Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

XGBoost and LGBM for Porto Seguro's Kaggle challenge: A comparison

Semester Project

Kamil Belkhayat Abou Omar

bkamil@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zurich

Supervisors:
Gino Bruner, Yuyi Wang
Prof. Dr. Roger Wattenhofer

January 20, 2018

Acknowledgements

This research was supported by Prof Dr. Roger Wattenhofer. I would like to show my gratitude to Gino Bruner and Yuyi Wang for sharing their pearls of wisdom during the course of this research and for their comments on an earlier version of the manuscript, although any errors are my own and should not tarnish the reputations of these esteemed persons.

Abstract

Porto Seguro, one of Brazil's largest auto and homeowner insurance companies, challenged participants of the competition to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year given a strongly imbalanced and anonymized training set. This report assesses the performance of two state of the art boosting methods, namely Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machine (LGBM), comparing their performance against a neural network (Multi-Layer-Perceptron). We show that for this specific case, boosting methods outperform neural networks.

Contents

Acknowledgements i

Abstract ii

1 Introduction 1

1.1 Motivation and Background 1

2 Boosting Methods 3

2.1 Adaptive Boosting (Adaboost) 3

2.2 Gradient Boosting Models 4

2.2.1 eXtreme Gradient Boosting (XGBoost) 5

2.2.2 Light Gradient Boosting Machine (LGBM) 7

3 Experimental Results 8

3.1 Dataset 8

3.2 Preprocessing 10

3.2.1 One Hot Encoding 10

3.2.2 Recursive Feature Elimination using gradient boosting . . 10

3.3 Normalized Gini 10

3.4 Baseline: Multi Layer Perceptron (MLP) 11

3.5 Results and Discussions 12

4 Summary 14

5 Bibliography 15

Chapter 1

Introduction

1.1 Motivation and Background

Boosting methods have been first proposed by Schapire and Freund's AdaBoost algorithm [1] as a derivative of ensemble methods based on the following observation: while it may be very hard to build a highly accurate classifier (strong learner), it is much easier to come up with many rough rules of thumb (weak learners) which we only expect to predict strictly better than a random guessing. The performance of these simple and biased classifier is then "boosted" by combining them together using a majority vote of the weak learner's prediction, weighted by their individual accuracy. It can be seen as a weighted ensemble method focusing on reducing bias by iteratively correcting the estimated classification function in order to improve accuracy: weights on misclassified training examples are increased, forcing the weak learners to focus on the hardest samples. New weak learners built on the residuals of the base weak learners are added sequentially to focus on the more difficult patterns. Shortly after AdaBoost was introduced, it was observed that the test error of the algorithm does not increase even after a large number of iterations [15] which was later explained by Schapire and al. [16]: the generalization performance is improved because Adaboost tends to increase margins of the training examples.

Friedman [2] proposed gradient boosting as a gradient descent method in function space capable of fitting generic non parametric predictive models based on the Schapire and Freund idea. It has been empirically demonstrated to be very accurate for classification and regressions tasks when applied to tree models [3] [4] [5]. While we witnessed these past years an infatuation around neural networks and deep learning, gradient boosting turns out to be one of the most used methods of winning competitions [13].

In this study, we assess two state of the art gradient boosting methods, namely eXtreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machine (LGBM) to predict the probability that a driver will initiate an auto insurance claim in the framework of the Kaggle Porto Seguro's Challenge. We compare them to an appropriate Multi-Layer Perceptron (MLP) to show their

efficiency.

Chapter 2

Boosting Methods

As explained earlier, the idea behind boosting is to augment weak classifiers accuracy using an ensemble averaging. Like Bagging, boosting is based on building a ensemble of models aggregated by averaging the estimations or voting. However, it differs in the way of constructing this ensemble which is in this case recursive : each model is an adaptive version of the previous one, giving more weight to misclassified samples in the next estimation. Intuitively, the model focuses hence on misclassified samples while the aggregation of models reduces the risk of overfitting. Many parameters can then be tuned in order to derive different version of boosting : The weighting , i.e the way of reinforcing importance of misclassified samples, the cost function which can be chosen more or less robust to unusual values etc...

