Fault Prediction in Networks from Service Providers Using Machine Learning*

Mauricio de Oliveira
Computer Science Department
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Minas Gerais, Brazil
oliveiramauricio@dcc.ufmg.br

ABSTRACT

Machine Learning has the potential to impact our lives in many different ways, whether by making companies more competitive or facilitating several of our daily tasks. In this paper, machine learning algorithms are used to predict service faults on a network owned by a Telecommunications company. We detail all the process behind building two machine learning models, as well as our assumptions and rationale, and we show that our models have great performance.

CCS CONCEPTS

• **Machine Learning** → **Supervised learning**; • **Data Science**; • **Networks** → Network reliability;

Reference Format:
Mauricio de Oliveira. 2018. Fault Prediction in Networks from Service Providers Using Machine Learning. In *Proceedings of Computer Science Department, UFMG*. UFMG, Belo Horizonte, MG, Brazil, 7 pages.

1 INTRODUCTION

In this paper, we describe the Data Science process [1] applied to data from Telstra Network [4], an Australian Telecommunications company who hosted an on-line competition available in [2]. The goal of the competition was to have participants submitting their work that uses a range of information to predict service faults, that is, a time and a location where the provider had issues in servicing their customers.

More specifically, the problem asks to predict the *fault severity*. The fault severity is defined by:

- Fault severity 0: no fault
- Fault severity 1: few faults
- Fault severity 2: many faults

As a result, our problem is a *multi-class classification problem* where we must build a model using *supervised* learning algorithms capable of telling which of the three classes an example belongs to. The model is evaluated using the *multi-class logarithmic loss* (or simply *logloss*) formula. Each data row of the dataset provided has been labeled with **one** true class. For each row, we need to calculate a set of predicted probabilities (one for every fault severity). The multi-class logarithmic loss is given by:

$$logloss = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \ln(p_{ij}) \qquad (1)$$

where N is the number of rows in the dataset, K is the number of classes, which in our case is 3, *ln* is the natural logarithm, yij is 1 if observation i belongs to class j and 0 otherwise, and pij is the predicted probability that observation i belongs to class j.

The outline of this document is organized as follows. In the rest of this section we state the objectives of our work and explain the motivation behind it; in Section 2 we detail our methodology, and in Section 3 we start describing the data science process, beginning with the Exploratory Data Analysis (EDA) where first we provide an overview of the data. Section 4 is dedicated to Machine Learning, that is, we describe the process that resulted in the models built for this challenge. Later, in Section 5 is where results are shown and discussed and the paper is finalized by reporting some related work (Section 6) and expressing our conclusion (Section 7).

1.1 Objectives

Although in the original competition the objective was mainly the performance of a model on unseen data, this is only one of the objectives of this project. We also intend to:

- build different prediction models and compare them with respect to execution time, validation score and *logloss* score.
- verify the effects of common machine learning problems, including but not limited to *overfitting* and *underfitting* and apply techniques to mitigate them.

1.2 Motivation

While searching datasets to use for this activity, the Telstra challenge spiked interest for the following reasons. First, the dataset is provided by a real company, which means there is the opportunity to solve a real world problem. Second, part of the data comes from logs, which are reports from actual networking devices, such as routers, base stations, and other equipment sent to the company's monitoring center, and the author is from the networking field. In addition, the idea of using machine learning to predict service faults is brilliant by itself and customers should be happy in knowing that there can be a system behind their subscribed service capable of pro-actively responding to network failures.

*Produces the permission block, and copyright information

2 METHODOLOGY

The methodology applied in the development of this activity is as follows. First we follow the steps known as the data science process which consist of acquiring and preparing data, analyzing it, building a model, evaluating it and visualizing. However, since we already have the data, our focus is mostly directed towards the analysis and machine learning modeling. In terms of evaluation, we decided to use the same metric as the challenge, *i.e.*, the *logloss* error (Equation 1), although we can see that different *single evaluation metrics* could have been employed, for instance, a metric that synthesizes maximizing the accuracy while maintaining an acceptable level of false positives for *fault severity* 2.

