in sync with the latest version of the system.

After an update on the arrangement of the data, the new version had 131434 rows and 586 columns. The previous analysis was repeated and we tried to find a pattern of the observations with missing values - now only 127. The only significant observation was that all causes were either *Common Failure* or*Internet Common Failure*. However, the total number of instances with either cause is very high compared to the number of instances with the missing values, and so that observation becomes meaningless. More importantly, there were services and symptoms variables with non-binary values: 133 values equal to 2. We were told that this was also a bug and that they should be replaced with 1.

Furthermore, there were still 124 constant columns equal to 0. This might be due to the fact that the variables are in *OneHot* encoding which means there is a "binarization" of the categories. When some categories are not present in the observations that leads to a lot of zeros present in the dataset, and consequently null variables. Large datasets can cause problems with regards to space and time complexity. Although learning in such high-dimensions can be limited, there are methods for dimensionality reduction like Principal Component Analysis (PCA) - introduced by Karl Pearson in [Pearson, 1901] - that are used for transformation in a lower dimensional space. However, according to [Feldman et al., 2016], sometimes these dimensionality reduction techniques have no performance guarantees in terms of memory used, e.g. in sparse datasets which is our case or with non-continuous features. Nevertheless, there is a popular method for dimensionality reduction of binary data called Logistic PCA. R has a package that implements the method of [Landgraf and Lee, 2020], which is an extension of Pearson's initial formulation of PCA. However, when approximating each data point in a two-dimensional latent space and trying to separate the classes, the resulting plot is very difficult to interpret as we can see in Figure 3.1. Therefore, we chose not to apply feature selection and dimensionality reduction techniques due to the large number of classes in this problem.

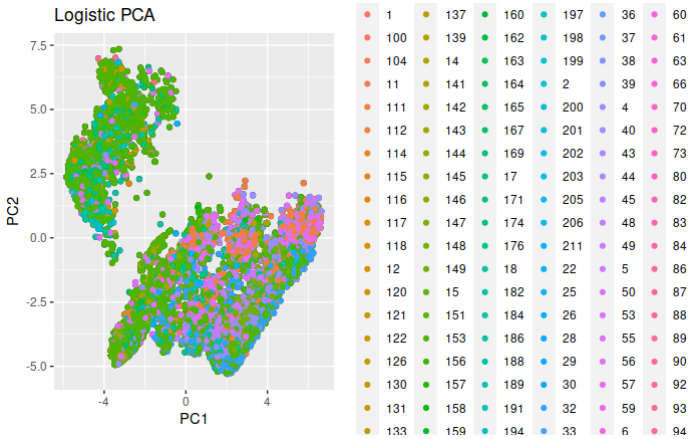


Figure 3.1: Plot of the Principal Components scores colored by their respective class.

Following this preliminary analysis, we looked for inconsistencies in each of the four groups of the

predictor variables. Before we begin, there are a few useful telecommunications acronyms in the context of this problem and relevant for the analysis that follows: Fiber to the Home (FTTH), Plain old telephone service (POTS), Asymmetric digital subscriber line (ADSL), Voice over Internet Protocol (VoIP), Costumer premises equipment (CPE),Set-Top box (STB).

3.2.1 Scenario checking

Regarding the background scenario, the dataset should conform to some rules for it to be coherent. The following table shows the verification of those rules where the first column describes the rule and the second column shows the number of observations that don't comply to it:

Rule	**Violations**
Either the client is private or business	2633
If the service tech is Mobile, all other service related attributes are 0	0
If FTTH$= 1$ then POTS$= 0$	0
If ADSL$= 1$ then FTTH$= 0$	0
If VoIP$= 1$ then the service is Voice and not Mobile	14469
If POTS$= 1$ then the service is Voice and ADSL$= 1$	1034

Table 3.1: Scenario rules verification

This non-compliance was reported to Border Innovation so that it would be investigated. For instance, taking into account that the total number of observations is 131434, the cases of the last rule but one consist of more than 10% of the dataset. Border Innovation informed us that these issues were also due to the use of old technical support software and a new corrected dataset would be generated. As the new version of the dataset was expected to comply with the scenarios rules, we continued analyzing the dataset.

3.2.2 Tests checking

We verified possible relations between binary test variables pointed out to us by Border Innovation, where either they appeared to be testing the same function or they were somehow correlated. The following table presents the results.

Test	Description	Conflicts
D38 = D39?	TV - activation operation in progress	2
D40 = D44?	TV channel package supplied	0
D42 = D43?	Inconsistency between supplied TV package in Siebel and in Media Room	0
D49 = D50?	TV - supply error or not supplied	275
D56 = D57?	Loop - both cables cut in the center	198
D336 = D337?	CPE not synced	4298
If D$70 = 1$ then D69 =D$71 = 1$?	Loop - Low downstream and upstream noise margin	70
If D$73 = 1$ then D72 =D$74 = 1$?	Loop - Current and voltage insufficient for analogical phone power	53

Table 3.2: Tests associations

The second and third verifications suggest that there may be some repeated columns. Furthermore, the four last verifications are concerning. On the one hand, each pair of tests on the three verifications before last are supposed to test the same function but show differences. On the other hand, D70 is supposed to be the conjunction of D69 and D71 as well as D73 is supposed to be the conjunction of D72 and D74. The explanation provided is that these faults are the result of several overlaps of *states* in an observation. This means that when a client calls and a new SESSIONID is created, a full initial battery of tests is ran. However, as the assistant interacts with the client, some of the tests are rerun and only those results are updated in the system. This way, an incoherent state of the data is generated due to contradictions between variables that test the opposite and have the same result or that test the same but have opposite results. This would also be investigated and a new version of the dataset would result from solving this problem.

3.2.3 Services and Symptoms checking

On the new version of the dataset, besides running all the previous data analysis, we also checked whether every observation had at least one service and one symptom and at most two of each. Note that for the model to be accurate and approximate reality, observations without services, without symptoms or without both do not make sense and bring chaos to the model. Fortunately, every observation had at most two services and two symptoms. However, there were 32997 instances without any symptoms. Not having information on the symptoms of the issue means that the label is based solely upon test variables. This combined with inconsistencies still found with the previous analysis would yet result in another version of the dataset.

3.2.4 Other findings

In the course of the data preprocessing, we were asked by Border Innovation to look for anything out of the ordinary that could show more inconsistencies in the dataset. This time we focused on the labels and found two that appeared to having no predictive pattern: 59 (Internet Protocol Television (IPTV) service inactive or suspended) and 144 (STB not authenticated). From all the test variables related to the first case, some of which somehow indicate the presence of an error in the IPTV service, one of the variables specifically tests if IPTV not provisioned or inactive. When checking if any of those variables were present in the observations with label 59, the result was negative: no signs of errors with IPTV service. In addition, there is another test variable that is positive if the IPTV is provisioned and it was present in all those observations. Secondly, we took all the test variables related to the second case that somehow indicates the presence of an error with the STB or that it is not authenticated. When checking if any of these variables were present in the observations with label 144, only one in a total of four observations had one of the test results positive. These findings show that nothing would suggest a pattern of error for those causes or a correlation between a group of tests and the causes. This situation hints at a poor profiling of the causes. That is, for some labels there may not be concrete evidence for them in the data which means that it may be hard to predict future observations with those causes.

3.3 Final version of the dataset

The last version of the dataset provided by Border Innovation has 65972 observations and 567 variables. The first column corresponds to the SESSIONID of the observation and the second column to the response variable. For modeling purposes, the first column is irrelevant and it was removed. Thus, there are 565 predictor variables and one response variable. Although there are no missing values, there are 2173 non-binary values on the covariates. Their values vary between 2 and 22 and they were all replaced with 1. There are still 136 constant columns equal to 0 and two secondary services are absent: share center and OneNet VoIP. In addition, the sparsity of the dataset is approximately 95%.

The scenario testing table for this dataset is the following:

Rule	**Violations**
Either the client is private or business	38
If the service tech is Mobile, all other service related attributes are 0	0
If FTTH$= 1$ then POTS$= 0$	0
If ADSL$= 1$ then FTTH$= 0$	0
If VoIP$= 1$ then the service is Voice and not Mobile	0
If POTS$= 1$ then the service is Voice and ADSL$= 1$	1

Table 3.3: Final scenario rules verification

We observe that almost all the conditions are verified. The last case can be considered invalid and was removed from the dataset.

One of the updates on the dataset was the removal of useless test variables and the reorganization of the remaining ones, namely the numbering. Many of the previous queries no longer apply since repetitions of some tests were deleted. That is why the following table shows different and fewer tests correlations.

Test	Description	Conflicts
$D43 = D44$?	TV - STB not compatible with CPE	2
$D45 = D46$?	TV - STB compatible with CPE	366
$D54 = D55$?	Loop - Both cables cut in the center	50
If $D64 = 1$ then $D65 = D66 = 1$?	Loop - Low downstream and upstream noise margin	4

Table 3.4: Final tests associations

The first three queries show that tests which supposedly check the same function have different values in certain instances. However, further analysis showed that $D43$, $D45$ and $D54$ are all null variables. This suggests that these pairs of test variables are repetitions of the same test where one is no longer active. The four instances of the last case can be considered invalid and were removed from the dataset. Furthermore, we found that three test variables were missing even from this final version and we were told by Border Innovation to ignore the issue since it would probably correspond to three more null columns.

There are only 67 cases without symptoms. We tried unsuccessfully to find a pattern in those observations, namely within the labels, and so the decision was made to consider them invalid and to be removed from the dataset. Hereafter, the dataset used for model fitting will be the final version cleaned of invalid observations, thus with 65900 observations.

3.4 Data imbalance

Although data cleaning helps to prevent the model from misbehaving, another challenge arises. In classification problems, a sufficiently representative number of observations of each class is desirable for a model to be able to learn how to better separate the classes. That is why imbalanced classifications pose a difficulty for predictive modeling as most of the ML algorithms used for classification are designed around the assumption of an similar number of observations for each class. This results in models having a poor predictive performance, specially for the minority classes.

In total, there are 217 possible causes and only 114 are present in the dataset, which means that approximately 47% of the labels are not represented. Figure 3.2 shows a summary of the class distribution of the dataset:

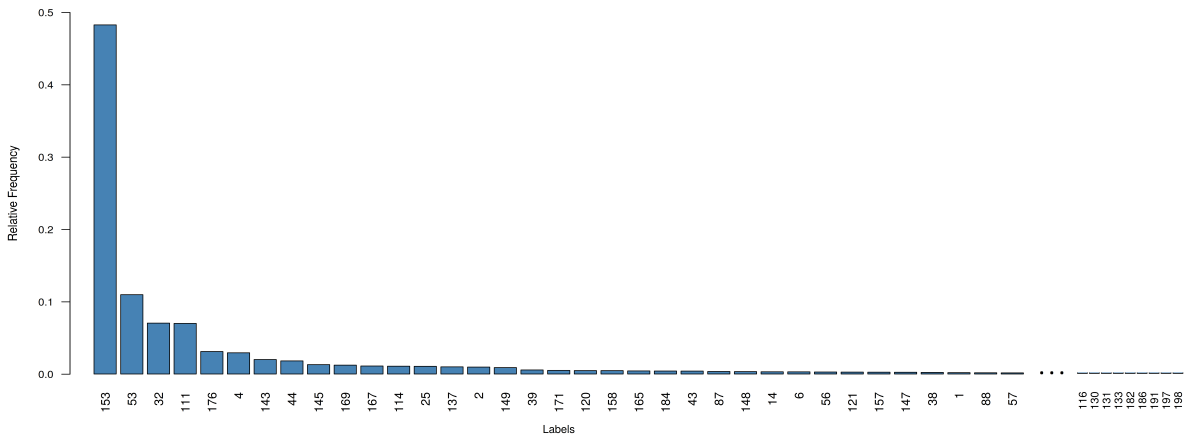


Figure 3.2: Relative frequencies of the causes

It is clear that the class distribution is severely imbalanced. The label 153, which corresponds to *Undetermined Cause*, holds almost half of the data points. Thus, the other half of the data points are allocated to the remaining 113 present classes. In addition, the frequencies of the majority of the labels are almost negligible in the barplot, roughly beyond the 16 more frequent classes. In the context of telecommunications customer support, it is comprehensible that there are causes more frequent than others since some technical problems are more likely to happen than others. However, one factor that possibly aggravates the disparity in class frequencies is the fact that the response variable in the dataset is determined by the technical assistant. In other words, what the assistant considers to be correct is the absolute truth. The labeling of the observations is based on the classification made by the assistant without any verification, which can lead to considerable human error. From this, it follows that the imbalance is a property of the problem domain - as some causes are more frequent - and the information that the dataset comprises may not be accurate by measurement error. Hence, taking the models we saw in the previous chapter, it is possible that, if the classes are not well separable, several of those models will generalize poorly as they will probably overfit. The models may have bias towards classes which have higher number of instances and assume that most technical issues are caused by those causes, without being the algorithm's fault but because of the properties of the data itself. The models tend to only predict the majority class and treat the instances of the minority class as noise and often ignore them. Thus, there is a high probability of misclassification of the minority class as compared to the majority class. However, as mentioned earlier, the fact that a model predicts better the majority causes may not be a significant problem since it will predict correctly more often. In other words, since theoretically the frequency of the classes is proportional with their likelihood of occurrence, it may be a good thing to focus more on the majority causes than on the others. Nevertheless, in addition to a model selection based on typically used metrics in ML, Border Innovation asked us to evaluate the models

based on whether the correct cause is in the top 2 or 3 causes predicted by the model. We will call this measure Top 3 accuracy.