We will first briefly describe the original boosting model, AdaBoost, in order to highlights the specificities of the gradient boosting tree models LGBM and XGBoost further.

2.1 Adaptive Boosting (Adaboost)

In AdaBoost, weights of each sample are first initialized to $1/n$ for the first model estimation and change at each estimation. The weight of a sample w_i remains unchanged if the sample is correctly classified, and grows proportionally to the lack of fitting otherwise. The final predictor has the following form:

$$H(x) = \Sigma_{m=1}^{M} c_m \delta_m(x_0) \tag{2.1}$$

where $\delta_1, ..., \delta_M$ the weak classifiers, in this case, decision trees.

Mathematically, it can be expressed as the following optimization algorithm:

$$(c_m, \gamma_m) = argmin_{c,\gamma} \Sigma_{i=1}^{n} L(y_i, \hat{f}_{m-1}(x_i) + c\delta(x_i; \gamma)) \tag{2.2}$$

with:

$$\hat{f}(x) = \Sigma_{m=1}^{M} c_m \delta(x; \gamma_m) \tag{2.3}$$

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + c_m\delta(x;\gamma_m) \tag{2.4}$$

c_m being a parameter, δ a weak classifier of x depending on parameter γ_m, L(.) a cost function. In the case of AdaBoost, $L(y, f(x)) = exp[-yf(x)]$.

$$(c_m, \gamma_m) = argmin_{c,\gamma}\Sigma_{i=1}^{n} exp[y_i(\hat{f}_{m-1}(x_i) + c\delta(x_i;\gamma))] \tag{2.5}$$

and hence

$$(c_m, \gamma_m) = argmin_{c,\gamma}\Sigma_{i=1}^{n} w_i^m exp[-cy_i\delta(x_i;\gamma)] \tag{2.6}$$

with $w_i^m = exp[-y_i\hat{f}_{m-1}(x_i)]$, independent of c and γ, playing the role of a weight that depend on the previous iteration. Hastie et al. (2001) [6] show that the solution is obtained by first searching for the optimal classifier and then optimizing parameter c_m.

$$\gamma_m = argmin_{\gamma}\Sigma_{i=1}^{n} \mathbb{1}[\delta_m(x_i;\gamma) \neq y_i], \tag{2.7}$$

$$c_m = \frac{1}{2} log \frac{1-\hat{\epsilon}}{\hat{\epsilon}} \tag{2.8}$$

The weights are updated using:

$$w_i^m = w_i^{m-1} exp[-c_m] \tag{2.9}$$

Algorithm 1 AdaBoost

$z = (x_1, y_1), ...(x_n, y_n)$
INITIALIZE $w = w_i = 1/n; i = 1, .., n$
for $m \in \{1, \ldots, M\}$ **do**
  Find classifier δ_m that minimizes classification error depending on difficulty of the samples
  Compute empirical error:$\hat{\epsilon} = \frac{\Sigma_{i=1}^{n} w_i \mathbb{1}[\delta_m(x_i) \neq y_i]}{\Sigma_{i=1}^{n} w_i}$
  Compute logit: $c_m = log((1-\hat{\epsilon})/\hat{\epsilon})$
  Compute new weights: $w_i = w_i.exp[c_m \mathbb{1}[\delta_m(x_i) \neq y_i]]$; $i = 1, ..., n$.
end for
Prediction $\hat{f}_M(x) = sign[\Sigma_{m=1}^{M} c_m\delta_m(x)]$
=0

2.2 Gradient Boosting Models

Based on Adaboost and more generally on adaptive approximation methods, Friedman (2002) [9] proposes to build a sequence of models such that at each iteration, each model added to the combination improves the overall solution. The main innovation here is that, in order to improve convergence, the optimal

solution is heading the direction of the gradient of the objective function. The previous adaptive model:

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + c_m\delta(x;\gamma_m) \tag{2.10}$$

is transformed as a gradient descent:

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + \gamma_m\Sigma_{i=1}^{n}\nabla_{f_{m-1}}L(y_i, f_{m-1}(x_i)) \tag{2.11}$$

Instead of looking for a better classifier like in AdaBoost, the problem is restrained to finding the best learning rate γ:

$$min_\gamma\Sigma_{i=1}^{n}[L(y_i, f_{m-1}(x_i) - \gamma\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)})] \tag{2.12}$$

Algorithm 2 Gradient Tree Boosting

INITIALIZE $\hat{f}_0 = argmin_\gamma\Sigma_{i=1}^{n}L(y_i,\gamma)$
for $m \in \{1,\ldots,M\}$ **do**
  Compute $r_{mi} = -[\frac{\partial L(y_i,f_{m-1}(x_i))}{\partial f_{m-1}(x_i)})]; i = 1, ..., m$
  Fit a decision tree δ_m to $(x_i, r_{mi})_{i=1,...,n}$
  Compute γ_m by solving: $min_\gamma\Sigma_{i=1}^{n}L(y_i, f_{m-1}(x_i) + \gamma\delta_m(x_i))$
  Update $\hat{f}_m(x) = \hat{f}_{m-1}(x) + \gamma_m\delta_m(x)$
end for
Prediction $\hat{f}_M(x)$ =0

The algorithm is here initialized by a constant, which mean a tree with a single leaf. For each node of the model and each sample of this node, we compute r_mi and the learning rate γ is optimized resulting in each update of the model.

2.2.1 eXtreme Gradient Boosting (XGBoost)

Chen and Guestrin (2016) [7] introduced the eXtreme Gradient Boosting (XGBoost). They introduce a regularization to the cost function, making it a 'regularized boosting' technique:

$$L(f) = \Sigma_{i=1}^{n}L(\hat{y}_i, y_i) + \Sigma_{m=1}^{M}\Omega(\delta_m) \tag{2.13}$$

with:

$$\Omega(\delta) = \alpha|\delta| + \frac{1}{2}\beta||w||^2 \tag{2.14}$$

where $|\delta|$ is the number of leaves of the classification tree δ and w the vector of values attributed to each leaf. the regularizer Ω penalizes the complexity of the model and can be interpreted as a combination of ridge regularization of coefficient β and Lasso regularization of coefficient α. When the regularization

parameter is set to zero, the cost function falls back to the traditional gradient tree boosting.

J. Friedman, T. Hastie, and R. Tibshirani (2000) [7] show a trick consisting in expressing the objective function as a second order Taylor expansion to quickly optimize the objective: For each leaves, it consists in adding first and second order derivative of the cost function to evaluate the regularized objective and maximize its decrease when searching for splits. This simplification also allows an efficient parallelization of tree construction, which was not possible in the original gradient boosting model.

$$L(f) \approx \Sigma_{i=1}^{n}[(L(\hat{y}_i, y_i)) + g_i\delta_t(x_i) + \frac{1}{2}h_i\delta_t^2(x_i))] + \Omega(\delta_t) \tag{2.15}$$

where $g_i = \partial_{\hat{y}} L(\hat{y}_i, y_i)$ and $h_i = \partial_{\hat{y}}^2 L(\hat{y}_i, y_i)$. By removing the constant term, we obtain the following approximation of the objective at step t:

$$\hat{L}(f) = \Sigma_{i=1}^{n}[g_i\delta_t(x_i) + \frac{1}{2}h_i\delta_t^2(x_i))] + \Omega(\delta_t) \tag{2.16}$$

By defining I_j as the instance set at leaf j and expanding Ω, we can rewrite equation (2.16) as [7]:

$$\hat{L}(f) = \Sigma_{j=1}^{T}[(\Sigma_{i\in I_j} g_i)w_j + \frac{1}{2}(\Sigma_{i\in I_j} h_i + \beta)w_j^2] + \alpha|\delta| \tag{2.17}$$

Finally, XGBoost does not use entropy or information gain for splits in a decision tree but rather use the following gain:

$$G_j = \Sigma_{i\in I_j} g_i \tag{2.18}$$

$$H_j = \Sigma_{i\in I_j} h_i \tag{2.19}$$

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L+\beta} + \frac{G_R^2}{H_R+\beta} - \frac{(G_R+G_L)^2}{H_R+H_L+\beta}] - \alpha \tag{2.20}$$

where the first term is the score of the left child, the second the score of the right child and the third the score if we do not split; α is the complexity cost if we add a new split. When implementing this we basically sort all of the attributes and linearly pass through all of the possible values of the split and compute the gain.

