CatBoost for Fault Impact Analysis:

Predicting an NE’s average data rate change when a fault occurs

Stephen Kolesh
Multimedia University Of Kenya

Abstract ー *In the current practices with large-scale and complex network structure, O&M engineers have to face massive faults & alarms in daily work. The traditional way to analyze faults is by setting rules based on network experts’ experience, such as duration of faults or predefined categories of faults, to determine which faults need to be handled with higher priority. In this research we seek to implement a ML approach to be able to predict a network element’s average data rate change when a fault occurs.*

1. Introduction

The goal of fault management in telecom O&M (Operation & Maintenance) is to ensure stable & reliable networks and services. In the RAN (Radio Access Network), the most significant part of O&M activities is network fault management, including fault monitoring, analysis, diagnosis, and repair processes. Among these processes, fault analysis is an essential part of troubleshooting.

In the current practices with large-scale and complex network structure, O&M engineers have to face massive faults & alarms in daily work. The traditional way to analyze faults is by setting rules based on network experts’ experience, such as duration of faults or predefined categories of faults, to determine which faults need to be handled with higher priority.

However, with the traditional method, the impact of each fault on network KPIs (Key Performance Indicator), such as coverage and data rate, are not explicitly assessed and taken into consideration. As a result, the reliability of network service cannot be quantified and it is impossible to optimally schedule O&M resources.

In this research we are going to exploit the power of CatBoost to predict the impact of fault on RAN KPIs. Specifically we are going to focus on predicting an NE’s average data rate change when a fault occurs. Moreover, we will make a brief analysis of the results and propose alternatives to improve the actual solution.

2. Dataset

All the data used during this research has been provided by ITU.

The provided dataset was composed of RAN KPI data, collected from over 100 5G and 4G NEs. Specifically we had 7,256 Train NE’S and 1932 Test NE’S. For each NE in the network, it provides 7 key KPIs, the duration of a fault, and the “distance” of the NE to the fault on an hourly basis. Specifically, each row includes:

- NE ID, which is a unique identifier for each NE;
- Hourly timestamp of the data;
- 6 RAN KPIs that is related to data rate, including Access Success Rate, Resource Utilizing Rate, TA (Time Advanced), BLER (Block Error Rate), CQI (Channel Quality Indicator), MCS (Modulation and Coding Scheme);
- Data rate of the hour;
- Fault duration (in seconds) during the hour.
- “Distance” (Relation) of the NE to the fault,

2.1 Pre-Processing Data

I had to structure the input and output as specified by the organizers:

Input:

Data rate and other features collected before the fault occurs, where fault duration is 0 and fault duration and relation in the first hour during which the fault appears.

Output:

The dataset provided had no predetermined labels and we were supposed to create the Labels for ourselves based on the rules provided by the organizers. The rules specifically were:

- The status of data rate change is defined as whether the value of data rate being less than the value right before the fault(i.e the fault duration is larger than 0). If so, the state is labeled as ‘1’, otherwise, the state is labeled as ‘0’.

2.2 Data cleaning

It is often the case, when working with large amounts of information, that among the values we have in our dataset some are corrupt, or simply have no value at all. In our case, in each NE we had multiple rows, but we were only interested in one row when the fault actually occurred. Then at that row we compare the data rate of the previous non fault row(fault duration zero) to the faulty row’s data rate, if the data rate decreased we label it a 1 or 0 if not. This shows that we were supposed to be having one row per file and which was further clarified during the challenge i.e train 7,256 rows, test 1932 rows . All other rows were dropped. I also dropped train rows where the calculated label was null, which was as a result of no previous row to compare the data rate with resulting in 7,224 train rows. In the test we never had this scenario.

2.3 Feature Engineering

My feature engineering process was pretty simple. As it had been specified, we were supposed to use the RAN KPIs data of the previous non-faulty row. I did exactly that. The only exception was the data they told us to use that was part of the faulty row.. These were fault duration and relation. This made no sense since these values are hindsight i.e we understand this when fault has happened already. At prediction time we don’t have these values and so I dropped them from my pipeline because I only wanted to use the data that was present before the fault occurred.