4

Brute Force Approach

Contents

This chapter describes the steps in choosing a ML algorithm by experimenting the various techniques seen in Chapter 2, since it is hard to guess which one will perform better. We will directly train them with the final version of the dataset, compare the results and choose the one with the best generalization score. Hereafter, all the computations related to modeling are done in the Python programming language, more specifically in a Google Colab Notebook using a hosted runtime in the Google Cloud. The notebooks run by connecting to virtual machines that have maximum lifetimes that can be as much as 12 hours, so one has to bear in mind that one continuous computation cannot last longer. Scikit-learn was the main library used for model fitting, model selection and evaluation. It is an open source Python library and all the documentation about the built-in machine learning tools used in this chapter is available at [Developers, 2020]. The code for this chapter is available at GitHub in the file `Model.ipynb`.

For the purpose described above, we need methods for estimating the generalization performance, a process known as *model evaluation*. We begin by introducing not only some basic and important performance measures for multiclass classification but also the approach used in this work to evaluate the performance of each algorithm. Finally, we present the results of fitting the models on our dataset and select the best one based on the evaluation metrics.

4.1 Model Evaluation

There are many possible classification models with different levels of complexity that can be used to capture patterns in the same training data. Since we are interested in the performance on unseen data, we must evaluate the model on a separate set not used for training and select the one that shows the highest generalization score. One approach is to partition the dataset into training data, validation data and test data. According to [Tan et al., 2005], the validation set is used to tune parameters of the algorithm or other regularization by predicting new observations with the already fitted model. On the other hand, the test set provides an unbiased evaluation for the model and it is used to obtain performance metrics.

4.1.1 K-Fold Cross-Validation

Cross-validation is a gold standard in machine learning for estimating performance metrics on unseen data. The dataset is randomly divided into K groups, or folds, of approximately equal size. One run of cross-validation involves one of the folds being set as the validation set while the model is fitted on the remaining $K-1$ folds, which make up the training set. After obtaining the evaluation scores for K runs (so that each fold is set as the validation set once), the final result is often the average of the scores. This approach generally results in less biased or less optimistic estimates.

The choice of K is usually between 5 and 10, but there is no formal rule. A small value of K will

result in a smaller training set at every run, which will lead to larger generalization error rate than what is expected of a model trained over the entire set. On the other hand, a high value of K results in a larger training set at every run, which reduces the bias in the estimate of generalization error rate. However, according to [Tan et al., 2005], the higher the K the more computationally expensive it is to run cross-validation, especially for large datasets.

4.1.2 Performance Measures

In order to assess a classifier performance, we must use an appropriate metric. According to [Luckert and Schaefer-Kehnert, 2016], although there is no perfect indicator for every matter concerning evaluation of machine learning algorithms, the most popular metrics are the following:

- **Accuracy** is the fraction of correctly classified data. If y_i is the actual label of data point i and $\hat{y}_i$ is the model's prediction, the accuracy is defined as

$$\frac{1}{M}\sum_{i=1}^{M} 1_{(\hat{y}_i = y_i)}$$

  The main problem with this measure is that the results highly depend on the class distribution. If the model only predicted the same class for all observations and that class happened to hold the majority of the observations, the accuracy would be high but the model would have no predictive ability. That is why accuracy is not a sufficient metric for imbalanced datasets. Furthermore, using training accuracy as the sole criterion for model selection is not reliable because a high value can be associated to overfitting.

- **Precision** is, for a certain class C_k, the proportion of observations which the model classifies with C_k and that are labeled as C_k. If being classified as positive means being classified with C_k, then:

$$precision = \frac{TruePositives}{TotalPredictedPositives}$$

  where True Positives is the number of observations that are correctly classified as positive and the Total Predicted Positives is the number of all observations predicted as positive. For example, a precision value of 1 means that every observation classified with C_k by the model has been correctly classified. However, it is important to note that this doesn't give any information on the amount of observations classified with some other class C_i when the actual label was C_k. Precision is a good measure to determine when the costs of wrongly classifying as positive (false positive) is high.

- **Recall** is, for a certain class C_k, the proportion of observations that are labeled as C_k which are

correctly classified by the model.

$$recall = \frac{TruePositives}{TotalActualPositives}$$

where True Positives is the same as before and the Total Actual Positives is the number of all observations labeled as C_k. For example, a recall value of 1 means that every observation with label C_k has been correctly classified. Note that this can be easily achieved by classifying every data point with C_k. Recall is a good measure to determine when the cost of wrongly classifying as not positive (false negative) is high.

- The F1-score combines both recall and precision. It is the harmonic mean between the two:

$$F1 = 2\,\frac{precision \times recall}{precision + recall}$$

- The confusion matrix, in turn, illustrates the performance of ML algorithms. In the multiclass case, it is a $labels \times labels$ table whose i^{th} row and j^{th} column entry indicates the number of observations where the actual label is C_i and the predicted label is C_j.

According to [Murphy, 2012], in multiclass classification, both precision and recall are determined for each class by having the positive label as that class and the negative label as all the remaining ones. In order to get an overall F1-score of a multiclass classification model, one usually averages the F1-scores over all classes. This average can be *macro*, where each class has equal weight and so it results in the simple arithmetic mean. The average can also be *micro*, where each observation has equal weight meaning we look at all the samples together. In this case, a prediction error is both a false negative for one of the classes and a false positive for the other and thus the proportion of prediction errors will be equal to the accuracy. Finally, the average can be *weighted*, where we weight the F1-score of each class by the number of observations from that class.

4.2 Model Comparison

Not every algorithm we have seen in Chapter 2 should be considered for our solution - we should restrict our set of hypothesis based on the assumptions we make. The intended recommendation system must consist of a probabilistic algorithm in order to provide the posterior probabilities of the classes to calculate the Top 3 accuracy. In addition, bearing in mind what we have seen in Section 2.4.1, we will discard the OVA heuristic because of the imbalanced class distribution and the OVO heuristic because of the large number of classes - it would become computationally expensive. Moreover, we will rule out the KNN approach due to the size of the dataset, justified by the curse of dimensionality referred in Section

2.4.2.B. Finally, referring to Section 2.4.2.E, we will not try to evaluate a Decision Tree since not only a Random Forest is more stable than a single DT but it also limits overfitting and it can generalize over the data in a better way. Therefore, we will compare between Naive Bayes, Softmax Regression, Random Forest and Neural Networks.

The models' parameters were tuned by performing a grid search with a 3-fold cross-validation in order to determine the best combinations of parameters based on the overall weighted F1-score. As discussed in Section 3.4, the reason why we opt for the weighted F1-score is because Border Innovation asked us to focus on the more frequent causes as it is more important for the recommendation system to predict correctly more often the dominant causes than to predict correctly every cause. So we must consider the proportion for each label in the dataset. The choice of the number of folds derives from the fact that our dataset is large enough that the training set is still representative whilst having a relatively low K. We employ the Python method `GridSearchCV` where the split of the folds is stratified which means they are selected in order to preserve the percentage of observations for each class.

Moreover, in order to achieve reproducible results in ML algorithms we must use the exact same code, dataset and sequence of random numbers. The reason why one may not get the same results when running the same script on the same training data is because certain ML models make use of randomness. Therefore, getting control over non-determinism of our experimentation process is crucial. The random numbers are generated by a pseudo random generator. This generator is parametric and deterministic, meaning that for the same input it will always output the same sequence of numbers that are random enough for most applications. This input value is called **seed** and if for a fixed seed we obtain the same sequence of random numbers for each algorithm we compare and each technique we try. Note that the variations of results for different seed values are within a range. This means that the performance metrics may vary within a small range for different seeds but in principle there shouldn't be a significant difference. Even when sampling the data to partition it into training set and test set we used the same seed so that the training set is always the same for every model we fit.

We split the whole dataset into training set (80%) and test set (20%), in a stratified fashion. After performing cross-validation on the training set, we fit each model with the best combination of parameters one more time to compute performance measures on the test set (precision, recall and the weighted F1-score) as well as to visualize the confusion matrix. This way we can have an ultimate perception of the model's performance.

Naive Bayes

As seen before, we must choose the type of NB classifier based on our dataset. Since the features are binary, we will first assume it follows a multivariate Bernoulli distribution and so we opt for the `BernoulliNB` estimator. The grid search combined with 3-fold cross-validation was relatively fast since

the only parameter to be tuned was the additive smoothing parameter[1], `alpha`. To help find the best value, we plot a validation curve which shows the influence of the parameter on the training and validation data. Validation curves are used for parameter tuning where we plot the evolution of a metric, in this case the weighted F1-score, against different values of a parameter.

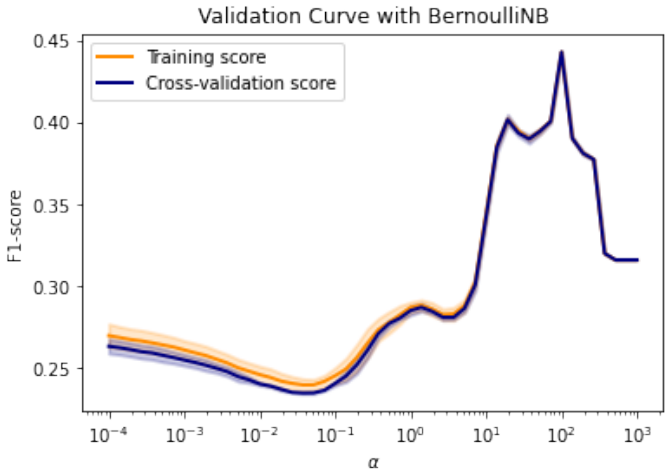


Figure 4.1: Validation curve for parameter tuning with `BernoulliNB`

We observe that there is no linear influence of the additive smoothing parameter on the model, with the plot having a lot of variation. Nevertheless, we can see there is a peak of the scores and the best `alpha` is 100 with the overall weighted F1-score being around $44,3\%$. This poor result led us to try assuming another distribution of the data, namely multinomial distribution. The scikit-learn library provides a classifier based on Multinomial NB that is designed to correct severe assumptions which make it suitable for imbalanced datasets: the `ComplementNB`. Below, we show the validation curve with this estimator.