We developed our model locally in a laptop running Linux 16.04, with a Intel Quad Core 2.40GHz CPU and 8GB of RAM. In terms of implementation, the Python language was used, more specifically the *pandas* library for data manipulation and analysis, *sklearn* for machine learning models and *matplotlib* for visualization. All these packages are known for their support for *vectorization*, which makes extensive use of parallelization thus computing tasks faster. If you are interested, the code is available on a online repository [1]; however, we believe that simply following the sections in this document is enough for your understanding.

Last, when analyzing the results, we had in mind certain upper and lower boundaries. We established what is a good and poor performance based on previous attempts. We deem this as important because in computing problems it is productive to know some details beforehand, specially problems whose limits are already established (best possible human performance, Bayes Optimal error, etc.). This rationale is similar to when we need to face NP-Hard problems - in this scenario, we should define reasonable parameters otherwise an attempt can be highly unproductive.

3 EXPLORATORY DATA ANALYSIS

Before starting with the EDA, we will first give an overview of the data. As mentioned earlier, the data contains log information but also location, event type, and so on.

However, it is important to mention that although we are given the datasets, the creators of the challenge did not care to provide much detail about it. Virtually no variable was given a description of what they represent. Perhaps they believed this would impact each participant' analysis in deciding whether or not a given feature was relevant to the prediction model. Nonetheless, next we offer a brief description of the data.

3.1 Data Overview

The challenge includes the following files:

- train.csv: the training set for fault severity (labeled)
- test.csv: the test set for fault severity (unlabeled)
- event_type.csv: event type related to the main dataset
- log_feature.csv: features extracted from log files
- resource_type.csv: type of resource related to the main dataset
- severity_type.csv: severity type of a warning message coming from the log

[1] Github: https://github.com/ausmauricio/telstra-network

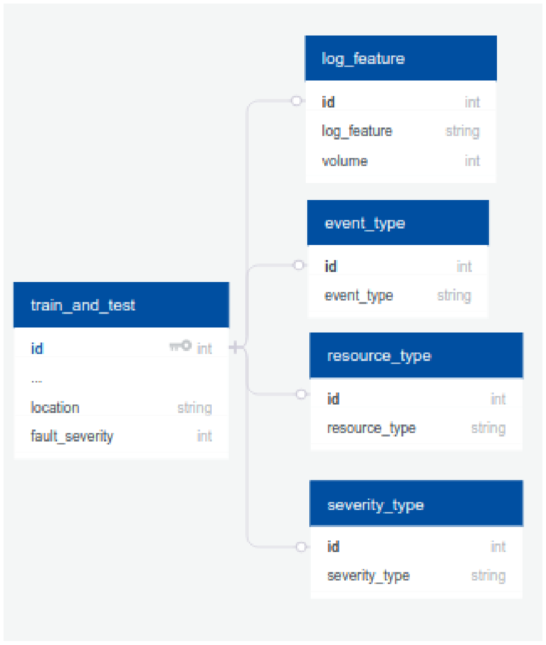

Figure 1: Representation of Telstra's challenge input data

The training set (and test set) relates to the other files via keys, which is the first column (named *id*) of the training and test sets and used in the other files like in a relational database. Moreover, there is typically a *one-to-many* relationship between the main files (train.csv and test.csv) and the others. Figure 1 summarizes the input data as a database schema [2].

A very important characteristic of the data is its type. Practically all of them are *categorical* and even the numerical ones are discrete, meaning they can be considered categorical as well. These facts are going to be very important in our EDA. In addition, features such as *event_type*, *log_feature*, *severity_type* are all valued with strings such as "type 1", "type 30" etc, but we are not aware what these mean. However, what really matters is to know how each of them impact on the fault severity. Therefore, during our exploratory analysis we want to answer questions such as: *"Is there a correlation between volume and fault severity?", "Do some events usually yield to more fault severity?", "Which locations fault severity 2 is more likely to occur?", "Is severity type and fault severity correlated?"*.