Missing values are handled in XGBoost by proposing at each split a default direction if a value is missing. The gradient is computed on the available values only.

2.2.2 Light Gradient Boosting Machine (LGBM)

LGBM is a new gradient boosting library implemented by Microsoft in April 2017 [9]. The aim was to make gradient boosting on decision trees faster. The idea is that instead of checking all of the splits when creating new leaves, we only check some of them: We first sort all of the attributes and bucket the observation by creating discrete bins. When we want to split a leaf in the tree, instead of iterating over all of the leaves, we simply iterate over all of the buckets. This implementation is called *histogram implementation* by its authors. The trees are grown depth first (or leaf wise) keeping the presorted state instead of level wise like other gradient boosting methods. The algorithm chooses the leaf with maximum delta loss to grow and does not to grow the whole level.

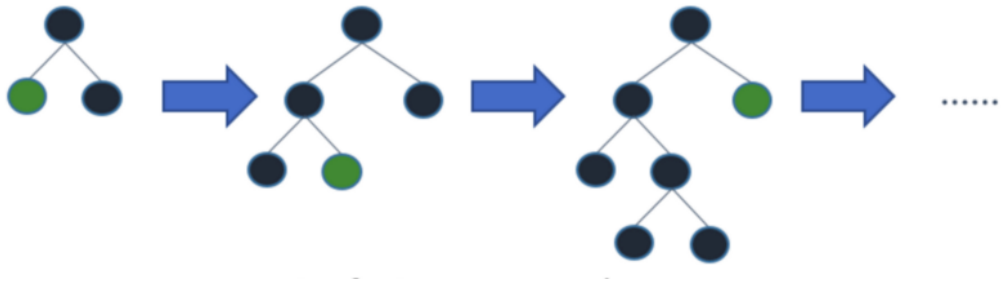

Figure 2.1: Leaf wise growth

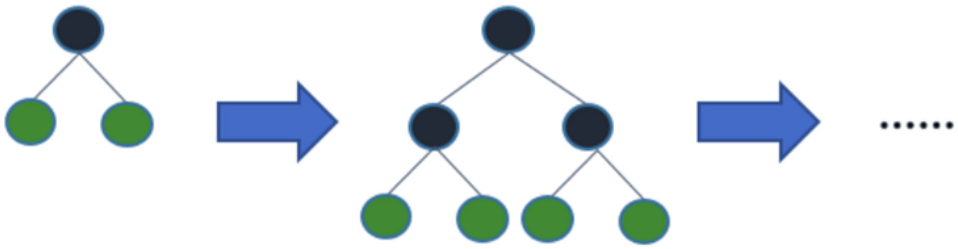

Figure 2.2: Level wise growth

.

Chapter 3

Experimental Results

Porto Seguro, one of Brazil's largest auto and homeowner insurance companies, observed that inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones. They challenged competitors to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year. A more accurate prediction will allow them to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers [6].

3.1 Dataset

The training Set contains 595212 observations and 57 features from which 17 binary variables, 30 categorical variables (14 nominal, 16 ordinal) and 10 float variables. Features are anonymized and renamed according to their data type. The target is binary and the dataset is strongly imbalanced: 3.65% is 1, the rest is 0. Missing values are represented by the value "-1".

We noticed that the distribution of the train and test set are almost identical which allowed us to fully trust our cross validation score. This observation was made by reducing both dataset to a small number of dimension and compare both scatter plot. The PCA scatter plot are shown in the Figure 3.1, where the principal axis 1 explains 9% of the variance and the principal axis 2 7.5%.