[1] The additive smoothing parameter is used to smooth categorical data

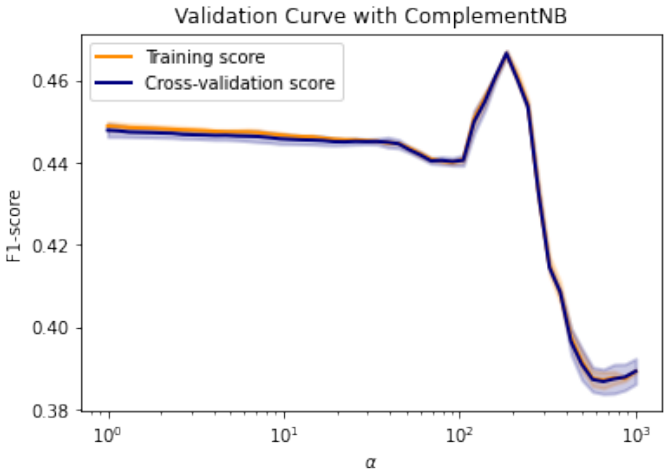
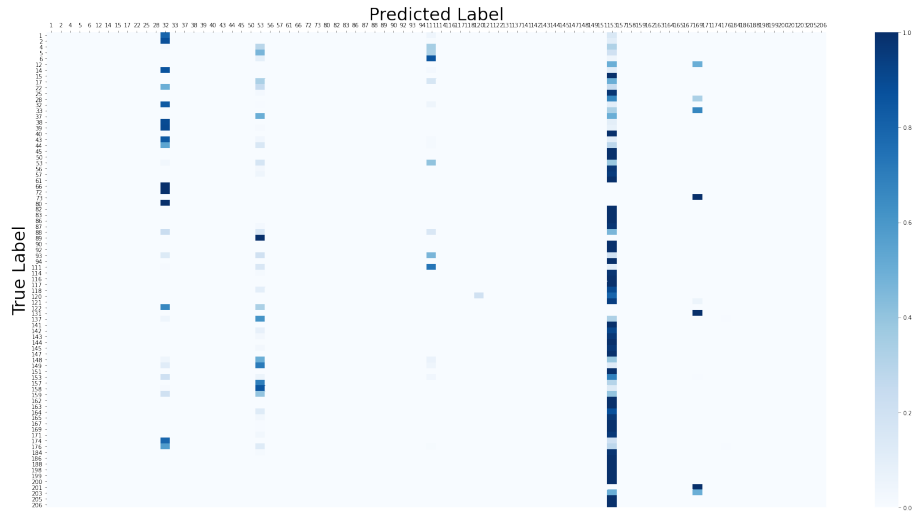
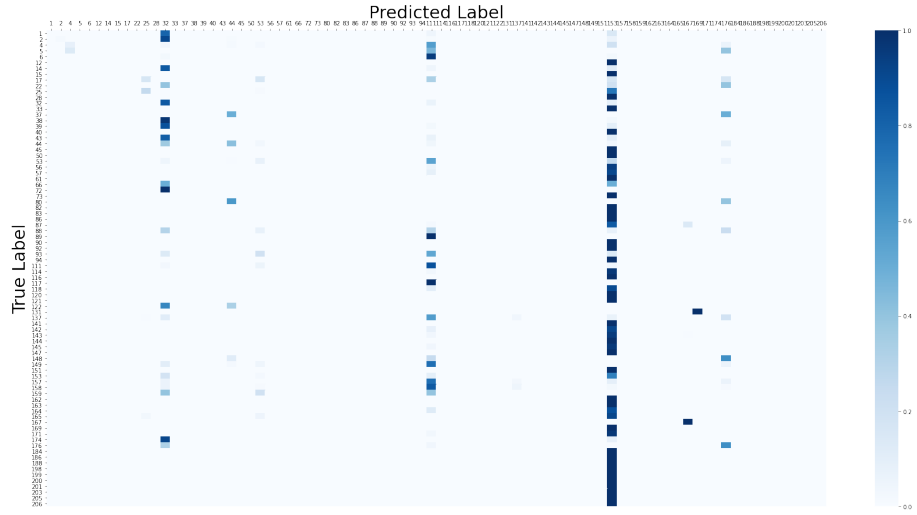


Figure 4.2: Validation curve for parameter tuning with `ComplementNB`

We see that the lines are more stable and there is also a peak of the scores before decreasing considerably. The best `alpha` is approximately 184 with the overall weighted F1-score still being low, around $46,7\%$. Finally, we present the confusion matrices for both estimators in order to visualize and compare their performance on the test set, which represents 20% of the dataset. Note that these confusion matrices are normalized where each entry (i, j) is the number of observations with true label i classified with label j divided by the total number of observations with true label i. This way we are able to visually interpret how the labels are being predicted and compare between them since we are dealing with percentages of correctly classified observations.



(a) `BernoulliNB`



(b) `ComplementNB`

Figure 4.3: Confusion matrices of the estimators with the best score, for both built-in functions

We observe that there are four columns that stand out correspond to the four most frequent classes in the dataset. In particular, the column with more blue entries is associated to the class 153 which is the most common cause. This is evidence of what was mentioned in Section 3.4: the models are too closely fit to the training set and not having predictive power for the minority classes. Furthermore, the two matrices are very similar and the only slight difference is that `ComplementNB` has a few more colored entries on the diagonal which is a sign of correct classifications.

Softmax Regression

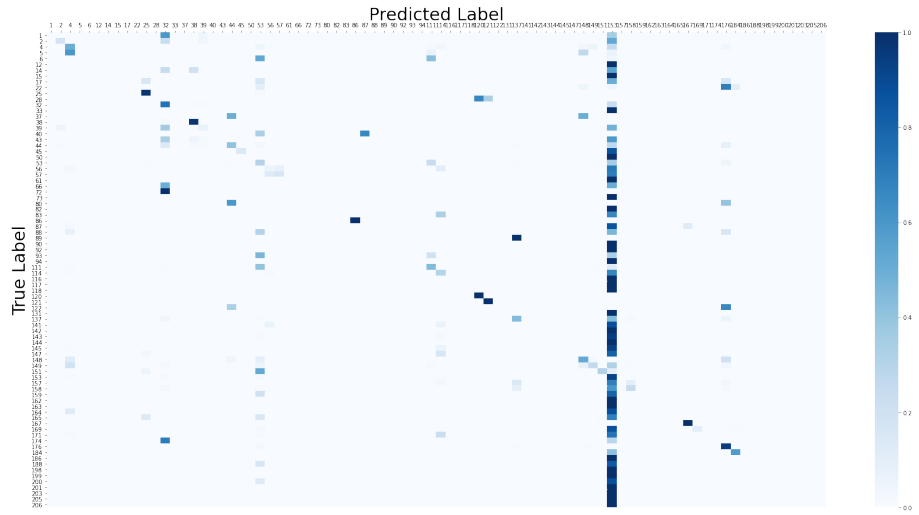
The built-in classifier that implements SR is the `LogisticRegression` with the parameter `multi_class` set to multinomial. However, this method does not have the option of SGD as the weight optimization solver, but instead it is able to learn from other gradient-based solvers like Stochastic Average Gradient (SAG), which is a variation of SGD with faster convergences rates[2]. Another alternative is the `SGDClassifier` but it only supports multiclass classification by combining multiple binary classifiers in an OVA scheme. Setting the `loss` parameter of this method to *log*, the binary classifier corresponds to the logistic regression and it implements an OVA scheme of logistic regression classifiers. Given this, we will compare the performance between `LogisticRegression` and the `SGDClassifier` with the parameters set to the values mentioned above.

The grid search with cross-validation on `LogisticRegression` looks for the best combination of parameters based on the weighted F1-score. Our choice of optimization `solver` is between the SAG and the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) algorithm, which belongs to the family of Quasi-Newton methods[3] and it is the default solver for `LogisticRegression`. The `penalty` for both of these techniques must be the L2-regularization but we will try different values for the inverse of the regularization parameter, `C`. The best score was $60,8\%$ for the model implementing the LBFGS algorithm and with `C` close to the default value which is 1.

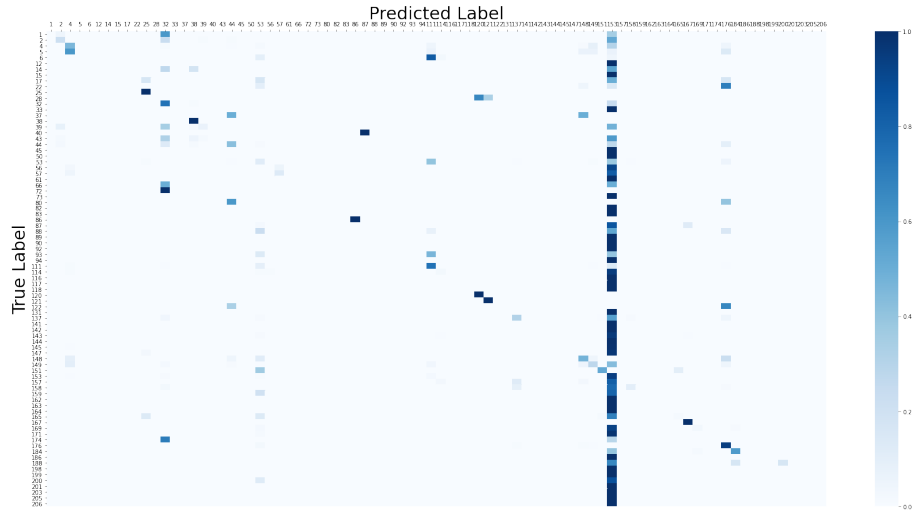
For the `SGDClassifier` with cross-entropy loss, we can opt for L1-penalty or L2-penalty and also search for the best regularization parameter, `alpha`. The best combination was the L1-penalty with the regularization parameter approximately 10^{-4}, yielding an average of overall weighted F1-scores of $59,5\%$. In order to better visualize the difference between the two classifiers, we show the confusion matrices for the two best classifiers:

[2]Please refer to [Schmidt et al., 2013] for more information on SAG

[3]Quasi-Newton methods are an alternative to Newton's method. They can be used when the Hessian matrix is difficult or time-consuming to evaluate, in order to find local minimum of a function



(a) `LogisticRegression`



(b) `SGDClassifier`

Figure 4.4: Confusion matrices of the estimators with the best score, for both built-in functions

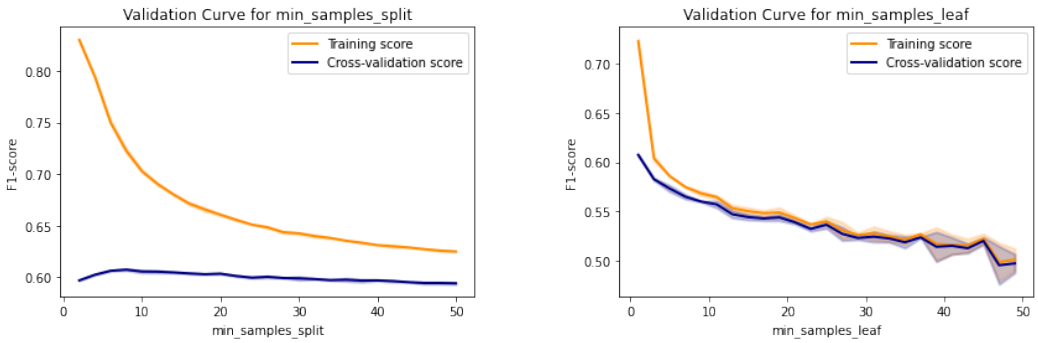
The confusion matrices are very similar which means that both classifiers have a similar performance on the validation set. Thus, the tie is decided by the overall weighted F1-score which was higher for `LogisticRegression`. In comparison to Naive Bayes, although they still have a lot of misclassifications with the most frequent class, the diagonal stands out much more with these two generalized linear models - as would be expected given the higher scores.

Random Forest

The built-in `RandomForestClassifier` fits a number of decision trees on either the whole dataset or on various sub-samples of the dataset, known as bootstrap samples, which are drawn with replacement. This feature is defined by the Boolean parameter `bootstrap`. Furthermore, during the construction of a tree, the function to measure the quality of a split (`criterion`) can be either the Gini impurity or the entropy, for calculating the information gain of each feature. Also, the best node split is found either from all input features or a random subset of size `max_features`. The purpose of these two sources of randomness is to decrease the variance of the forest estimator and avoid overfitting, sometimes at the cost of a slight increase in bias. This randomness in RF generates decision trees with somewhat decoupled prediction errors and, by taking an average of those predictions, some errors can cancel out.

First we perform a grid search on the parameters `bootstrap`, `max_features` and `criterion` as well as on the number of decision trees that make up the ensemble, `n_estimators`. The best model uses bootstrap samples and fits 75 decision trees with the Gini criteria and the default `max_features` - meaning that the number of features to consider when looking for the best split is the square root of the total number of features. The overall weighted F1-score was approximately $59,7\%$.

Next we plot validation curves for the parameters `min_samples_leaf` and `min_samples_split`, which represent the minimum number of instances required to be at a leaf node and the minimum number of instances required to split an internal node, respectively. They help control the size of the trees and their default values lead to fully grown and unpruned trees. In the following plots we can compare both the training score and the validation score and choose the parameter value which lead to the best validation score:



(a) `min_samples_split` **(b)** `min_samples_leaf`

Figure 4.5: Validation curves for both parameters with `RandomForestClassifier`

In Figure 4.5(a), we can see that the validation score remains almost the same for different values of `min_samples_split` and that the training score decreases for higher values. Nevertheless, the parameter

value with the best validation weighted F1-score is 8. In Figure 4.5(b), we can see that both lines tend to decrease from the beginning to the end of the plot and even show some instability for higher numbers. The best value for `min_samples_leaf` is therefore the smallest value, which is 1. Furthermore, we can see that in both plots the training score is much higher than the validation score for smaller values of the parameters, which is a sign of overfitting. This is due to the fact that these parameters control the size of each decision tree, and smaller values mean a deeper tree. Decision trees tend to overfit the deeper they are because at each level of the tree the splits are dealing with a smaller subset of the data there.

