Supervised Learning Methodologies to Improve Customer Support

Inês F. Marques
inesfmarques@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

December 2020

Abstract

This article is about classifying technical issues from customer support and 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 using a previously collected dataset which consists of labeled technical issues, where 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 train several popular classifiers and choose the one with the best generalization score. We evaluated the models 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 best results were produced by Neural Networks achieving almost 70% of accuracy. and 90% of Top 3 accuracy. Since we are looking for a way to help the assistant, having a set of 3 possible causes to choose from, where one of them is correct with 90% confidence, is a better option 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: Telecommunications, Classification, Supervised learning, Class imbalance

1. Introduction

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. 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. 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?

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 which allows the system to set automatically the *background* of the problem. 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 subservices 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.

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.

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.

Classification is one of the most popular topics of machine learning, with a broad array of applications. The number of available methods has increased and logically there is a need for method comparisons to find the best one for different situations, as showed in [3], [2] and [14]. 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.

2. Background

According to [1], 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 $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{M}$ which is called the training set and where M is the number of training observations. Each training input $\mathbf{x}$ is a vector of numbers called features, explanatory variables or covariates, where $\mathbf{x} \in \mathbb{R}^N$. 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. $y \in \{C_1, \dots, C_K\}$ 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. In addition, 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, $\hat{y} = \hat{f}(\mathbf{x})$. 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. In the following sections we discuss some of the most popular multiclass classification algorithms.

2.1. 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 [9] 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.

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

2.2. Softmax Regression

Softmax regression (SR) is an extension of 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:

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

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, e.g. gradient descent. For that we need to define a cost function that will be minimized. As [1] suggests, we use the cross-entropy error function for the multiclass classification problem. Cross-entropy measure is a widely used alternative for the mean squared error when the output of the algorithm is a probability distribution.

2.2.1 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. In a DT, 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.

Most algorithms 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 into successively purer subsets - i.e., subsets with instances from less different classes. The algorithm must provide a measure for evaluating the goodness of the condition in each internal node such as entropy, Gini and classification error. Moreover, 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.

[13] praises DT as being simple to interpret and fast classifiers. On the other hand, [6] states that DT are unable to extract linear combinations of features and that over-complex trees could overfit to the data. 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. Furthermore, if the class distribution is imbalanced, the generated decision tree could generalize poorly to the minority classes. Finally, DT can be unstable because small variations in the data might result in a completely different tree being generated.

2.2.2 Random Forest

Random Forest (RF), as the article [15] 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.

2.2.3 Neural Networks

Neural Networks, as [1] explains, are 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.

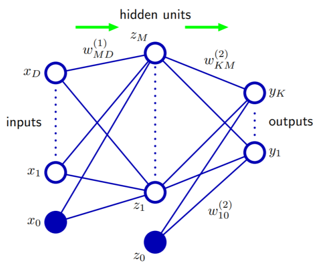


Figure 1: Scheme of a feed-forward neural network. Image from [1].

In the case of a neural network with only one hidden layer as pictured in Figure 1, we first take the D input variables of an observation and construct linear combinations of the form

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

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:

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

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:

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

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 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*[5].

3. Data Preprocessing

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. All the data analysis in this work was done in R software environment for statistical [11]. 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 work.

The dataset provided by Border Innovation contains real cases from a telecommunications company technical support. It consists of a table *observations* × *features*. The columns are divided into the response variable and the predictor variables. As mentioned in the introduction, 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.

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 wasn't 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.

The initial dataset had 192684 rows and 582 columns. 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. Then, we 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, there were still 124 constant columns equal to 0. One possible explanation for this is the fact that the variables are in dummy 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. We also checked whether each variable had the correct domain type and range. 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. Following this preliminary analysis, we looked for inconsistencies in some of the four groups of the predictor variables.

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. Table 1 presents the results. 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.

Test	**Conflicts**
D38 = D39?	2
D40 = D44?	0
D42 = D43?	0
D49 = D50?	275
D56 = D57?	198
D336 = D337?	4298
If D70 = 1 then D69 =D71 = 1?	70
If D73 = 1 then D72 =D74 = 1?	53

Table 1: Tests associations

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.

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. From all the test variables related to each label, some of which somehow indicate the presence of an error in the service in question, 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.

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%.

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.

3.1. 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. Table 2 shows the class distribution counting how many classes belong to a certain range of number of observations in the dataset:

Range of frequency	Number of classes
$[1, 10[$ observations	56
$[10, 100[$ observations	22
$[100, 1000[$ observations	28
$[1000, 10000[$ observations	7
$[10000, +\infty[$ observations	1

Table 2: Number of classes per range of frequency

It is clear that the class distribution is severely imbalanced. The label with the higher frequency is the class 153, which corresponds to *Undetermined Cause*, and it holds almost half of the data points. The other half of the data points are allocated to the remaining 113 present classes. In addition, the majority of the classes are almost negligible, meaning they have less than 10 observations in the whole dataset. 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. 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

We attempt to train the models seen in Section 2, since it is hard to guess which one will perform better, using the final version of the dataset. 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. Scikit-learn was the main library used for model fitting, model selection and evaluation, available at [4]. The code for this chapter is available at GitHub in the file `Model.ipynb`.