. We also observed significant overlap between positive and negative samples in low dimensional space suggesting that no small sample of features is easily able to predict future claim making. We show this by plotting positive and negative samples of the training set in a two-dimensional feature space for few pairs in Figure 3.2

.

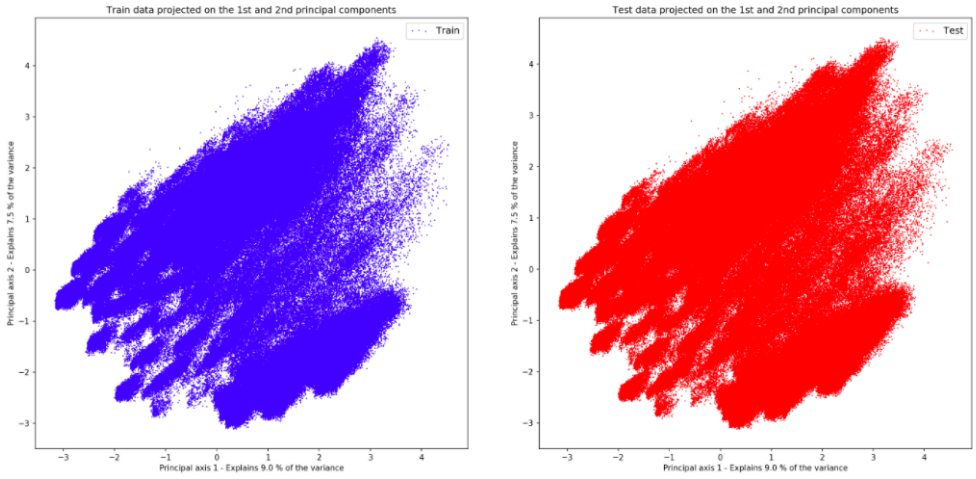

Figure 3.1: Train (blue) and test (red) data projected on the 1st and 2nd principal components

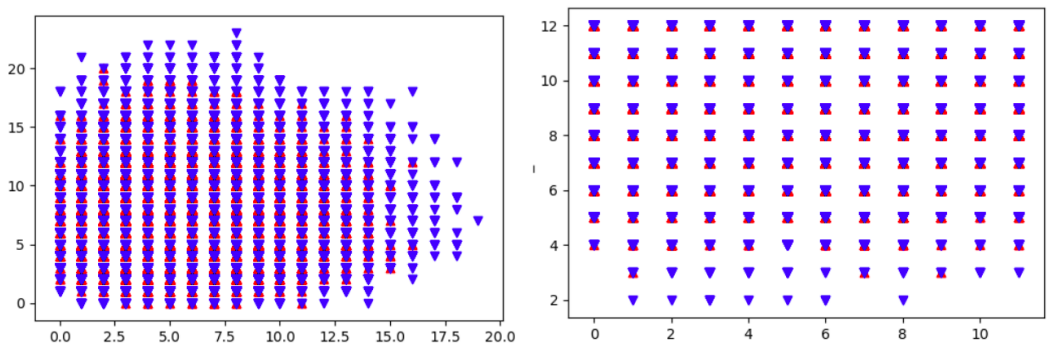

Figure 3.2: Distribution of positive (blue) and negative (red) example for two pairs of features

3.2 Preprocessing

3.2.1 One Hot Encoding

A one-hot encoding consists of representing states using for each a value whose binary representation has only a single digit 1. A one-hot encoding function can be defined as the function that takes a z vector as input and redefines the largest value of z to 1 and all other values of z to 0.

For example, in a one-hot encoding, for three possible states, the binary values 001, 010, and 100 can be used.

When using boosting trees, it is important to use a one hot encoding in order to reduce the noise while getting the splits for most useful categories.