3.2 EDA

In this subsection we describe the Exploratory Data Analysis of the data used to to predict network faults. It is not intended to detail all the work done in this phase but instead we wish to inform the main contributions to the model this step was responsible for. We divide this section with early conclusions, gathered upon first glance on the data and later we describe the results from a deeper analysis on each variable's behavior that defines how we present the data to the prediction models.

[2] the train_and_test table does not correspond to a single table, instead this block represents both train and test files since they have the same composition and their relation to the other files is the same.

Table 1: Number of unique values for each feature

Feature	Number of unique values
location	929
resource_type	10
severity_type	5
event_type	53
log_feature	386
volume	341

Some early conclusions from our analysis are:

- the distribution of the features as measured in their original files is very similar to when applying histograms using only the training set keys. This makes us believe that the training set is not *skewed*, that is, it's a good sample of the data, which is certainly good news for our modeling. For example, the overall distribution of *severity_type* [3] and *resource_type* nearly repeats in the training data. The same is true for other features.
- Table 1 shows that when "merging" files to our main sets, we need to be careful with *one-hot encoding* so as not to *overfit* the training. We probably do not want 929 columns for location, 386 for log features and etc, but we should be fine with one-hot encoding for variables such as *severity_type*.
- Figure 2 shows us the distribution of *fault_severity* in the training set. In it, we can see that the classes distribution is very skewed. For our training set with size $N = 7381$, we have that approximately 9.8% of our examples are of *fault_severity* 2, which is the worst case for fault severity (1 corresponds to 25.3% and 0 to 64.8%). This situation suggests we might face the *class imbalance problem* [6], which is an issue that arises in many machine learning algorithms since they work better when a reasonable distribution of classes is available. Nonetheless, we hope that the amount of data we have is able to capture what it takes to classify a fault severity of type 2 (and the others) since we can not acquire more data.
- fortunately, we don't need to do much *data cleansing*, such as fixing typos and there does not seem to be any *missing data*.
- the training set contains only 7381 examples whereas the test set has 11171. This makes this challenge hard in the sense that we need to be careful not to make avoidable mistakes. Our training must be not only good but also as unbiased as we can make it, which means that techniques such as *early stopping* and *cross-validation* are very important.

The next part of our EDA is given next where we describe the behavior of some variables that should impact our models. However, since our next step is machine learning, and it is important to go back and forth between these two steps, we decided to leave some of information for the next section as well.

[3] not to be confused with *fault_severity*, which is the attribute valued at 0, 1 or 2 we wish to predict.

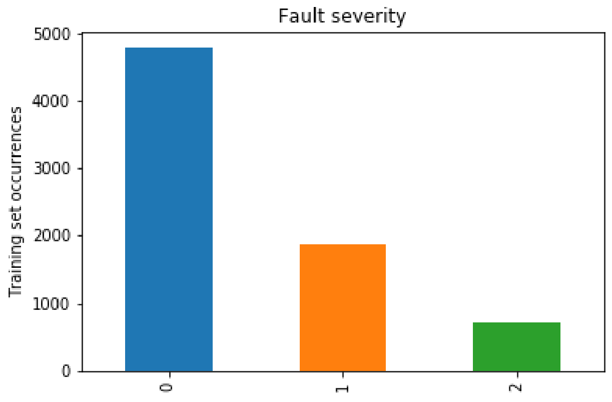

Figure 2: Fault severity histogram on training data

- luckily, some features take different values depending on the fault severity. For instance, the distribution of *log_feature* for entries where *fault_severity* is 0 is significantly different for entries whose *fault_severity* is 2. Same is also true for other variables such as *resource_type*. These facts assure us we are adding features capable of providing the machine learning algorithm with useful information for differentiating examples.
- the distribution of the location feature is intriguing. There are a total of 929 different locations; however, only 163 have experienced fault severity of type 2. In additional to some modest one-hot encoding for this variable, including a extra one that encodes the fact that a given location is one of the 163 "rare" locations can be useful.

