Anomaly detection in 4G cellular networks: Explore ML solutions for the detection of abnormal behaviour of eNB

Gerard Castell*, Sergio Gimenez*
*Universitat Politecnica de Catalunya (UPC)
gerard.castell@estudiantat.upc.edu, sergio.gimenez.anton@estudiantat.upc.edu

***Abstract*—Detect abnormal behaviors in the utilization of the network is a current challenge in order to build optimal configuration of base stations of next generation cell communications. Binary classification problem is a common issue to resolve through a Machine Learning approach. In this report, the authors explore a set of well known data processing and machine learning techniques do deal with that problem, going from data pre-processing analysis, techniques to solve the classification, fix the miss-prediction and how the whole training process including parameter tuning is carried out. Using XGBoost as a classifier leads to an accuracy higher than 99% is achieved.**

The whole implementation of the model as well as detailed data visualisation can be found in [1].

***Index Terms*—XGBoost, Ensemble Methods, Binary Classification and Parameter Tunning**

I. Introduction

The purpose of this report is to solve a binary classification problem proposed as a competition in the Kaggle InClass platform [2], where we have applied some of the concepts and techniques studied in class combined with other methodologies researched for exploratory data analysis, feature selection and classification.

One of the main topics on the design of a cellular network consists on the optimization of energy and resources that guarantees a smooth operation even during periods with higher traffic load. Therefore, as time goes by cellular networks are attempting to establish a dynamic management and configuration in order to adapt the varying traffic demands in the most efficient way. This way, the network avoids the over-provision of resources and energy savings. Hence, the task is defined clearly: to build a network operator capable of anticipating to those variations in the users' traffic demands to get the most optimized management of network resources.

To do so, a Machine Learning approach is carried out in order to explore the possibilities to detect unusual behaviors in the utilization of the network that would motivate a change in the configuration of the base station.

II. Feature Analysis

First thing to do regarding data pre-processing is to make sure all features are ready to be used, specially the non numerical ones: `Time` and `CellName`. `Time` has relevant correlation with the maximum traffic hours, although the day is not included in the date, it is still a relevant enough feature. In order to make `Time` ready to use, there are several approaches. However, the chosen one has been to convert the time (minutes) to radians and split them up into two features, one applying a cosine and the other with sine. Regarding `CellName`, a simple mapping 1:1 with a numerical identifier for each cell name has been carried out.

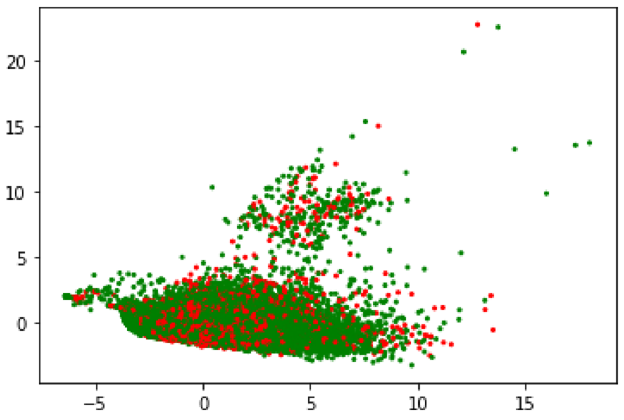

Fig. 1. Normalized PCA reduction in 2 dimension of the whole dataset

Once the whole dataset is ready to be used, we considered to explore the data in order to look at distributions and seek for potential correlations between variables. Even this dataset has only 13 features (14 after the sanitizing stage described previously), for human understanding is usually more suitable to make a graphical representation. This is done by doing a projection of that features in a 2D or 3D space. With the aim of achieving such dimensionality reduction, we used PCA (Figure 1) and t-SNE (Figure 2).

Regarding PCA, given the number of new features desired, tries to provide the projection using the correlation between some dimensions and keeping the maximum amount of information about the original data distribution. As can be seen in Figure 1, there are not clear clusters nor a clear defined patterns. Usual and unusual samples are mixed in many ways, fact that constraints the classifier that is going to be used, e.g linear classifiers can be directly excluded.