3.2.2 Recursive Feature Elimination using gradient boosting

The main idea here is to use LGBM or XGBoost $feature\ importance$ method to reduce the dimensionality of the training data not by selecting the important ones but rather by eliminating the unimportant ones. The underlying idea is that if the gradient boosting splitting algorithm does not select some feature to split, it means that this feature does not add value to the boosting trees model. We hence first run LGBM with a small depth and remove feature with 0-importance (Figure 3.3).

The next step to make the data more robust is to remove noisy and correlated features. We used a simplified version of the Boruta algorithm available in LGBM library which is the $shuffling$ method [10] allowing to randomize selection of feature. Finally, we remove features with 0-importance in the new randomized dataset.

Also, we removed all the features starting with "calc" as they seemed to be randomly generated.

.

3.3 Normalized Gini

The Normalized Gini (NG) score is a very common metric for decision tree and can be expressed as a function of AUC:

$$Gini = 2 * AUC - 1 \tag{3.1}$$

The main advantage of this metric is that it sets the performance of a random classifier to 0 which is useful here as our data set is strongly imbalanced so any random predictions with a majority of 0 would seem to have a good performance

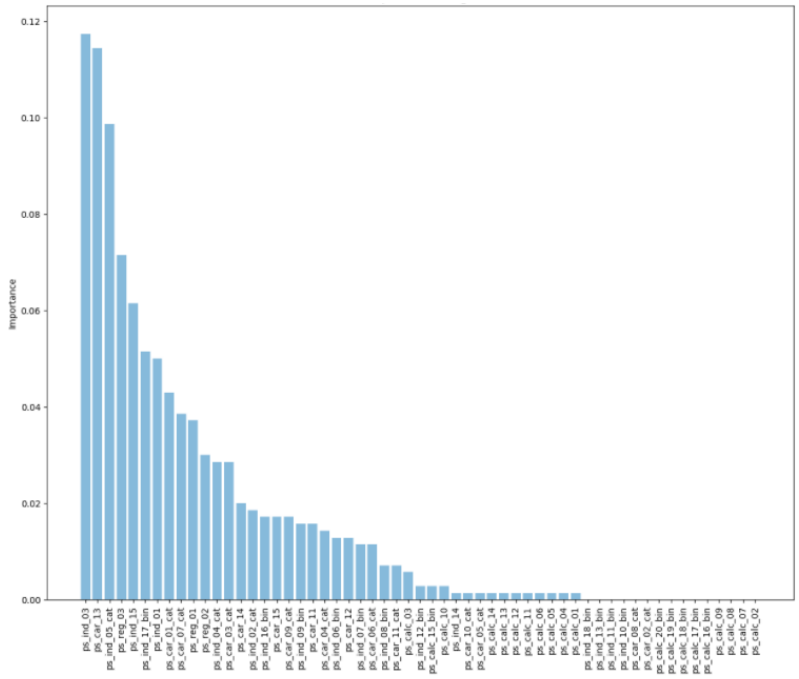

Figure 3.3: Feature Importance using LGBM

with another metric. The normalization improves the other end of the scale and ensure that a perfect classifier has a score of 1 rather than a maximum achievable AUC which is less than 1.

3.4 Baseline: Multi Layer Perceptron (MLP)

We designed a multi layer perceptron as a benchmark to our boosting trees libraries. We observed that among our 57 features, we have 8 types: *psind*, *psindcat*, *psindbin*, *psreg*, *pscarcat*, *pscar*, *pscalc* and *pscalcbin*. The only pre-processing done for the MLP is normalizing the features and the removal of both *pscalc* and *pscalcbin* as explained earlier. We imputed the missing values using the most frequent value of the feature. For the feature selection, we let the neural network learn the more suitable representation of the data. We hence designed a neural network whose input is divided in three parts 'reg', 'car' and 'ind'; each part is subdivided depending to its type: categorical or numerical. Because categorical features are fed into the network using one-hot encoding their dimensionality is very high and it's necessary to use a different embedding to avoid the curse of dimensionality. For each different input, the network learns an encoding which will then be combined to produce the prediction. We hence have 6 inputs, three layers in parallel, allowing to do a better fine tuning of the

size of the layers and assign to each type of feature the importance that it has for the prediction. After a grid search for parameter tuning, we came up with a tanh activation function on the embedding with L1 and L2 regularization to fight the curse of dimensionality , ReLu on the classifier with dropout, and sigmoid for the output layer.