4 MACHINE LEARNING MODELS

In this section we describe our modeling phase where we finally start training learning algorithms to classify our data. We use two different learning algorithms: Neural Networks and Random Forests. We assume the reader has previous knowledge about each and therefore we do not explain them. You can find more information about these algorithms on [7] and [5].

Our approach to build models can be summarized as: start simple and go from there. Overall, we start by building simple models and refine them as we progress. From analyzing the results, we decide what we should do next to help us build a better prediction model, that is, one with low logloss.

4.1 Neural Networks

We divide this subsection with description, analysis and conclusions of two different sets of NNs we trained. The first set is *simple* and consist of training made with variables already available in the data [4], that is, we did not create any new ones. The second set, on the other hand, contains features we added that we considered to have more "predictive power", after analyzing their behavior and impact.

[4] not considering simple one-hot encoded variables as new features.

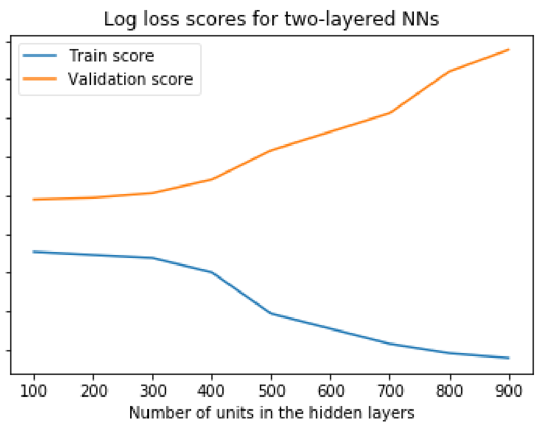

Figure 3: Train and validation logloss scores

The first set contains a number of features ranging from 300 to 700 that were obtained via simple one-hot encoding. That means our input layer had up to 700 units. The difference in the amount of variables encoded had to with including a number of features that corresponded to a certain percentage of the data. For instance, the 300 most popular locations out of 929 corresponds to 75% of the locations. The 300 least common locations happened only once or twice, and do not necessarily correlate to a certain fault severity so they do not add important information. Similar analysis were made for the other features to determine their respective number of one hot encoded variables.

Unsurprisingly, the more variables we included, better the training score, but not so much the validation one. This suggests overfitting except the validation score would not change by a significant amount as more variables were added, which is what we would expect with overfitting. This characteristic is suggestive of *multicollinearity* between some variables, which adds unnecessary information to the model but it doesn't necessarily make the prediction worse. In our case, overfitting was more visible with the size of the networks. Figure 3 shows a graph obtained using the training and validation scores during training, where we plot these metrics using different hidden layer sizes (two hidden layers of same size N).

We found that our "sweet spot" was two hidden layers equally sized with 300 units. Three layers proved to overfit easily, and a single hidden layer of the same size had similar performance, but we noticed that we were able to capture some information better with two layers, which we will mention later. In addition, we tested different parameters such as learning rate (for gradient descent), tolerance (for convergence), optimizers and activation functions. In the end we chose the combination that suited us best, including the Adam optimizer and the *relu* activation function to obtain non-linearity.

$$\begin{tikzcd}
BestScore \arrow[d] \\
TrainScore \arrow[d, "Gap/Variance"] \\
ValidationScore \arrow[d] \\
TestScore
\end{tikzcd}$$

The diagram above represents the performance of our model on different sets. We can see that our validation score *generalizes* well as it is close to the test error, which is calculated via a submission file containing the predicted probabilities for every *id* from the test set to the Kaggle website. This is great news but we can also verify that the distance between the training score and the validation score is evident, which we interpreted as being indicative of significant variance in our model. In the diagram we also display a best score representation, which is test score value we determined as good performance early on. For now we just show that the training score and the best score are very close [5] but later we will give more details about each of these scores.