In order to go through a different approach, a non-linear reduction such as t-SNE has been tried out. T-SNE is an unsupervised method that minimizes the divergence between

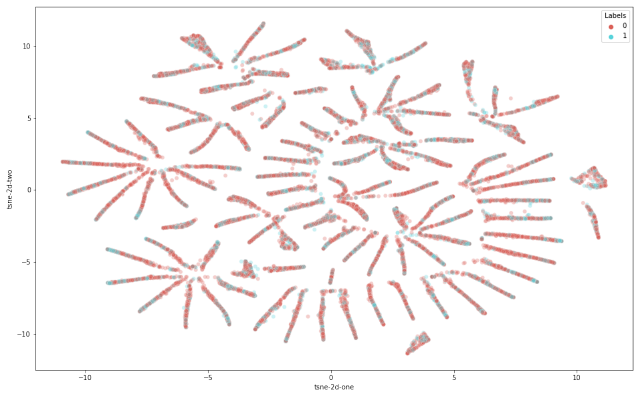

Fig. 2. Normalized t-SNE reduction in 2 dimension of the whole dataset

a distribution that measures pairwise similarities of the input and a distribution that measures the similarities of the corresponding low-dimensional points in the embedding. As it can be observed in Figure 2, t-SNE has built a set of separable clusters but within samples of both classes mixed, without a clear visual pattern.

At this point, a priori feature reductions seemed like it did not give us an advantage before starting the training of the model. However, after choosing a model and tune its hyperparameters as will be explained in detail at section III, we started to focus on the importance by feature that our model applied and we could take profit of their utility in our model. So, once the model was trained, first we obtained the feature importances, which follows the Figure 3. With them we could iterate to validate which threshold is the most suitable and in our case it filters just three features: `meanUE\_UL`, `maxUE\_UL+DL` and `PBRUsageDL`. As it can be seen in Figure 4 those are the three features with more importance for our model, which maximize the discriminability between classes and remove the noise and complexity added by the other features. With this approach, we could reduce the error on validation and test set as will be explained in the section V. To sum up, finally without any transformation, just the pruning of the less important features (from 14 to 3) was the best way to ease the modeling of our classifier.

Afterwards, we thought about non-linear feature reduction so we used t-SNE, since we deemed there was not enough date to train an Autoencoder on a Neural Network.

III. Classification Methods

Due to the dataset nature, a linear classifier will not achieve an acceptable performance. A neural network did not work either due to the small dataset. A SVM with a Gaussian kernel was tested as well but, the results were not as good as the ones obtained by decision tree. Some results applying plain vanilla classifiers without any parameter tuning are shown in Table III. As it can be seen, the best performance is obtained with a plain vanilla decision tree. Therefore, we decided to dive deeply into that approach.

Following the decision tree approach, in order to get higher performance, the best option is to ensemble several decision

Classifier	Train Err.	Valdiation Err.	Accuracy
Decision Tree	0	0.03	0.96
SVM (rbf)	0.26	0.26	0.73
MLP	0.27	0.27	0.73

TABLE I
Plain vanilla classifiers performance

trees. One of the widely known ensembled methods are gradient boosted decision trees. In a summarized way, boosting takes an iterative approach for training models. It trains models in succession, with each new model being trained to correct the errors made by the previous ones [3]. A widely used implementation of the training described before is XGBoost. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable [4]. XGBoost stands for ”Extreme Gradient Boosting”, where the term originates from [5].

The best set up of hyperparameters for the XGBoost model that we found, as will be explained in detail in section V, were the following:

```
{'colsample_bytree': 0.0534,
 'eta': 0.2029,
 'gamma': 0.8872,
 'learning_rate': 0.3849,
 'max_depth': 26,
 'min_child_weight': 1,
 'estimators': 565,
 'subsample': 0.9781}
```

Some insights about the most relevant hyperparameters.