The model with the best parameters yielded a weighted F1-score of approximately $60,7\%$. In order to better visualize the performance of the resulting model, we present its confusion matrix:

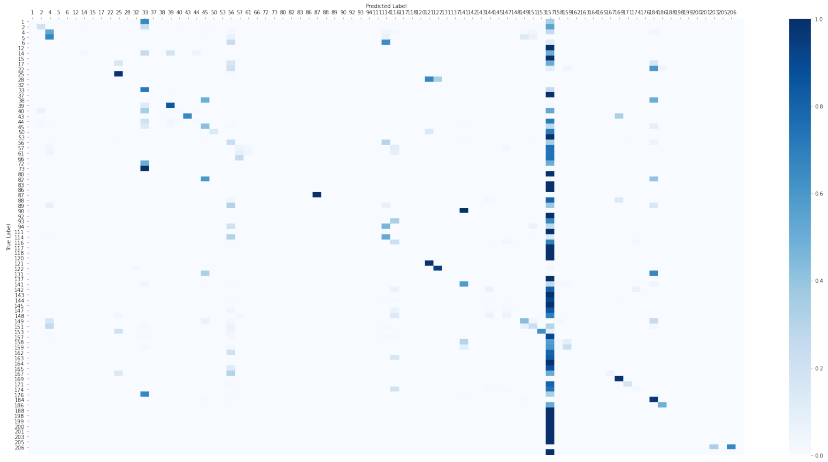


Figure 4.6: Confusion matrix of the Random Forest that yielded the best score

The confusion matrix is similar to the ones of both Softmax Regression classifiers. The performance on the validation set, and in particular the overall weighted F1-score, are very close to the `LogisticRegression` model. We can see that the diagonal is highlighted roughly in the same areas and labels. This pattern may be due to the fact that many of the zero entries on the diagonal belong to labels which are being classified with the majority classes. So it is most likely a consequence of the imbalanced class distribution where some labels dominate over others and it is persistent through all the algorithms treated so far.

Neural Networks

The scikit-learn function that implements a MLP is the `MLPClassifier`, which optimizes the cross-entropy loss function using L2-penalty. Note that it is different from logistic regression, in that between the input and the output layer there can be one or more non-linear hidden layers. In addition, for the case of multi-class classification, the model applies softmax as the output function. For the grid search, we will vary the number of neurons in the hidden layers as well as the regularization parameter `alpha`.

The weight optimization solvers we will choose from are LBFGS, since it was our best solver for softmax regression, and Adam which is the default solver of `MLPClassifier`. Adam is a gradient-based optimization algorithm different from classical SGD. According to [Kingma and Ba, 2014], while SGD maintains a single learning rate[4] for all weight updates and it does not change during training, Adam computes individual adaptive learning rates for different parameters. It uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the NN. We consider the gradient of the cost function to be a random variable[5] since it is evaluated on some small random batch of data. In order to estimate these moments, Adam uses exponential moving averages which apply more weight to recent estimates. The decay rates for the two moments are also controlled by two momentum[6] parameters. In order to provide some differentiation from the classifiers above, the activation functions for the hidden layer that we will choose from are $tanh$ and the rectifier, $reLU$. Finally, for the Adam solver we will consider the parameter `early_stopping` that makes the model terminate training when the validation score no longer improves by setting aside a sub-sample as a validation set, where the split is stratified - the sets are split in a way that they contain the same distribution of classes, or as close as possible.

First, in trying more than one hidden layer, the model started to overfit the training set and so we opted for using just one hidden layer. Second, the LBFGS solver does not converge with MLP which means that no local minimum of the cost function is found in a reasonable number of iterations. While Adam solver converges in less than 400 iterations, the LBFGS solver doesn't converge for less than 1000 iterations. Generally, when a dataset does not present an organized and discernible pattern, machine learning algorithms might not be able to find a convergence point, which is a localized optimal state. In addition, according to [Anand et al., 1993], the backpropagation algorithm converges very slowly for classification problems in which most of the observations belong to one dominant class. This leaves us with the Adam weight optimization solver for determining the optimal weights for each neuron. One way to mitigate this problem is to use early stopping because the model fitting will stop when the validation score does not improve albeit a local minimum may not have been found. The best combination of the other parameters returned by the gird search with cross-validation was a MLP with 75 neurons and the rectifier activation function in the hidden layer, using early stopping and Adam optimization solver. For the `alpha` parameter we plotted a validation curve to visualize its influence on the model:

[4]The learning rate ranges between 0 and 1 and it specifies how fast the learning process is performed. A learning rate of 0 would result in no optimization at all while if the value is set too high the weights can oscillate and may be hard to find the optimal values.

[5]If X is that random variable, the first moment is the mean $E[X]$ and the second moment is $E[X^2]$.

[6]The momentum is the fraction of the last weight change that is added to the new weight, in order to smooth out the optimization process.

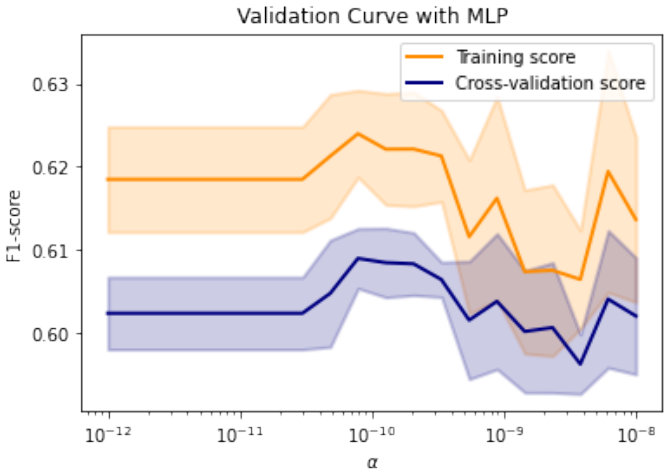


Figure 4.7: Validation curve for parameter tuning with `MLPClassifier`

We observe that there is a lot more variance in this model and that the best L2-penalty regularization parameter is approximately 10^{-10}. The overall weighted F1-score was $60,8\%$. Next, we show the confusion matrix of the estimator implemented with the optimal parameters.

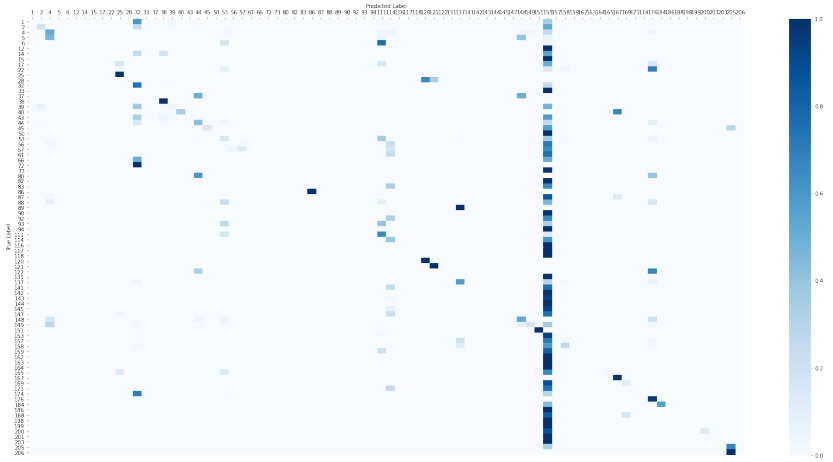


Figure 4.8: Confusion matrix of the MLP that yielded the best validation score

Once again the confusion matrix is very similar to the last two algorithms, and the validation score is also close. The small difference between the performance scores of the methods we have seen (excluding the two Naive Bayes estimators) may be due to one or another minority class being correctly learned or the distinction between a majority class from any other is deepened. In order to learn more about how the labels are being predicted, we use the scilkit-learn function `classification_report` which returns the precision, recall and F1-score of each label. As we have seen earlier in this chapter, recall is a performance metric that for each class indicates how many actual positives are captured by the model while precision indicates how precise the model is out of all predicted positives. Bearing this

in mind, we computed the difference of the values in the classification report of the best `MLPClassifier` and the best `SGDClassifier` for our case. We found that for several minority classes (labels with a small support in the validation set) the MLP estimator has higher recall scores which means that the model is able to correctly predict more instances from all the actual instances of those classes.

An alternative to the scikit-learn library when it comes to neural networks is to use Keras, which is a deep learning Application Programming Interface (API) written in Python running on top of the machine learning platform TensorFlow. It is user-friendly and it is widely used for its deep-learning powered features. Neural layers, cost functions, optimization solvers, activation functions and regularization schemes are all standalone modules that one can combine to create new models. Since there are an infinite number of combinations of neural network features that can make up our model, it is almost impossible to get the correct parameter values, number of neurons in each layer and even the correct number of layers of each type, as it requires years of experience and expertise and as well as extensive trial and error to find the optimal values for our model. According to [Heaton, 2008], a NN with one hidden layer can approximate any function that contains a continuous mapping from one finite space to another whereas a NN with two hidden layers can approximate any smooth mapping to any accuracy. [Heaton, 2008] also stresses the fact that even though generally 2 hidden layers will enable the network to model any arbitrary function, sometimes multiple hidden layers can be fruitful for large and difficult problems. In addition, the author mentions some rules-of-thumb as a starting point for determining an acceptable number of neurons to use in the hidden layers, e.g. the number of hidden neurons should be between the size of the input layer and the size of the output layer. Ultimately, the selection of an architecture for the neural network will come down to trial and error.

Since Keras does not handle `GridSearchCV` properly, we will perform parameter tuning with cross-validation manually. Unfortunately, it is difficult to obtain reproducible results with Keras. Therefore, in order to smooth the effects of randomness and to get more reliable performance scores, we increased the number of folds to 10. Furthermore, we ranged over the number of neurons in the hidden layers, the batch size and the number of epochs. The batch size is the number of instances that are propagated through the network before updating the weights. Having a batch smaller than the whole training set requires less memory and the network typically trains faster because it is updating the weights more often. However, the smaller the batch the less accurate the estimate of the gradient will be. The number of epochs corresponds to the the number of times that the entire training set will be processed. As the number of epochs increases, more number of times the weight are changed in the neural network and we can go from underfitting to optimal values to overfitting.

The types of layers we will consider in order to keep the model simple is the Dense layer, the LeakyReLU layer and the Dropout layer. The Dense layer is a regular fully-connected layer, where

for each neuron we take the dot product between the input vector $\mathbf{x}$ and the weight kernel matrix W featured in the Dense layer, add a bias vector (if we want to include a bias), and finally take an element-wise activation of the output values.

$$f(\mathbf{x}) = \text{activation}(\mathbf{x} \cdot W + \text{bias})$$

This type of layer is usually implemented as the first hidden layer. The activation function to be used will be the rectifier, ReLU, as it was the best choice for the MLP. The only parameter to be tuned will be the number of neurons of the layer.

The LeakyReLU layer uses a leaky version of the ReLU activation function. The traditional ReLU is equal to $f(x) = \max(x, 0)$ but this leads to *dead* neurons that output always the same value 0, specially when the optimizer learns large negative weights or bias. Once a ReLU neuron ends up in this state, it is unlikely to recover because the gradient at 0 is also 0 and so gradient descent learning will not alter the weights. The paper that introduces the leaky version [Maas et al., 2013] argues that the death of neurons can be avoided by allowing a small gradient when the unit is not active:

$$f(z) = \begin{cases} \alpha z & z < 0 \\ z & z \geq 0 \end{cases}$$

Finally, the Dropout layer helps prevent overfitting. It randomly sets input neurons to 0 with a frequency of the parameter $\texttt{rate} \in [0, 1]$ at each step during training time. Inputs not set to 0 are scaled up by $1/(1 - \texttt{rate})$ such that the sum over all inputs is not changed. Although in Keras it is called a layer, it is more of a regularization which makes the NN become less sensitive to the specific weights of neurons. This, in turn, results in a network that is capable of better generalization.