The first set of Neural Networks (NNs) we trained also confirmed us we indeed had a problem of *class imbalance*. The errors occurred with higher proportion for fault severity 2 and 1 than for fault severity 0, relative to their proportion in the training and validation sets. Figure 4 puts this situation in perspective, which was measured using a two hidden layered network. For a single-layer network the distribution is even worse, which is one of the reasons we chose to work with two layers instead of one, and for three or more layers, it tends to get better (more evenly distributed), but it does so at the cost of more variance and poor generalization. In this scenario, we concluded we should engineer extra features that would help our learning algorithms classify correctly every class, specially "1" and "2".

Therefore, we defined our second set of NN trainings as the set we included variables we *engineered*. Some of these variables are [6]:

- common values of the features not only regarding overall presence in the training, but some exclusive ones only for *fault_severity* 1 and 2.
- number of times each id appears on *log_feature*. The idea behind this was that, considering the nature of the data, more log instances for an *id* could mean more severity, and the volume could imply severity as well, such as a warning

[5] this proximity also tells us we did not have significant bias in our model
[6] Some items could also go on the EDA section but as we said before we chose two describe them here.

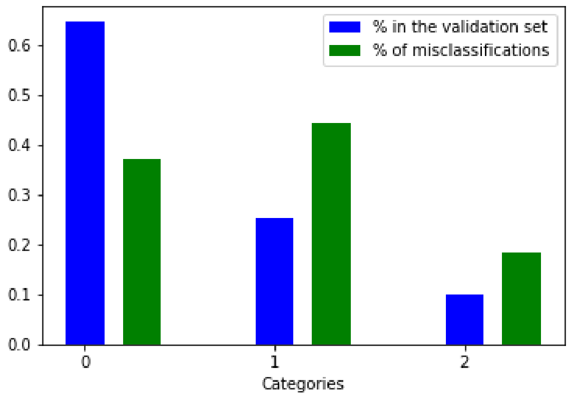

Figure 4: Proportion of errors for each class

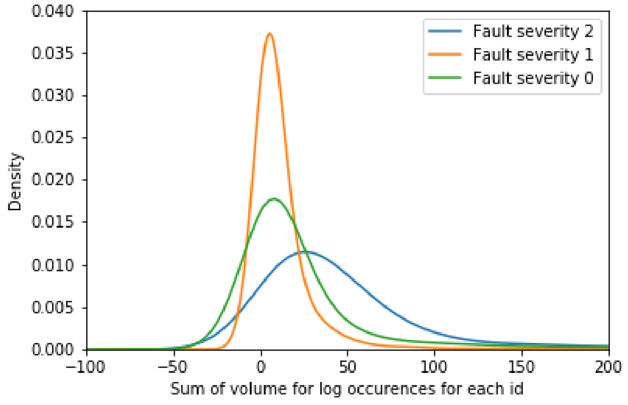

Figure 5: Distribution of the sum of volumes for each label

message being more frequently reported to the monitoring center. This addition proved good.

- the sum of the volume for each id. Since this is a numerical value, simple operation such as sum, mean, standard deviation can tells us something even if their very simple. Following the same rationale as the previous item, we thought the sum of volumes could be meaningful. It turned to be as well.
- number of times each id maps to resource types and event types sets.

It is important to show why we convinced ourselves that these variables were good. There are several ways of looking at data, and each different way leads to different conclusions. The way we investigated most variables was according to their behavior for each fault_severity, that is, how different they are considering each label. Therefore, we were looking at "x" given "y", which is a *Bayesian approach* that, by the time we did it, we weren't even aware of.

During our analysis, we investigated several variables but we focus on a few that offers interesting visualization. Figure 5 shows the distribution of the sum of volumes for each *id*, given their label. That means we have three different curves, and we see clearly a good fit for the Gaussian distribution. Since they have three distinct curves, we can use this information to add variables to our model. One approach would be simply using one column with their values but it seems we can do even better by assigning three probabilities to each value, one for each label, that tells the likely hood of that instance belonging to one of the classes as follows:

$$prob(i = k) = prob(k) * \frac{1}{\sigma_i \sqrt{2\pi}} e^{-(x_i - \mu_i)^2 \Big/ 2\sigma_i^2} \tag{2}$$

where i represents the i-th example, $prob(i = k)$ is the calculated probability of instance i belonging to class k (0,1, or 2), $prob(k)$ is the overall probability of an class, considered the same as its percentage on labeled data [7], x_i the value of the sum of the volumes for the i-th example, σ_i is the standard deviation of the sum of volumes for i-th, and μ_i is its mean.