- `max_depth`, `min_child_weight` and `gamma` directly control model complexity.
- `subsample` and `colsample_bytree` add randomness to the training in order to make it robust to noise [6].

IV. Evaluation Metrics

A. Confusion Matrix

There are several evaluation metrics used in order to solve the classification problem. To know information on true/false positives and true/false negatives, is useful to have the confusion matrix

```
validation error: 0.002439
validation confusion matrix:
[[5340    4]
 [  14 2023]]
```

B. Classification Report

Furthermore, scickit learn provides a wide set of useful tools when it comes to validation. The classification report gives us some insights on how is performing our model, i.e as exposed in Section VI.

Finally, when the validation report is considered good enough, i.e better score than the previous training, then is updated to kaggle in order to evaluate the performance within the test dataset.

V. Experiments

A. Data Split

Since the number of submissions into the competition have a limit, is useful to split the whole dataset within train set and validation set. The train set is used to train the model whereas the validation set is used to evaluate the performance of the model with unseen data. Scikit learn provides a routine specifically for this:

```
train_test_split(X,
                 y,
                 train_size=0.8,
                 random_state=1,
                 stratify = y)
```

This routine is also useful to help with unbalanced data problem. With the `stratify = y` option it is ensured that relative class frequencies are approximately preserved in each train and validation set.

B. Hyperparameter Tuning

In order to control implementation aspects of the model, hyperparameter tuning is really important in order to achieve a good performance solving the classification problem. A starting point for hyperparameter tuning was to start from a well known working set of parameters [7]. Once we had an initial set of potential hyperparameters, we delimited the range of values through a different distributions for each parameter and then run a RandomizedSearch. We started using distributions within a wider interval of values for all of them. By doing several iterations of the RandomizedSearch, concretely a training of 500 iterations with cross-validation across 5 folds, we could start to check as many combinations as possible and find out which were the hyperparameters that most take part in our classification model; checking `clf.best_params_` in order to see the best hyperparameters for the whole training. Afterwards, we obtained the best set hyperparameters chosen by the RandomizedSearch, we iterated through the whole process some times only reducing (or even changing) the distribution interval of the hyperparameters and testing new ones.

Through the method described above, the best set of hyperparameters found are the ones introduced on section III.

With this simple approach and closing the range of values we quickly attain close to 99% score in train and validation set.

C. Solve the missprediction

After having already a decent score, we decide to evaluate which cases we were misspredicting. Observing the obtained correlation matrix of the validation set we could see that the main error where when deciding class 1, i.e. `Unusual`, when they are `Usual`. So we get the ball rolling with some experiments to enhance that outcomes.

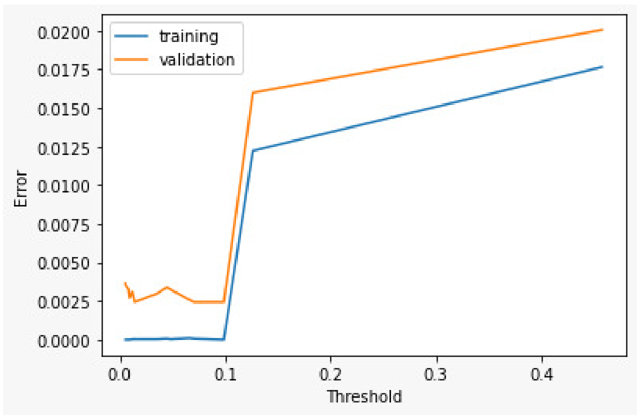

Fig. 3. Loss depending on threshold for feature reduction

1) Voting Classifier: The first that we tried was to build a more complex pipeline with a Voting Classifier. This module consist on a set of several models of free selection. So we tried to combine the already trained XGBoost with other classifiers, such as Decision Trees, KNNs, even more ensemble methods such as Adaboost, GradientBosting, RandomForest, and so on. But there was no luck with any of them, what is more, the accuracy of the final model got worse. Other test was to switch the voting strategy for the global class decision from `hard`, which uses the majority rule voting, to `soft`, which is based on the $argmax$ of the sums of the predicted probabilities, and neither improve the obtained scores. Any classifier that we attempted to combine could offer an improvement for the samples that the XGBoost missclassified, so we set the Voting Classifier aside.