Now that we have an overview of the layers we will consider for this model, we look for the best combination of these layers and the best parameters α and $\texttt{rate}$. In order to compare the different models, we created a function that computed the weighted F1-score during the training phase since Keras does not handle the scikit-learn metric nor it has a built-in weighted F1-score. Furthermore, for training, the batch size and number of epochs that we found to be the more suitable were 1000 and 150, respectively. Moreover, when compiling the NN we will use Adam as the weight optimizer because according to [Kingma and Ba, 2014] the method is *"computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters"*. In Keras, the default learning rate for Adam is $0,001$ and the default momentum parameters for the first and second moment are $0,9$ and $0,999$, respectively. Furthermore, we will use the categorical cross-entropy as the loss function which corresponds to the softmax regression loss: a softmax activation plus a cross-entropy loss. Figure 4.9 shows the NN that yielded the best weighted F1-score, after performing a 10-fold cross validation on various combinations of layers. The input layer receives the 565 covariates and the first Dense layer is responsible for transforming those neurons us-

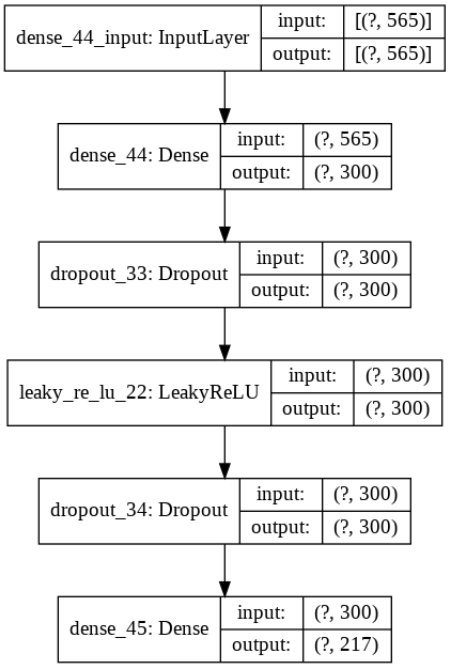


Figure 4.9: Scheme of the Keras Neural Network

ing the rectifier activation function as mentioned and giving 300 neurons as output. The input layer is followed by a Dropout layer with probability $0,2$ of setting the input neurons to 0. Then, it comes a LeakyReLU layer with $\alpha = 0,3$ which is the default value. Next, we have another Dropout layer that sets its input to 0 with probability $0,3$. Lastly, the output layer is a Dense layer with the softmax activation function and outputs 217 neurons, which corresponds to the total number of classes in our problem. The question mark in the figure corresponds to the yet unknown batch size. The overall validation weighted F1-score was $62,72\%$. Figure 4.10 shows the confusion matrix of the performance of the described NN on the test set.

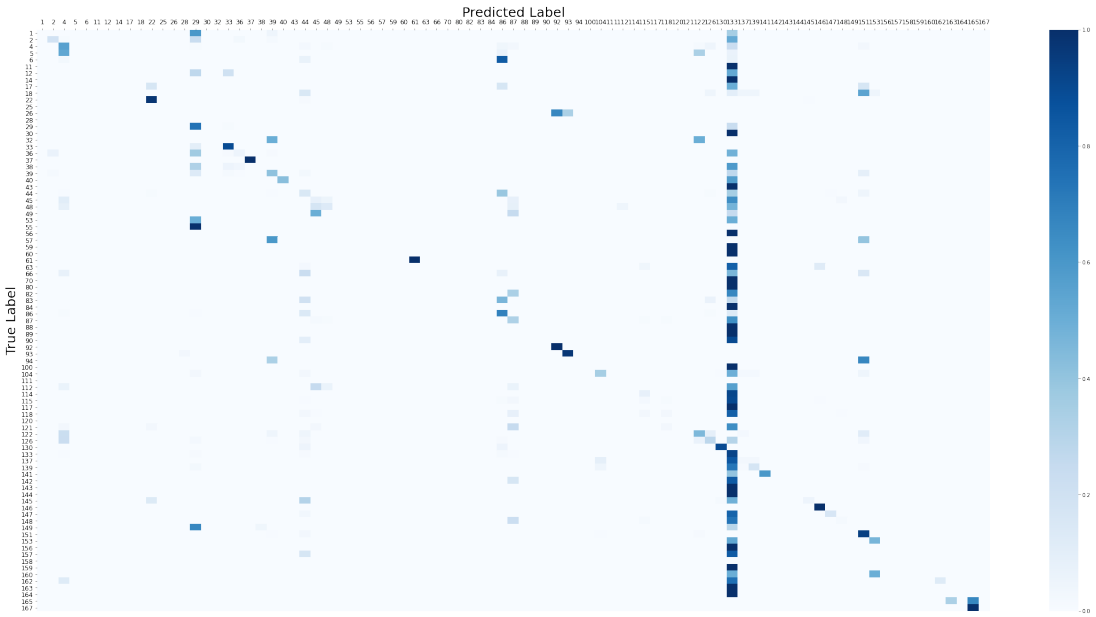


Figure 4.10: Confusion matrix of the NN with the best generalization score

The confusion matrix is very similar to the one of the MLP, despite having a slightly better validation score. The more accentuated columns are overall the same throughout the models and they correspond to labels that the models predict more. In other words, those persistent ”vertical lines” are a sign of the dominance of the more frequent labels for which the algorithm confuses labels it does not learn so well. We conclude that the performance of the ML algorithms on the raw classification of the final dataset is limited to a certain point due to the properties of the dataset itself. The class imbalance and the poor profiling of the causes present challenges to the classification.

4.2.1 Model Selection

After evaluating a suite of ML algorithms on the technical issues dataset, we bring this approach to a conclusion by putting the models side to side and selecting the one with better performance. The following plot shows each sample of weighted F1-scores after cross-validation as a box and whisker plot with the same scale so that we can directly compare the distributions.

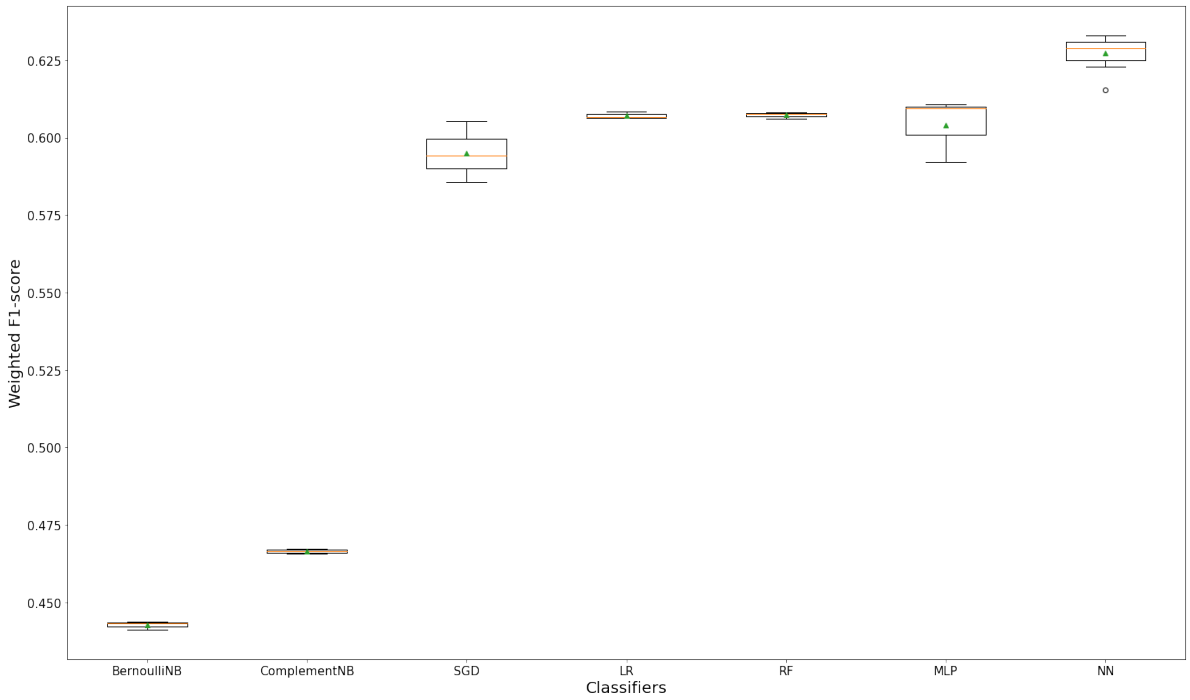


Figure 4.11: Box plot of the results from model evaluation

The figure shows one box and whisker plot for each algorithm's sample of results. The box shows the middle 50 percent of the data, the orange line in the middle of each box shows the median of the sample, and the green triangle in each box shows the mean of the sample. As already seen, the Naive Bayes methods yield poor results. The SR classifiers, the Random Forest and the MLP give similar scores, the lowest belonging to the SR with SGD training. In addition, we observe that both LR with OVA approach and Random Forest have very small variance of their weighted F1-scores. In most cases, the mean and median are close on the plot, suggesting a somewhat symmetrical distribution of scores that may indicate the models are stable. The best model was indeed the Keras Neural Network but unfortunately even the best result is far from what we would hope for.

In Section 3.4, we mentioned the fact that Border Innovation was interested in another type of accuracy that shows a broader view of the model performance. Instead of computing a performance metric that evaluates if the most probable cause is the true cause, we will evaluate whether the true cause is in the 3 most probable labels, and we will call it Top 3 accuracy. To this end, we created a function `predict_NN` that given a new observation returns the 3 most probable classes and the respective probabilities if specified. Then, we created another function `evaluate_NN` that computes the Top 3 accuracy given the model and the test set. The code is available at GitHub in the file `Classifier.ipynb`. The Top 3 accuracy for the Keras Neural Network was $90,77\%$. We also computed the Top 3 accuracy for the other discriminative models studied in this chapter and found that the scores fluctuated around 90%. This shows that the models tend to opt first for the majority causes and only then predict the minority

ones. If the true cause is within the Top 3 of causes predicted by the model with more than 90% confidence and if the technical assistant has access to that set of causes, there is a higher chance of solving the problem than with just a single output of the model. The improvement of results suggests that the final recommendation system should have multiple possible choices for the cause of a customer support issue. Once the model outputs the Top 3, the assistant can go through each item or even deliberate on the problem - since the personal contact with the user always gives more details that a machine may not comprehend -, and decide what seems to be the best choice.

4.3 Resampling

As we have seen, only 53% of the possible causes are represented in the dataset, the majority of which has very few observations. There is no ”cure” for imbalanced classification. It remains an open problem, and practically must be identified and addressed specifically for each training dataset. Nevertheless, one approach to handle class imbalance is to balance classes in the training set (data preprocessing) before providing the data as input to the machine learning algorithm. The main objective of the so called resampling techniques is to either increase the frequency of the minority class or decrease the frequency of the majority class, in order to obtain approximately the same number of instances for all the classes. There are two main strategies:

- **Over-sampling** randomly replicates instances in the minority classes so that those classes have a higher representation in the dataset. Over-sampling may increase the likelihood of overfitting since it replicates the minority class events, according to [Fernández et al., 2018].

- **Under-sampling** randomly removes instances in the majority classes so that their dominance is weakened. Under-sampling can discard potentially useful information which could be important for building rule classifiers, according to [He and Ma, 2013].

Since there are several minority classes, we do not want to replicate within a single minority class. Therefore, for the over-sampling method, we created a dictionary with the classes as keys and the class frequency in the dataset as the corresponding value to serve as sampling strategy. After trying different values, we determined that each label with less than 100 observations in the training set would have an extra 100 observations. For the under-sampling method, we resampled only the majority class 153 to 10000 observations which is almost half of its absolute frequency in the training set. Finally, we attempted a combination of both sampling strategies.

The code is available in the end of the file `Model.ipynb`, in the GitHub account where all the code related to modeling is located. We will use the implementations provided by the imbalanced-learn Python library and we will combine these techniques with one of the ML models that we obtained in the pre-

vious section. We tried using the `LogisticRegression` classifier and the `MLPClassifier` with the best combination of hyperparameters as seen in the previous section. They are simple estimators and they gave some of the best results. Since the results were similar, we chose to present the outcome of the `LogisticRegression` in the following table:

	Train Accuracy	Train F1-score	Validation Accuracy	Validation F1-score
No sampling	0.6667	0.6167	0.6691	0.6191
Over-sampling	0.6662	0.6140	0.6641	0.6156
Under-sampling	0.5801	0.5454	0.6344	0.6184
Combination	0.6017	0.5684	0.6200	0.6120

Table 4.1: Resampling results

All these approaches yield similar results to the no sampling strategy. Unfortunately, resampling does not improve the overall performance of the model as none had a better score. Also, combining over and under-sampling does not achieve better results. There are other strategies but they mainly focus on distance-based approaches. In other words, new synthetic similar instances are created by generating data points that are close to the minority classes. However, in our case, we deal with binary data and similarity measures between binary vectors do not necessarily make sense in this problem: what is a technical issue close to another?

5

Multi-step Approach

Contents

This chapter addresses another attempt to mitigate the consequences of class imbalance. This approach tries to combine the best of the models we have seen in the previous chapter into a single model. Since the classes in our problem have very different frequencies, we will try to divide them into groups of classes with similar presence in the dataset. The model will be a four-step classification where in each step we will fit a sub-model on different groups of labels. Also in each step, after we split the data into training set and test set, we preform a 3-fold cross validation on every algorithm we try so that we have a better perception of their performance and we also present the variance of the scores. The first step consists on a binary classification, separating the most frequent class 153 and grouping all the other classes into one class. Then, from that group of other classes, we take a number of classes that make up a certain cumulative frequency of the remaining data points and group all the other classes into one class, and so on until we classify the remaining classes in the final step. The code for this chapter is available at GitHub in the file `MultiStepModel.ipynb`.