[7] information of Figure 2

Figure 6, one the other hand, does not fit any well known distribution but it gives an extremely valuable insight. The curves, which tells the density of each location according to different labels, shows regions where each *fault_severity* is more frequent. The density for *fault_severity 2* and 1 grows as the location *number* is bigger and the opposite happens for fault_severity 0. The location number, which is a categorical value, appears to be ordered. This suspicion is validated by the fact that the organizers said that there was time information on the sets. However, this information was not really on any of them, at least not explicitly. Hence, considering the order of the locations, we established that their numbers were sorted in time, which means we extracted the time information from the sets that the organizers claimed to be there.

This conclusion lead us to consider the location feature as numerical, which excludes the necessity of one-hot encoding for a huge number of columns. Our training sets had now much less features and no performance loss was visible. This made us a lot more comfortable with our sets, since now we only needed one normalized column for the location, instead of hundreds. In addition, we now needed smaller neural networks.

All these additions enabled us to reduce the gap between the training score and validation score, but the neural networks started to become too sensitive. Our trainings required a lot of parameter tunning and even early stopping wouldn't stop them from training too much. The first set (probably) did not have this problem, even though they were trained using even larger number of features, because the columns where very sparse (too much zeros and very little ones). The second set, however, had more dense variables (probability values, sums, etc) and could be the reason for this sensitivity.

4.2 Random Forest

As we added more variables to our models (what we call the *second sets*), the Neural Networks began to have problems with overfitting. Hence, we decided to migrate to models less prone to overfitting. Random Forest is appropriate for that purpose.

Our performance using Random Forest present in the *sklearn* package was very successful in that the training generalized well.

Table 2: Performance results

Set	Characteristics	Model	Execution Time	Validation Score	Test Score
1	one hot encoded variables	NN	1s	0.62	0.63
1	one hot encoded variables	RF	10s	0.61	0.62
2	one hot encoded variables, probability values, density values	NN	1s	0.57	0.58
2	one hot encoded variables, probability values, density values	RF	10s	0.53	0.53

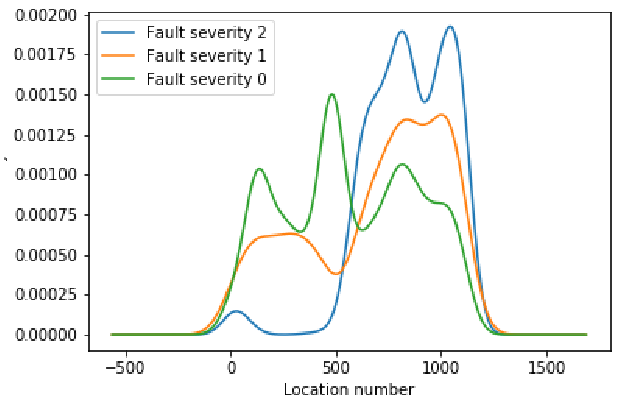

Figure 6: Distribution of location values for each label

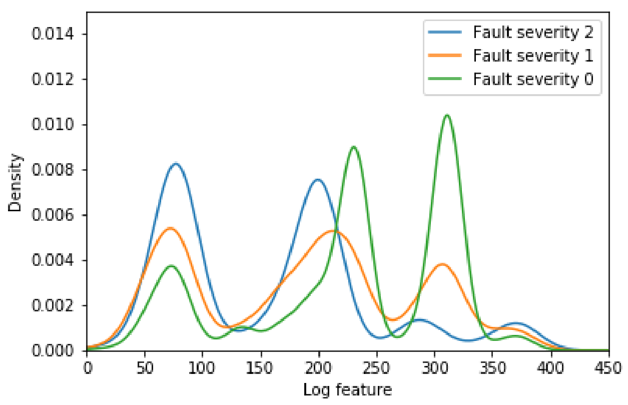

Figure 7: Distribution of log feature values according to each label

We had to spend less time with tuning features but we still felt we could build a better predictive model. Usually the most important parameters was the number of trees and the number of features considered each time.