2) F1-Score to train the XGBoost: Besides the Voting Classifier, we realized that we had not defined the score metric for the fit of the XGBoost, hence we evaluated which score metrics could be suitable for our challenge. Taking [8] as example, we could test several evaluation metrics and finally choose F1-score. F1-score is useful since we are dealing with imbalanced classes, precision and recall have high impact following an empiric evaluation, hence F-1 score is the more effective metric to evaluate this situation. So we repeated the same training with the F1-score selected as evaluation metric, improving by then accuracy by 0.2% . Recall improved too, which represents a big enhancement in that point when we were already over 99% of scoring.

D. Class imbalance

We understood that a big part of the problem came up from the imbalance. Moreover, with the F1-score we had found out that focusing on the imbalance there were possibilities of improvement and, in fact, we had a huge imbalance. 70% of the total dataset were `Usual` samples. Therefore, we performed a couple of tests to cover it.

1) Threshold tuning: First technique consists on tune the threshold. With the model built, we were deciding the classes from the estimated probability with a threshold of 0.5 and that could be wrong whether if we take into account such

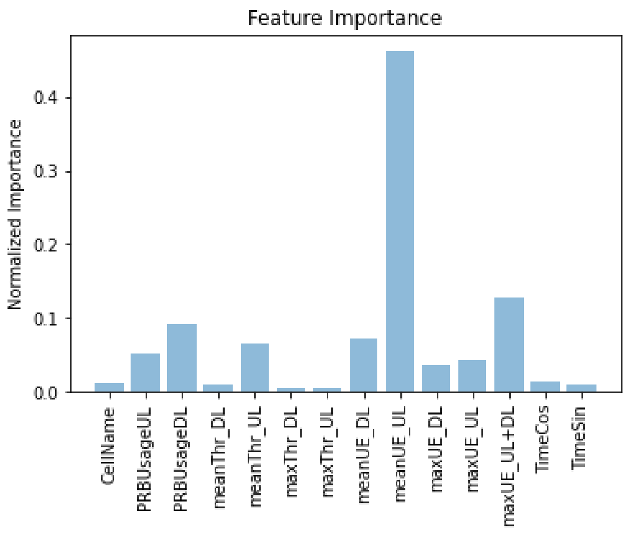

Fig. 4. Feature importances by the XGBoost

imbalance. To do so, we developed a code on the validation set to seek the best threshold. With this variation, we achieved to reduce the missclassified `Usual` samples, but there had appeared missclassification for the `Unusual` samples that previously did not exist. In brief, this did not improve the prediction.

2) Weighted penalties: As [9] suggests, we used the hyper-parameter `scale_pos_weight` to penalize differently the errors when training the XGBoost. So, we set this parameter to 7, since the `Usual` case represents 70% of the total samples and retrained with all accumulated improvements on the model. Both train and validation improved the score, obtaining a precision and recall of 99%, and moreover, this represented an improvement of 0.35% approximately on the test set, which was a big point.

E. Select features from model

The last experiment that led to a significant enhancement was to take advantage of the features selected by the model to perform a feature reduction. Taking into account that XGBoost, in essence, is a set of ensemble decision trees, we could evaluate its features importances, once the model is trained, as seen in Figure 4. Evaluating that data as explained in section II and following the idea suggested by [10], we get the optimal threshold to reduce the accuracy using just the three more relevant features. The `SelectFromModel` module let us to apply the thresholds to the already trained model and, once the best threshold is validated, as is plot on Figure 3, transform the datasets using just the features that go beyond the threshold. This final experiment let us reduce the recall from 44 errors in validation to 14. In the test set we achieved a score of 99.83%, which was our best performance in the challenge.