5.1 Step I

In the first step, we will deal with the biggest disparity in the class distribution. The class 153, which corresponds to the issue having no determined cause, has 31944 observations while the second most frequent class has only 7258 observations. Therefore, we created a new dataset where all the labels different from 153 are grouped together into a single class, giving a binary classification problem. The following barplot shows the distribution of the classes:

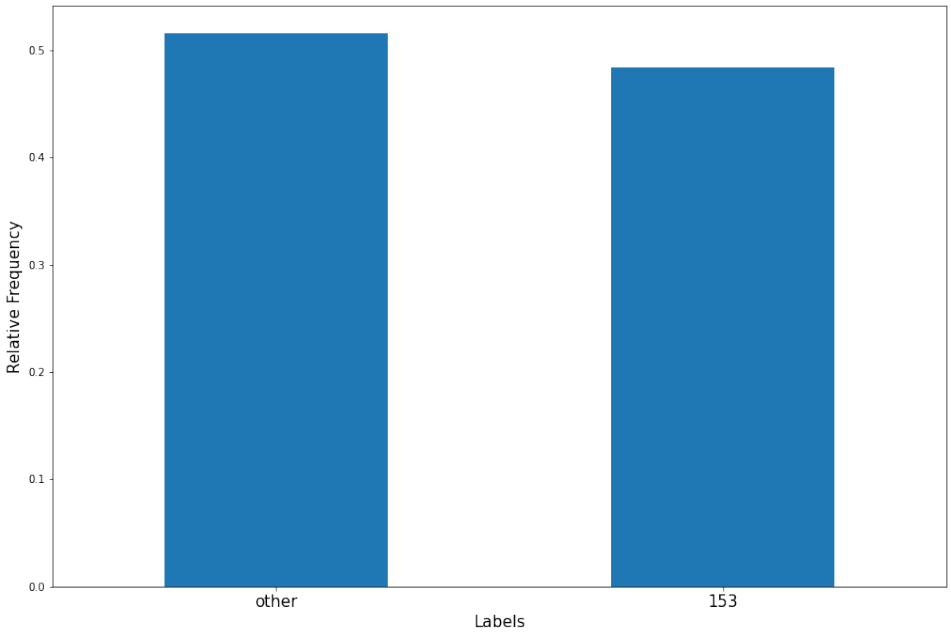


Figure 5.1: Class distribution for Step I

We tried several algorithms based on those we have seen previously in this work from the scikit-learn library but now considering we have a binary classification to predict the label 153 and the other label comprising all the determined causes. Upon performing 3-fold cross validations, the function with better performance was the `MLPClassifier`, with 30 neurons in the hidden layer and the default regularization parameter. The overall weighted F1-score was $80,24\% \pm 0,24\%$. Figure 5.2 shows the confusion matrix after using the chosen MLP classifier to predict the test set.

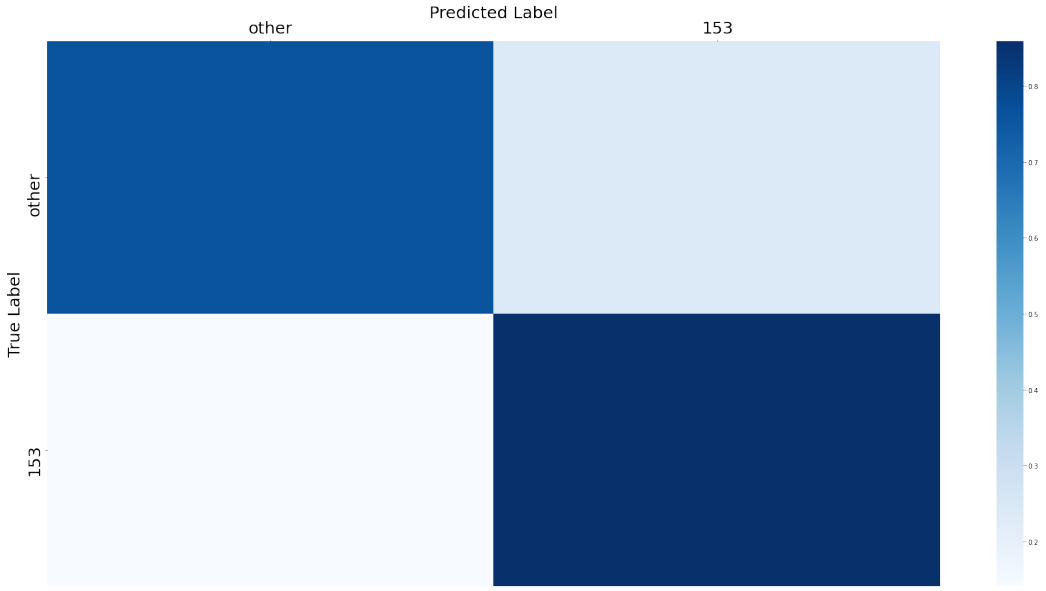


Figure 5.2: Confusion Matrix for Step I

We observe that the label 153 is being correctly predicted but some observations with other causes are being classified as having no determined cause.

5.2 Step II

In the second step, putting aside all the observations with label 153, we take the group of all determined causes from step I and select the labels which together make up half of the dataset, sorting their relative frequency from highest to lowest. In other words, we create a new dataset without any observation of label 153 and from the new set of classes we keep those that their cumulative relative frequency reaches 50% and group the remaining classes into a single class. The following barplot shows the distribution of the labels:

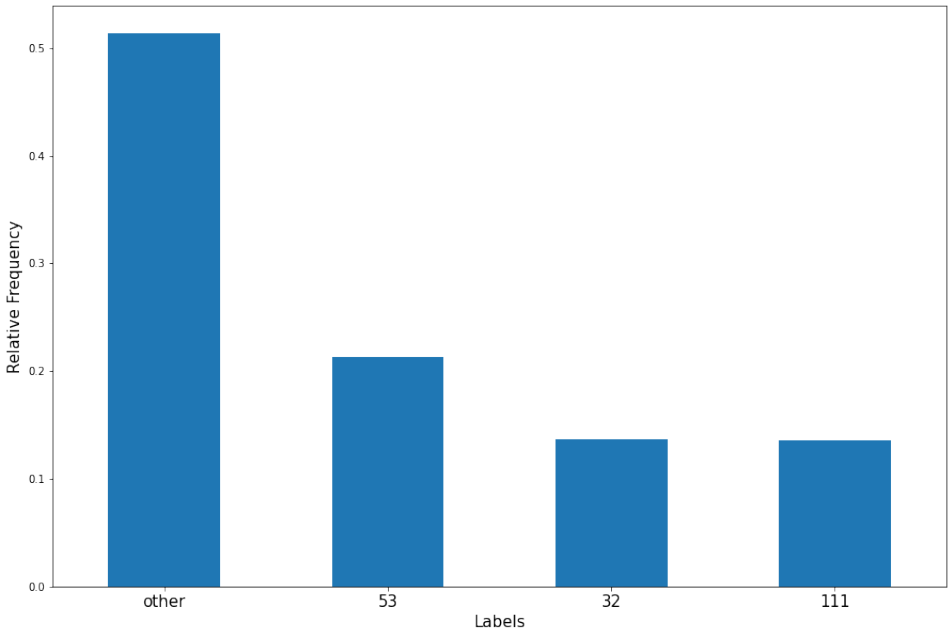


Figure 5.3: Class distribution for Step II

Only three labels are needed to hold 50% of the remaining observations upon removing the most frequent label. The best algorithm for this classification problem was also the MLP, this time with 50 neurons in the hidden layer and the regularization parameter equal to 10^{-10}. The overall weighted F1-score was $71,12\% \pm 0,39\%$. Figure 5.2 shows the confusion matrix when predicting the test set.

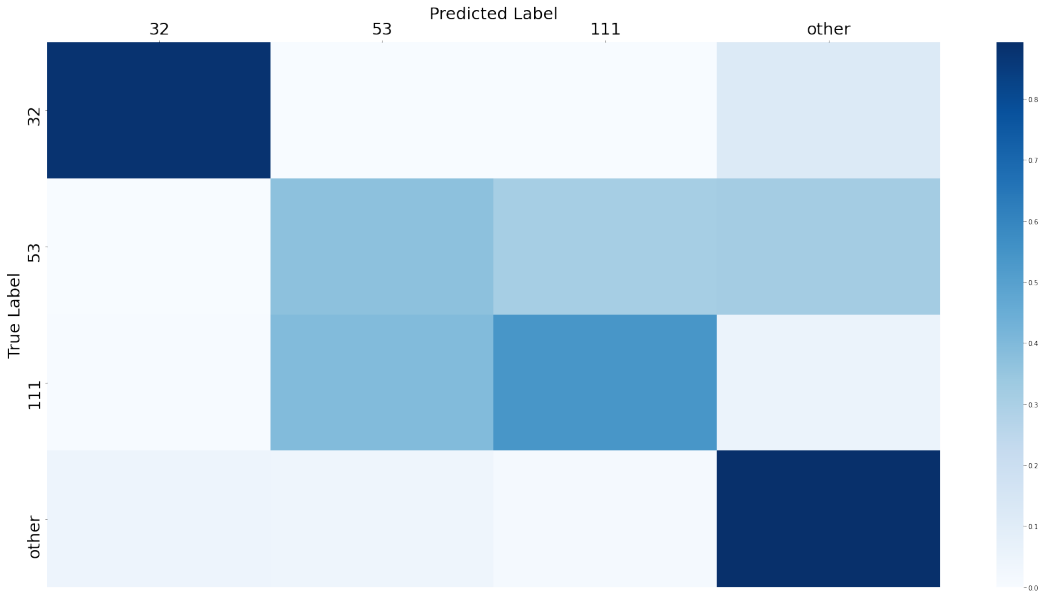


Figure 5.4: Confusion Matrix for Step II

We observe that the class 32 and the class *other*, which represents the group of the remaining

classes, are relatively well classified. However, the class 53 is often misclassified where more than one different label is being predicted. This suggests that the predictive profile of the class 53 does not have a distinctive pattern. In addition, the class 111 is somewhat mixed up with class 53. Following this, we checked what these labels represent: the cause 111 corresponds to a non-registered Optical Network Terminal (ONT), which is related to the Internet, and the cause 53 corresponds to difficulty in connecting. We were informed by Border Innovation that this is indeed a case of human error as the distinction between theses two causes is not very clear and the technical assistant may confuse them.

5.3 Step III

In the third step, we look into the new *other* class and adopt a similar approach as before. Taking into account the cumulative relative frequency of the new dataset without any of the labels represented in the previous steps, the minimal set of classes that hold 70% of this dataset is kept and all the others are grouped into a single class. The following barplot shows the distribution of the classes:

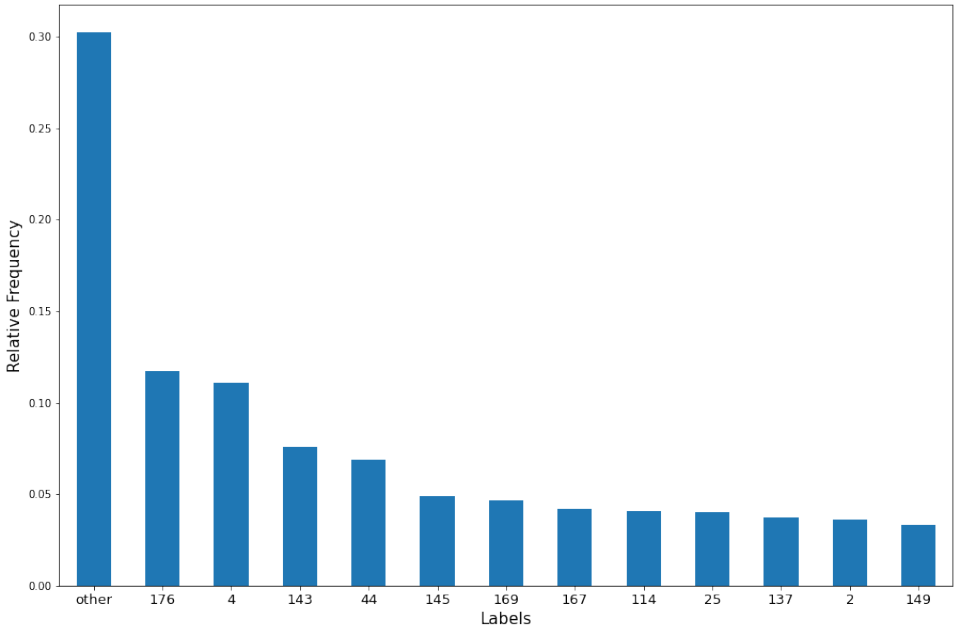


Figure 5.5: Class distribution for Step III

We can see that the frequencies of the remaining classes are becoming smaller and smaller, and so the number of classes that are kept is higher because more classes are needed to represent a significant portion of the dataset - in other words, to meet our requirement regarding the cumulative relative frequency. The algorithm that had the best weighted F1-score in average was in this case the `LogisticRegression`, with the same parameters as the best one in the previous chapter and with

a weighted F1-score of $66,88\% \pm 0,33\%$. There is a downwards trend of the cross validation scores which can be due to the fact that the more steps we take the more labels we have in our classification sub-problem. However, we must also take into account that as we get closer to the minority classes, the predictive power of the model becomes more fragile. Figure 5.2 shows the confusion matrix upon predicting the test set.