Other features I added were:

- data_rate_prev - this was the data rate of the previous 2 timestamps
- data_rate_prev_3 - this was the data rate of the previous 3 timestamps
- count - this was the count of the total timestamps that had passed before the fault occurred. My reason to do this was I thought I would get the relationship of the count to the relation attribute since I had dropped it from the features I was going to feed in my model.
- data_rate_mean - this was the mean of the data rate of all non faulty rows: i.e the data rate of rows where fault duration was zero
- hour - the hour when the fault occurred
- day - the day when the fault occurred
- month - the moth when the fault occurred
- part_of_day - the session when the fault occurred i.e morning or afternoon or evening

I also dropped access_success_rate as values were pretty constant in most of the Network Elements. At the end of the Feature Engineering process I had these features that I was going to feed to my model:

- *resource_utilition_rate, TA, bler, data_rate_mean, cqi, mcs, data_rate, day, part_of_day, month, hour, data_rate_prev, data_rate_prev_3, count*

3. Modeling

Once the dataset is ready to be used, the second phase in which we find ourselves is “how” the selected information should be used to predict average data rate change when a fault occurs. For this I chose a gradient based decision tree approach specifically catboost (Dorogush et al., 2018).

3.1 Why CatBoost?

One unique characteristic of CatBoost is that it uses symmetric trees. This means that at every depth level, all the decision nodes use the sample split condition. This symmetry can help in balancing the decisions made across the tree, potentially leading to improved generalization and robustness in the model.

I tried all other GBDT models like Lightgbm and Xgboost but at the end I found better and generalizable results using Catboost and so I ended up using Catboost

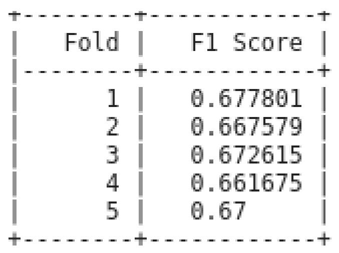
3.2 Cross Validation

Since this was a classification task I used Stratified KFold with 5 folds. I stratified based on a custom column called skf instead of the target column. The skf column was a string combination of the target column and a binned relation column. The relation column was binned based on whether the fault occurred exactly at the NE , the fault occurred at a neighboring NE with weak adjacency and neighboring NE with strong adjacency I used this column since I saw similar relation distributions in both the train and test datasets and I wanted my model to generalize well in each of those cases and also generalizing on the target column too.

3.3 Evaluation Metric

F1-score was used to evaluate the performance of the model. The F1-score which balances precision and recall, is a fundamental metric in classification tasks, particularly when dealing with imbalanced datasets.

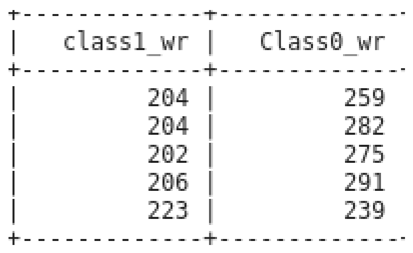
3.4 Initial Results Without Post Processing



```
+--------+-----------+
|  Fold  |  F1 Score |
|--------+-----------+
|      1 |  0.677801 |
|      2 |  0.667579 |
|      3 |  0.672615 |
|      4 |  0.661675 |
|      5 |  0.67     |
+--------+-----------+
```

Figure 1: F1 Scores without thresholding

The figure above shows the F1 score across folds when using the default threshold of 0.5 to convert raw probabilities to Labels.

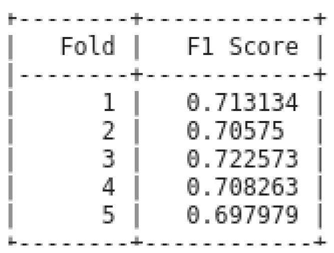


```
+-------------+-------------
|  class1_wr  |  Class0_wr
+-------------+-------------
|         204 |         259
|         204 |         282
|         202 |         275
|         206 |         291
|         223 |         239
+-------------+-------------
```

Figure 2: Distribution of False Negatives and False Positives without thresholding

The figure above indicates a table of class1_wr i.e wrongly classified class 1 and class0_wr wrongly classified class 0 when using the default threshold of 0.5.