First, we partition the dataset into training data and test data. Then, we split the training data again, applying what is called the k-fold cross-validation. The training data 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. The final result is usually the average of all the scores. According to [13], 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.

In order to assess a classifier performance, we must use an appropriate metric. The most popular metric is **Accuracy** which is the fraction of correctly classified data. **Precision** is, for a certain class C_k, the proportion of observations which the model classifies with C_k and that are actually labeled as C_k. **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. The F1-score combines both recall and precision and it is the harmonic mean between the two:

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

According to [10], 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.

4.1. Model Comparison

We will compare between Naive Bayes, Softmax Regression, Random Forest and Neural Networks. The models' parameters are tuned in 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 3.1, 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 split the whole dataset into training set (80%) and test set (20%). 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).

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, `alpha`. The best `alpha` is 100 with the overall weighted F1-score being around $\mathbf{44,3}\%$. This poor result led us to assume 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`. The best `alpha` is approximately 184 with the overall weighted F1-score still being low, around $\mathbf{46,7}\%$. Furthermore, the models are classifying a lot of unseen data with the four most frequent classes which is evidence that the models are too closely fit to the training set and there is no predictive power for the minority classes.

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 Stochastic Gradient Descent (SGD) as the weight optimization solver, but instead it is able to learn from other gradient-based solvers like Stochastic Average Gradient, which is a variation of SGD with faster convergences rates[1]. The grid search with cross-validation on `LogisticRegression` yielded a best score of $\mathbf{60,8}\%$ with the LBFGS[2] solver and inverse of the regularization parameter, `C`, close to the default value which is 1. In comparison to Naive Bayes, although the model still has a lot of generalization errors with the most frequent class, the higher score is a sign of less chaos within the other classes.

[1] Please refer to [12] for more information on SAG

[2] 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

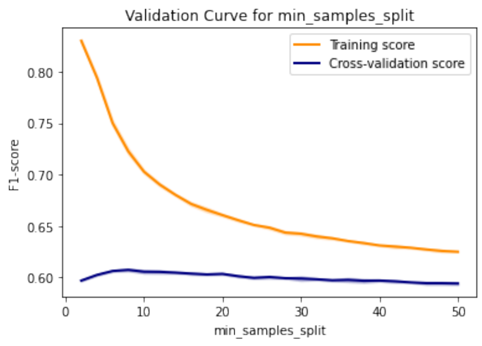
Random Forest

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. 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.

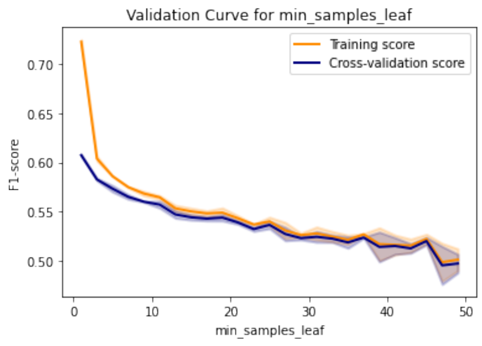
First we perform a grid search on the parameters `bootstrap`, `max_features` and `criterion` as well as on the number of decision trees that make up the ensemble, `n_estimators`. The best model uses bootstrap samples and fits 75 decision trees with the Gini criteria and the default `max_features`. The 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. Figure 2 compares both the training score and the validation score: In Figure 2(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 2(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 $\mathbf{60,7\%}$. The overall weighted F1-score, are very close to the `LogisticRegression` model. The classification errors, as before, are 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.



(a) `min_samples_split`



(b) `min_samples_leaf`

Figure 2: Validation curves for both parameters with `RandomForestClassifier`

Neural Networks

An alternative to the scikit-learn library when it comes to neural networks is to use Keras, which is a deep learning programming interface written in Python running on top of the machine learning platform TensorFlow. Ultimately, the selection of an architecture for the neural network will come down to trial and error.

We will perform parameter tuning with cross-validation manually, increasing 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. This type of layer is usually implemented as the first hidden layer. The activation function to be used will be the rectifier, ReLU and 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 [8] argues that the death of neurons can be avoided by allowing a small gradient when the unit is not active. Finally, the Dropout layer helps prevent overfitting. It randomly sets input neurons to 0 with a frequency of the parameter `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 `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 [7] 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.

The NN that yielded the best weighhted F1-score is as follows: the input layer receives the 565 covariates and the first Dense layer is responsible for transforming those neurons using 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 $\mathbf{62,72}\%$. 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 and the models are not able to learn the minority classes so well.

We mentioned the fact that Border Innovation was interested in another type of metric 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 $\mathbf{90,77}\%$. We also computed the Top 3 accuracy for the other discriminative models studied in this section 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.

5. Conclusions

This work was set to answer 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. 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. Another challenge was 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.

The proposed model consists of a neural network with 2 hidden 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, 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. 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.

References

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

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

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

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

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

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

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

[8] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

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

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

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

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

[13] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison Wesley, May 2005.

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

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