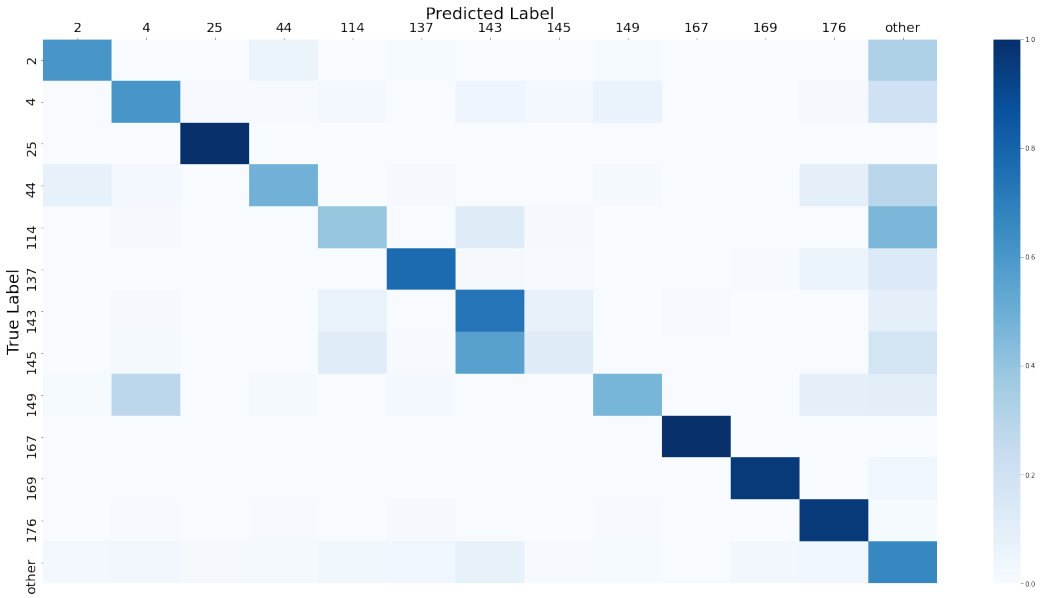


Figure 5.6: Confusion Matrix for Step III

Many of the represented labels are misclassified as being in the *other* class. This suggests that there is a lot of confusion in this part of the dataset where higher frequency labels are being mixed up with some of the lower frequency labels, which points to indistinct outlines between causes.

5.4 Step IV

The last step consists of classifying all the remaining labels. Upon creating a new dataset with only the labels that were not represented in the previous steps, several scikit-learn functions were fitted on that dataset. However, not only they showed poor results when compared to the performance on the whole dataset in Chapter 4 but also they tend to overfit the training set. That is why we chose to use a Keras NN for this last step of the model, as it was the technique with best results in the previous chapter. First, the following barplot shows the distribution of the remaining classes:

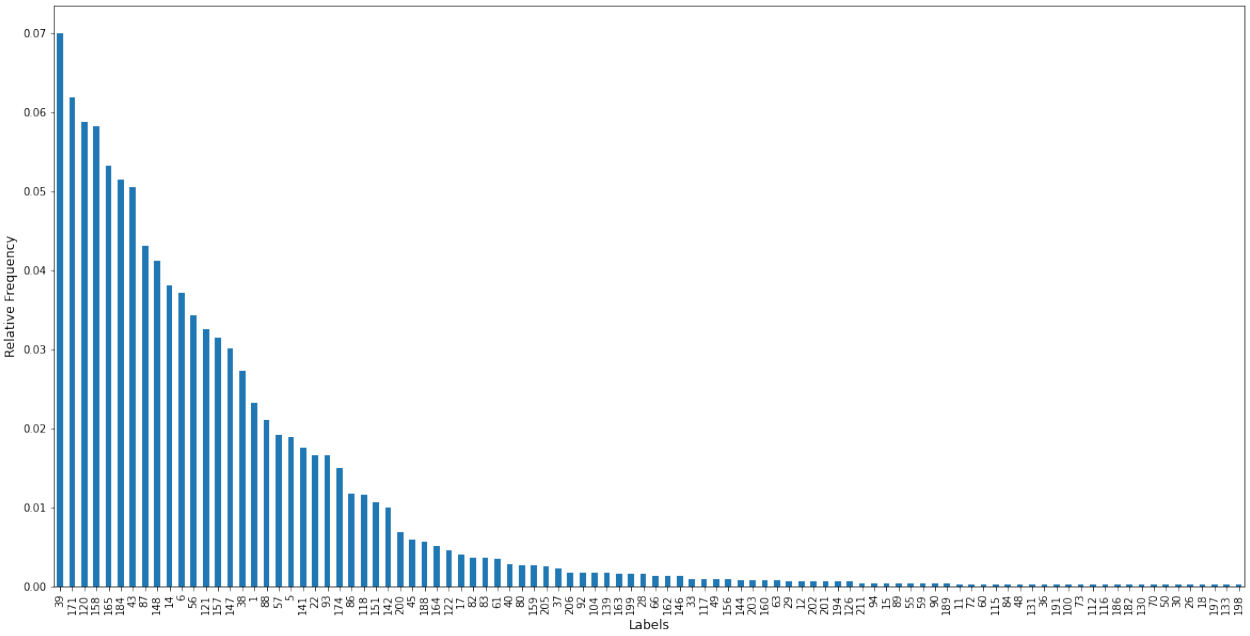


Figure 5.7: Class distribution for Step IV

There are still 98 labels left. This means that from the 114 causes that are represented in the dataset, only 16 were handled in the previous steps of the model, as these hold significantly more observations than the 98 minority causes. We considered dividing the model into further steps but the algorithms start to have a severe problem of overfitting. This is because many of the labels in the barplot, 24 to be specific, have just one instance. Therefore, when splitting the dataset into training and test set some of these labels will appear only on the test set without the model being trained with them and so the labels will not be correctly classified. Furthermore, the class distribution is very imbalanced. Even without the more dominant causes, there is still a great disparity between the ones with lower relative frequency which also explains the poor performance of the algorithms tried in this last step. Nevertheless, the Neural Network with the best weighted F1-score after performing 10-fold cross validation on the training set is similar to the one chosen by the brute-force approach. It has three hidden layers: the first is a Dense layer with 300 neurons; the second is a LeakyReLU layer with $\alpha = 0,3$; the third is a Dropout layer with `rate` equal to $0,5$. The overall weighted F1-score was $62\% \pm 1,5\%$. Figure 5.8 shows the confusion matrix upon predicting the test set.

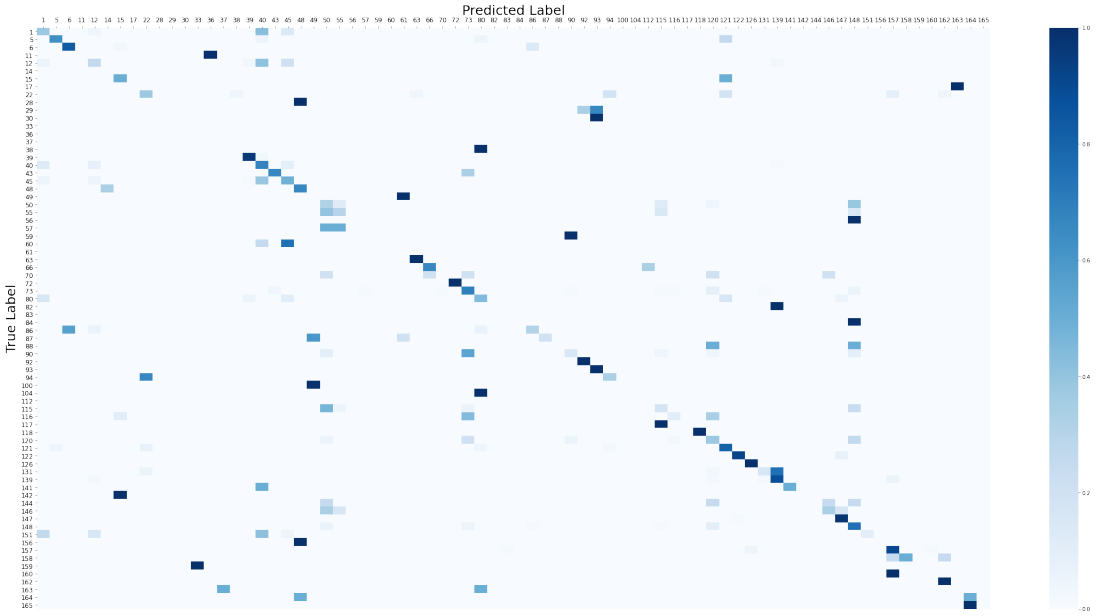


Figure 5.8: Confusion Matrix for Step IV

Despite the fact that the diagonal stands out for the most part, there is a lot of chaos in the prediction where several labels are being mixed up. This helps to explain why the performance measures are declining when we reach the minority classes. Moreover, the least populated classes have only 1 member and therefore, when splitting the dataset for cross validation, they will be misclassified if not in the training set.

5.5 Model Evaluation

In this section, we combined all the steps and constructed a single model through the function `models`. The code is available at GitHub in the file `Classifier.ipynb`. In order to evaluate its performance and since we are dealing with a new and complex model, we created a function, `predict_steps`, that given a new observation returns the 3 most probable causes (and the probabilities if specified). This prediction method passes the observation through each sub-model and propagates the probabilities. For a new observation, the probability of it belonging to class 153 is given directly by the first sub-model. Then, the probability of belonging to a class of step II is the probability given directly by the second sub-model multiplied by the probability that in step I it belonged to the *other* class. If x is the new instance, $P(other_i|x)$ represents the probability of it belonging to the class *other* of step i and $C_k^{(i)}$ represents the class C_k where $i \in \{1, 2, 3, 4\}$ indicates the number of the step that class is represented, we have:

$$P(C_k^{(i)}|x) = \begin{cases} P(153|x) & \text{if} \quad i = 1 \\ P(C_k|x) \prod_{j=1}^{i-1} P(other_j|x) & \text{otherwise} \end{cases}$$

Moreover, we created two performance metric functions, one for accuracy, `acc`, and another one for the weighted F1-score, `f1_score`. After splitting the whole dataset and fitting the assembled model on the training set, we computed those metrics for the test set: the accuracy was $64,7\%$ and the weighted F1-score was $59,3\%$. Unfortunately, the performance of this multi-step approach was not better than a single ML algorithm.

Regarding the Top 3 accuracy, it yielded $86,1\%$ which is still lower than the score of Keras Neural Network. Even though these results are not better than the previous approach, this chapter gave a better insight on the imbalance of the dataset and how the causes are being predicted. It become clear that there are some labels that are confused with each other by the technical assistant and that a lot of them are mixed up by the model itself due to poor predictive power. Finally, there is a lot of confusion with the minority labels, specially the ones with the lowest frequency which represent more than a fifth of all the present labels.

6

Conclusion

Contents

In this chapter we present the conclusion of this work and highlight the major findings. Many of the points made here are also discussed in Chapter 4 and Chapter 5. We end this work by highlighting possibly interesting future work.

6.1 Main Results

This work answered the following research question: *How can we help the assistant properly diagnose a technical issue?* This was done by gathering data, preprocessing data, attempting several classification algorithms and choosing the one with the best generalization score. We found that Neural Networks are the best approach for this dataset. However, it is hard to draw any further conclusion given the properties of the dataset itself. The preliminary analysis hinted not only at the problem of high dimensionality, particularly the number of covariates, and sparsity of the data but also at the imbalanced class distribution. Although discriminative models performed fairly well, the dominance of the more frequent classes over others resulted in a drop of the generalization score. Even though some problems are more frequent than others, this imbalance can also be due to the fact that we rely on the labeling provided by the technical assistant which adds human error.