The Figure 3.4 explains the architecture of the MLP, the rounded boxes are inputs, squared ones are nodes: .

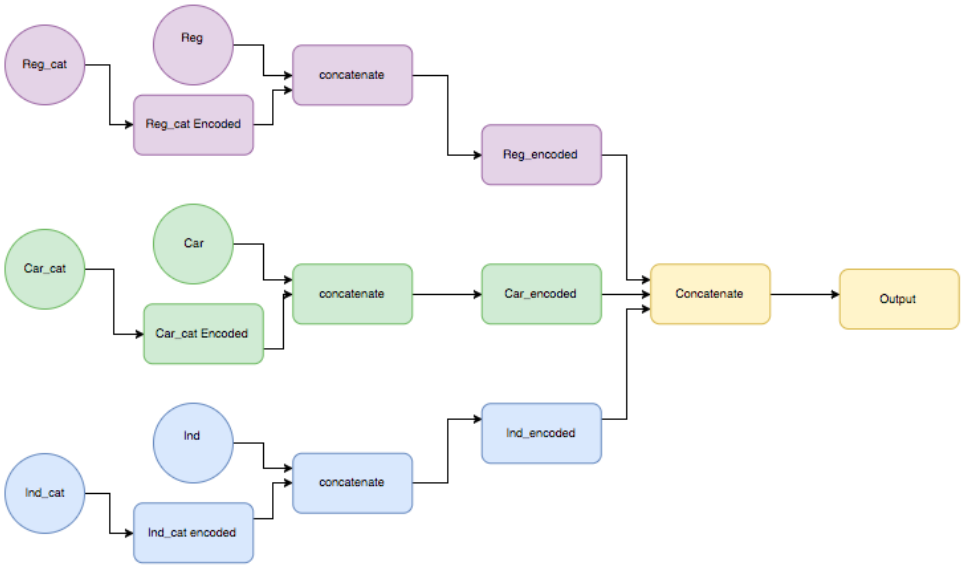

Figure 3.4: Neural Network Architecture

3.5 Results and Discussions

Below is a table summarizing the performance of our models according to Kaggle public and private leader board where the public leader board is computed based on a fraction of the test set while the private leader board is computed on the whole test set (so that if you overfit the test set you can do very well in the public leader board and very bad in the private one).

We have conducted numerous experiments locally, on ETH's CPU and GPU. All the experiments employ the preprocessing stage described in Sections 3.2.1 and 3.2.2. Table 3.1 showcases the main results of our experiments. The best performing model was XGBboost with both one hot encoding and gradient boosting feature selection. We can see that the gradient boosting methods individually outperform the MLP and Adaboost. It is probably due to the fact that

Model	NG Public Score	NG Private Score
MLP	0.2712	0.2805
AdaBoost	0.2231	0.2310
LGBM + OHE	0.2791	0.2810
LGBM + OHE + Feature Selection	0.2825	0.2873
XGB + OHE	0.2807	0.2867
XGB + OHE + Feature Selection	0.2858	0.29105

Table 3.1: Accuracy comparison of GB models and MLP

both LGBM and XGBoost have sparsity penalties while Adaboost excessively increases weight for noisy observations without regularization. Both LGBM and XGBoost have been optimized using Bayesian grid search for hyperparamater tuning [13] [14]. A more complex architecture of neural network could possibly result in a better normalized Gini score. .