We decided then to try and include more variables. It still had to do with log information, and instead of one-hot encoding log features, we would insert the respective volume. Upon careful analysis we saw that for each id, there was only one (log_feature, volume) mapping, so our intuition was that the log_feature alongside its volume would have more predictive power instead of just binarization. Indeed, the distribution of log_features according to fault_severity varied, as shown in Figure 7. We can see that the Gaussian beams are in different parts of the graph, with fault_severity 2 more at the beginning and fault_severity 0 more concentrated by the end, and fault_severity 1 somewhere "in between".

This new feature had modest impact on the model. It also required scaling because the values where very dynamic, so we we used the *MaxMinScaler*. At this point we were hoping to break a certain *logloss* barrier, but we only got closer. Hence, In the next section, we comment on the results.

5 RESULTS

In this section, we show the results for both models and both set of training data. Table 2 summarizes the results. The set number 1 had between 400 to 600 features, and the set number two only 297. The *bestScore* we mentioned but not defined earlier was established as 0.4. This is the logloss score of the first positions of the competition. Your training data and our models are capable of getting up to 0.53 logloss, which is not bad, but we were hoping to get lower than 0.5. Nonetheless, we are happy with our results and we certain than it can be easily improved.

As of now, we are sitting at 0.53 logloss. As a future possibility, we can continue to enhance this model, as now we have more expertise in these libraries and other learning algorithms. Specifically we should try building an ensemble as they are a good way improve precision while maintaining low variance. In addition, there are several techniques that affect performance we could try, such as different scalers.

6 RELATED WORK

Machine Learning and networks intersect not only in fault detection systems but can also be used to affect the way Internet Service Providers route data. The company Cisco, one of the most important Internet equipment manufacturer, is building routers and switches that gather user preferences and are used to define how routing and switches should be executed. Such technology is being marketed as intent-based networking [3], as has machine learning as its one of main components.

7 CONCLUSION

This document shows all the steps while building prediction models for the task of fault prediction. During this process, several conclusions were drawn.

First, Neural Networks are very prone to overfitting and very sensitive to parameters and hyperparameters, which sometimes makes it hard to use them when you want your model to generalize really well.

A limitation of our approach is that we do not create completely different training sets for different models. A training set could be optimized to run only on a Neural Network instead of also being used to run Random Forest.

In addition, building models require significant savviness in language packages, specially *pandas*, which at times does not even seem Python. A considerable part of the work was getting familiar and achieve some expertise in it, which requires time and effort due to its learning curve. This shows how interdisciplinary Machine Learning is, and even if you are very good at Mathematics, Statistics, or know a lot about your data, your models' performance can be potentially constrained by your level of expertise in machine learning libraries.

REFERENCES

[1] 2013. Data Science Workflow: Overview and Challenges. https://cacm.acm.org/blogs/blog-cacm/169199-data-science-workflow-overview-and-challenges/fulltext. Accessed: 2018-06-21.

[2] 2015. Kaggle Datasets. https://www.kaggle.com/c/telstra-recruiting-network/data. Accessed: 2018-06-15.

[3] 2018. Intent-Based Networking. https://www.cisco.com/c/en/us/solutions/intent-based-networking.html. Accessed: 2018-06-26.

[4] 2018. Telstra. https://www.telstra.com.au/. Accessed: 2018-06-15.

[5] Rich Caruana and Alexandru Niculescu-Mizil. 2006. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*. ACM, New York, NY, USA, 161–168. https://doi.org/10.1145/1143844.1143865

[6] H. He and E. A. Garcia. 2009. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering* 21, 9 (Sept 2009), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

[7] S. B. Kotsiantis. 2007. Supervised Machine Learning: A Review of Classification Techniques. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 3–24. http://dl.acm.org/citation.cfm?id=1566770.1566773