VI. Results

In Figure 5, as a result of all the previously conducted experiments, there is a summary of all the blocks that take part

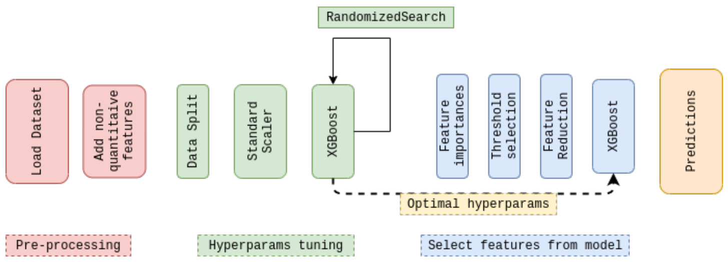

Fig. 5. Final pipeline breakdown

in order to solve the classification problem. Below is shown the best prediction obtained with the trained classifier with the optimal hyperparameters.

```
TRAINING SET
             precision    recall  f1-score

           0       1.00      1.00      1.00
           1       1.00      1.00      1.00

    accuracy                           1.00
   macro avg       1.00      1.00      1.00
weighted avg       1.00      1.00      1.00


VALIDATION SET
             precision    recall  f1-score

           0       1.00      1.00      1.00
           1       1.00      0.99      1.00

    accuracy                           1.00
   macro avg       1.00      1.00      1.00
weighted avg       1.00      1.00      1.00
```

VII. Conclusions

During this research, we have explored a vast amount of possible classifiers and techniques to deal with the binary classification problem exposed in this report. Among those all classifiers, XGBoost has resulted to be the proper one to perform this task, as other related resources suggested ([4], [5], [11]).

Far from the scope of this work, on a real production environment, this system is combined with lagged variables and treat as a time series challenge, as mentioned on [12]. However, the work conducted in this report serves as a proof of concept showing how powerful XGBoost is when comes to tackle problems of similar nature.

References

[1] S. Giménez and G. Castell, “Anomaly detection in 4g cellular networks. explore ml solutions for the detection of abnormal behaviour of enb (mlearn - matt 2020).” [Online]. Available: https://github.com/sergio-gimenez/anomaly-4G-detection/

[2] “Anomaly detection in 4g cellular networks.” [Online]. Available: https://www.kaggle.com/c/anomaly-detection-in-4g-cellular-networks

[3] G. Seif, “A beginner’s guide to xgboost.” [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7

[4] xgboost developers, “Xgboost documentation.” [Online]. Available: https://xgboost.readthedocs.io/en/latest/

[5] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. [Online]. Available: http://www.jstor.org/stable/2699986

[6] X. Parameters, “A beginner’s guide to xgboost.” [Online]. Available: https://xgboost.readthedocs.io/en/latest/parameter.html

[7] S. O. M. Power), “Python hyperparameter optimization for xgbclassifier using randomizedsearchcv.” [Online]. Available: https://stackoverflow.com/questions/43927725/python-hyperparameter-optimization-for-xgbclassifier-using-randomizedsearchcv

[8] T. proper way to use Machine Learning metrics, “Félix revert.” [Online]. Available: https://towardsdatascience.com/the-proper-way-to-use-machine-learning-metrics-4803247a2578

[9] XGBoost and I. C. P. H. Cancellations, “Michael grogan.” [Online]. Available: https://towardsdatascience.com/boosting-techniques-in-python-predicting-hotel-cancellations-62b7a76ffa6c

[10] J. Hirko, “Intro to classification and feature selection with xgboost.” [Online]. Available: https://www.aitimejournal.com/@jonathan.hirko/intro-to-classification-and-feature-selection-with-xgboost

[11] xgboost developers, “Xgboost model documentation.” [Online]. Available: https://xgboost.readthedocs.io/en/latest/tutorials/model.html

[12] J. Brownlee, “How to use xgboost for time series forecasting.” [Online]. Available: https://machinelearningmastery.com/xgboost-for-time-series-forecasting/