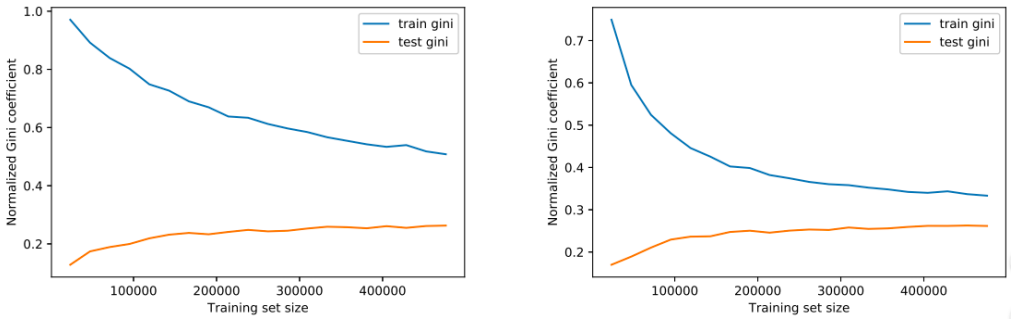

Figure 3.5: XGBoost and LGBM Learning Curves

The general boosting trees performance trends showcased in the literature [7] have been showcased in our experiments as well: tree based models solves tabular data very well. However, our data set was not very large and usually only deep learning methods are able to absorb huge amounts of training data without saturating in performance.

Chapter 4

Summary

This report compared the use of XGBoost and LGBM in the framework of Kaggle Porto's Seguro Challenge. The experimental results show that individually, gradient boosted tree models outperformed a simple Multi-Layer-Perceptron. Data preprocessing also played an important role as results showed that the feature selection procedure used improved XGboost performance over a simple one hot encoding. Many architecture could further enhance accuracy. Feature engineering has been shown to improve accuracy: blending several neural networks, Xgboost, and LGBM models would have probably resulted in a better accuracy as seen from the winning solution which was a linear stacking of 1 LGBM and 5 Neural Networks. Also, new boosting libraries as CatBoost [11] and InfiniteBoost [12] could have been tested as well and used in a ensemble averaging.

Chapter 5

Bibliography

[1] Y. FREUND and R.E. SCHAPIRE (1997). "A decision-theoretic generalization of online learning and an application to boosting," Journal of Computer and System Sciences, 55(1):119-139.

[2] J.H. FRIEDMA (2001). "Greedy Function Approximation: A Gradient Boosting Machine," Annals of Statistics 29(5):1189-1232.

[3] P. BÜHLMANN, B. YU, (2003). Boosting with the L2 loss: Regression and classification. J. Amer. Statist. Assoc. 98 324–339.

[4] G. RIDGEWAY. (1999). The state of boosting. Comput. Sci. Statistics 31 172–181.

[5] D.F MCCAFFREY, G. RIDGEWAY, and A. R. G. MORRAL (2004). Propensity score estimation with boosted regression for evaluating causal effects in observational studies. Psychological Methods 9 403–425.

[6] T. HASTIE, R. TIBSHIRANI and J FRIEDMAN (2009), The elements of statistical learning : data mining, inference, and prediction.

[7] T. CHEN, C GUESTRIN (2016), XGBoost : A Scalable Tree Boosting System, Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p. 785–794.

[8] J. FRIEDMAN, T. HASTIE, and R. TIBSHIRANI (2000), Additive logistic regression: a statistical view of boosting. Annals of Statistics, 28(2):337–407.

[9] H. SHI (2007), Best-first decision tree learning.

[10] M. B. KURSA, W. R. RUDNICKI (2010), Feature Selection with the Boruta Package.

[11] A. V. DOROGUSH, V. ERSHOV, A. GULIN (2017), CatBoost: gradient boosting with categorical features support.

[12] A. ROGOZHNIKOV (2017), InfiniteBoost: building infinite ensembles with gradient descent.

[13] J. SNOEK, H. LAROCHELLE (2012), Practical Bayesian Optimization

of Machine Learning Algorithms.

[14] E. BROCHU, V M. CORA and N. DE FREITAS (2010), A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.

[15] H. DRUCKER and C. CORTES . (1996). Boosting decision trees.

[16] R. E. SCHAPIRE, Y. FREUND, P. BARTLETT, and W. S. LEE (1998). Boosting the margin: A new explanation for the effectiveness of voting methods.