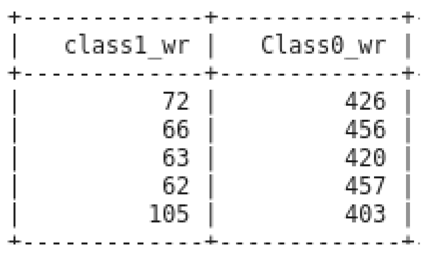
3.5 Initial Results With Post Processing



```
+--------+-----------+
|  Fold  |  F1 Score |
|--------+-----------+
|      1 |  0.713134 |
|      2 |  0.70575  |
|      3 |  0.722573 |
|      4 |  0.708263 |
|      5 |  0.697979 |
+--------+-----------+
```

Figure 3: F1 Scores with thresholding

The Figure above shows the F1 scores across folds when using a tuned threshold of 0.38 to convert raw probabilities to labels



```
+-------------+-------------+
|  class1_wr  |  Class0_wr  |
+-------------+-------------+
|          72 |         426 |
|          66 |         456 |
|          63 |         420 |
|          62 |         457 |
|         105 |         403 |
+-------------+-------------+
```

Figure 4: Distribution of False Negatives and False Positives without thresholding

The figure above indicates a table of class1_wr i.e wrongly classified class 1 and class0_wr wrongly classified class 0 when using a tuned threshold of 0.38.

3.6 Post Analysis Conclusion

The second set (with post processing) has significantly fewer false negatives on average compared to the first set. This indicates that the second model with thresholds is better at identifying instances of class 1.

Given these results, the second set(with post processing) benefits from significantly better recall due to fewer false negatives, even if it suffers a bit in terms of precision due to the higher number of false positives. This trade-off seems to favor the F1 score for the second set, making it higher across the folds compared to the second set.

Finding the optimal threshold is of paramount importance in the pursuit of maximizing the f1 score. The choice of threshold directly impacts the trade-off between false positives and false negatives, influencing the precision and recall values accordingly.

By systematically adjusting the threshold , it's possible to find the sweet spot that aligns with the project's goals and requirements, leading to an optimized F1 score and a more accurate predictive model.

This should be done after a careful consideration of the specific problem domain, the consequences of false negatives and false positives , and the desired balance between precision and recall.

3.7 Feature Importance

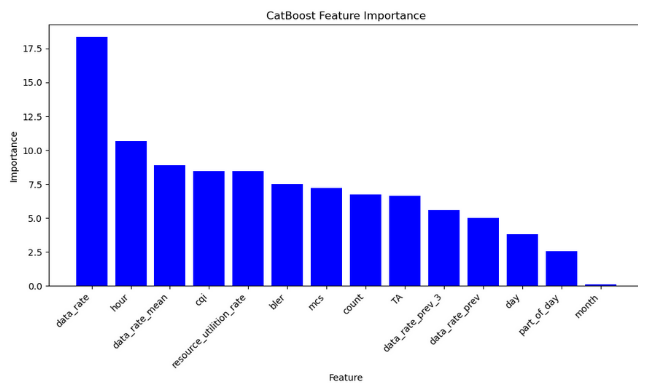


Figure 5: Feature Importance Plot

The figure above shows how each individual feature influenced my model. The most influential were the data rate, hour signifying time importance and data rate mean.

4. Conclusion:

In summary, this research delves into a novel perspective for addressing the challenges posed by fault analysis in modern, intricate network setups. Through the application of machine learning techniques, specifically predicting the average data rate changes triggered by network faults, the study aims to equip O&M engineers with predictive insights. This shift from conventional rule-setting approaches can foster a dynamic fault management framework, enabling proactive responses and informed decision-making that aligns with the demands of complex network operations.

References

1. Dorogush, A. V., Ershov, V., & Gulin, A. (2018, October 24). *[1810.11363] CatBoost: gradient boosting with categorical features support*. arXiv. Retrieved August 24, 2023, from https://arxiv.org/abs/1810.11363