The multi-step approach, despite not giving the best results, helped give a clear view on how the classes are being predicted. We saw that alongside the imbalanced distribution problem, several minority labels are being mixed up. This was another major finding: the poor profiling of the causes. Great part of the causes, not only have they less than 10 instances in the whole dataset, but they also have profiles that are mixed up by the models in general. In particular, the ones with only one observation in the dataset are always misclassified in the validation set. In the preprocessing analysis we found that some causes showed no pattern of distinction from the others and that the correlation between the test group and those causes was poor.

Nevertheless, the goal of this thesis was to contribute to a recommendation system for telecommunication technical issues. Given the properties of the dataset and after experiencing with the more appropriate machine learning methods, we reached the result that Neural Networks are able to better learn the relationships in the data and generalize to new observations. The proposed model consists of a neural network with 2 hidden layers: the first Dense layer and the LeakyReLU layer. The Dropout layers are considered regularization implemented on other layers. Although the maximum results did not exceed 70% in accuracy and 65% in weighted F1-score, we reached scores higher than 90% when computing the Top 3 accuracy - which indicates whether the correct cause is in the 3 most probable labels that the model predicted.

Based on these conclusions, it is our recommendation that a verification on the root cause of the training examples be made in order to mitigate the human error. Furthermore, we recommend a new

organization of the causes in order to make them more separable. For example, group minority causes with the same resolution and remove vague and ambiguous causes. This simplification and the decrease in the number of classes may help to make the problem less complex and consequently contribute for a better performance of the algorithms. Finally, in terms of the training examples having poor predictive power, we recommend that an analysis should be done on the insertion and update of the users technical issues in the database (from which the dataset was retrieved) due to problems as the one of overlapping test results referred in 3.2.2. Also, the fact that some observations labeled with a certain cause and the covariates related to that cause are not indicative of any error reveals that the values on the predictive variables may not be in sync with the respective causes. Nonetheless, some of the discriminative models of the brute force approach show potential for being good classifiers once the impact of these issues is reduced.

6.2 Future Work

This work presents a classification system for customer support technical issues using machine learning methods. Extending the work on this topic, an attempt to further improve classification quality could be made by giving as input to the model more information about the problem that quantitative measures cannot express. During the conversation between the user and the technical assistant, there are many details that the binary data does not convey. For example, if the user is experiencing symptoms that are not accounted for or if there is something in what the user says that absolutely points to a specific cause, our dataset alone is not able to show it to the model. It could learn far better how to distinguish each cause if these aspects are written by the assistant in the form of text and used as input by the model. This approach would result in more informative and complete profiles for each cause which the model could learn from, since the algorithm's certainty for the sentence-based classifications is taken into account. Text mining is an artificial intelligence methodology that uses natural language processing to transform unstructured text in documents and databases into normalized, structured data suitable for analysis or to drive machine learning algorithms. Through the process of examining texts to discover new information and help find a solution to our research question, we might achieve better generalization scores.

The idea is to produce a multiple input model which results from concatenating our model proposed in Subsection 4.2.1 with the predictive model proposed in [Oliva, 2020]. The thesis submitted by Mariana Oliva was developed within the same context of this work but for text classification. The dataset used for the training phase consists of customer call descriptions in text form labeled with the respective cause and the final model in her thesis is a Recurrent Neural Network that given a call description, outputs the most probable cause. Note that before giving the description text as input to the NN, it has to

be preprocessed by performing text correction, word lemmatization, removal of *stop words* and word indexing.

Since the two models in question were developed in the same framework, Border Innovation provided a new dataset where each row had the call description and the binary features together, labeled with the respective cause. In a multiple input model we can have different types of input data and a final single output, but the question is: *How to combine numerical and text features in neural networks?* In order to combine different feature spaces inside the neural network, in our case a binary vector and a text feature, we need to have two submodels. The constructed model is implemented in Keras API and it goes as follows:

1. After preprocessing the text and the binary data, we define two input layers and treat them in separate models. The first input layer is used to input the sequences of indexes, presented in [Oliva, 2020], and the second input layer is used to input the binary vectors.

2. The output of the first submodel is the result of applying the hidden layers of the text classification model proposed by [Oliva, 2020].

3. The output of the second submodel is the result of applying the hidden layers of the model proposed in this thesis.

4. These two outputs are concatenated using the Concatenate class from the `keras.layers.merge` module.

5. This combined vector is now classified in a Dense layer and finally the softmax activation function is used in the output neurons.

The diagram of the complete model is presented in Figure 6.1.

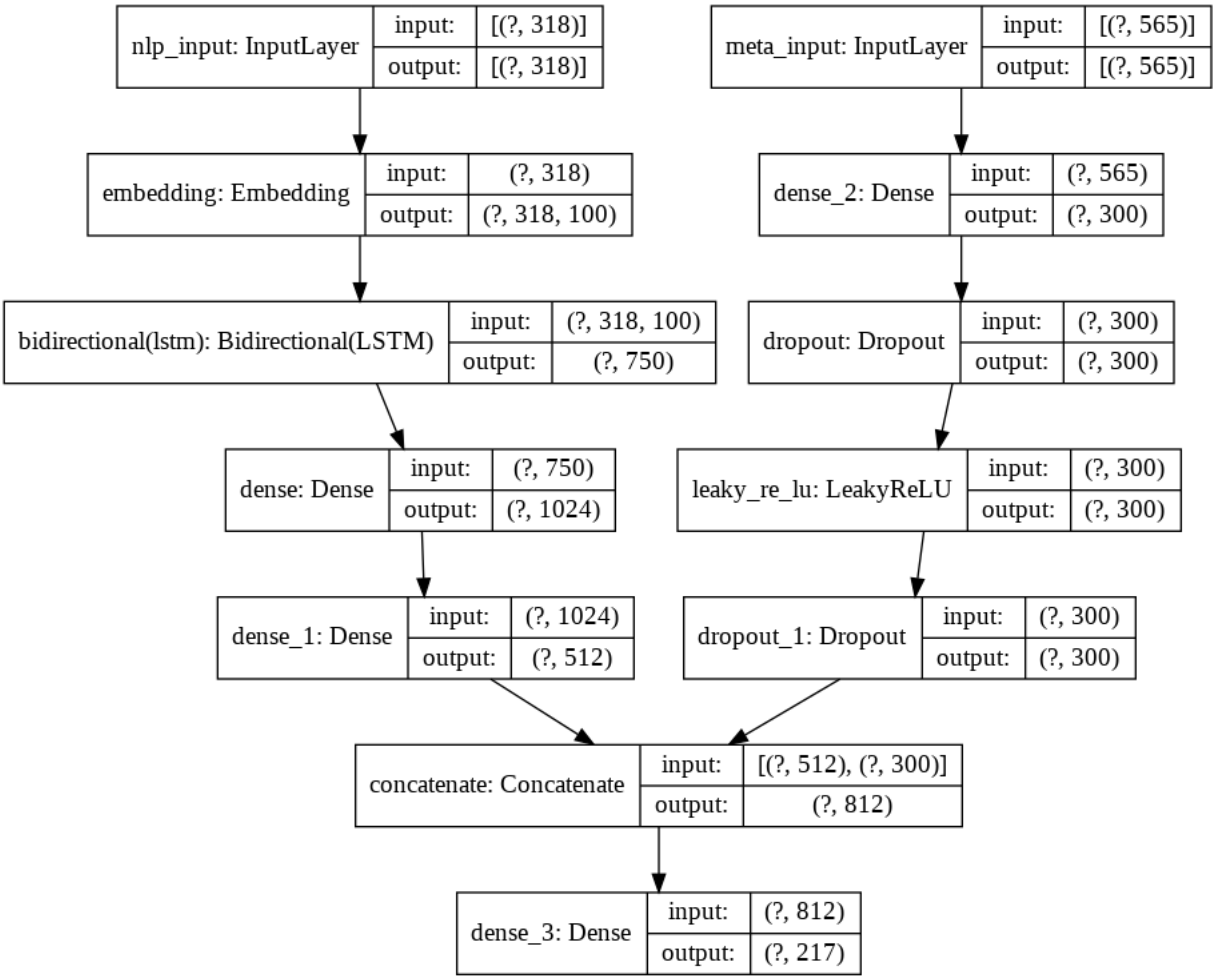


Figure 6.1: Scheme of the Multiple Input Model

The results from this neural network surpassed the single model introduced in this thesis. It achieved an accuracy of 75.5% and a Top 3 accuracy of 94.74%. This shows that a joint model is indeed an improvement from two separate models. Moreover, the addition of relevant information, that comes from the assistant description of the problem, appears to help the algorithm have a more clear distinction between classes. However, it should be noted that the dataset used to build the multiple input model is not the same used for constructing the solution of this thesis.

Bibliography

[Anand et al., 1993] Anand, R., Mehrotra, K. G., Mohan, C. K., and Ranka, S. (1993). An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969.

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer.

[Brownlee, 2014] Brownlee, J. (2014). *Machine Learning Mastery*.

[Caruana and Niculescu-Mizil, 2005] Caruana, R. and Niculescu-Mizil, A. (2005). An empirical comparison of supervised learning algorithms using different performance metrics.

[Chan et al., 2001] Chan, J.-W., Chan, K.-P., and Yeh, A.-O. (2001). Detecting the nature of change in an urban environment: A comparison of machine learning algorithms. *Photogrammetric Engineering and Remote Sensing*, 67:213–225.

[Developers, 2020] Developers, S.-L. (2020). *Scikit-Learn User Guide*, release 0.23.1 edition.

[Dietterich and Kong,] Dietterich, T. and Kong, E. B. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms.

[Erb, 1993] Erb, R. J. (1993). Introduction to backpropagation neural network computation. *Pharmaceutical Research*.

[Feldman et al., 2016] Feldman, D., Volkov, M., and Rus, D. (2016). Dimensionality reduction of massive sparse datasets using coresets. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 2766–2774. Curran Associates, Inc.

[Fernández et al., 2018] Fernández, A., García, S., Galar, M., Prati, R., Krawczyk, B., and Herrera, F. (2018). *Learning from Imbalanced Data Sets*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Gupta, 2017] Gupta, P. (2017). Decision trees in machine learning. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052. Last accessed on Jul 10, 2020.

[Géron, 2017] Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media.

[He and Ma, 2013] He, H. and Ma, Y. (2013). *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, 1st edition.

[Heaton, 2008] Heaton, J. (2008). *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, Inc., 2nd edition.

[Khondoker et al., 2013] Khondoker, M., Dobson, R., Skirrow, C., Simmons, A., and Stahl, D. (2013). A comparison of machine learning methods for classification using simulation with multiple real data examples from mental health studies. *Statistical methods in medical research*, 25.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

[Landgraf and Lee, 2020] Landgraf, A. J. and Lee, Y. (2020). Dimensionality reduction for binary data through the projection of natural parameters. *Journal of Multivariate Analysis*, 180.

[Luckert and Schaefer-Kehnert, 2016] Luckert, M. and Schaefer-Kehnert, M. (2016). Using machine learning methods for evaluating the quality of technical documents.

[Maas et al., 2013] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models.

[Metsis et al., 2006] Metsis, V., Androutsopoulos, I., and Paliouras, G. (2006). Spam filtering with naive bayes - which naive bayes? *In CEAS*.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

[Murphy, 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

[Ng and Jordan, 2001] Ng, A. Y. and Jordan, M. I. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes.

[Oliva, 2020] Oliva, M. (2020). Text analytics to improve telecommunication customers service management from unstructured data. unpublished thesis.

[Pearson, 1901] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[R Core Team, 2020] R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[Roos and Edvardsson, 2008] Roos, I. and Edvardsson, B. (2008). Customer-support service in the relationship perspective. *Managing Service Quality*, 18.

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

[Schmidt et al., 2013] Schmidt, M., Roux, N., and Bach, F. (2013). Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162.

[Starkweather and Moske, 2011] Starkweather, J. and Moske, A. K. (2011). Multinomial logistic regression.

[Sá et al., 2016] Sá, J., Almeida, A., Pereira da Rocha, B., Mota, M., De Souza, J. R., and Dentel, L. (2016). Lightning forecast using data mining techniques on hourly evolution of the convective available potential energy.

[Tan et al., 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Addison Wesley.

[Trawiński et al., 2012] Trawiński, B., Smundefinedtek, M., Telec, Z., and Lasota, T. (2012). Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms. *International Journal of Applied Mathematics and Computer Science*, 22(4):867–881.

[Utgoff, 1986] Utgoff, P. E. (1986). *Machine Learning of Inductive Bias*. Kluwer, B.V.

[Yiu, 2019] Yiu, T. (2019). Understanding random forest. https://towardsdatascience.com/understanding-random-forest-58381e0602d2. Last accessed on Jul 10, 2020.

[Zheng and Casari, 2018] Zheng, A. and Casari, A. (2018). *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 1st edition.