Comparison between XGBoost, LightGBM and Catboost using Bayesian Optimization for Parameter Tuning on Imbalanced Dataset

***Abstract*—The aim of this paper is to compare and evaluate the optimized performance of XGBoost, LightGBM and Catboost algorithms using Hyperopt, Optuna and Random Search for parameter tuning on a highly imbalanced dataset i.e. Credit Card default obtained from Kaggle.**

***Index Terms*—Gradient Boosting, LightGBM, Catboost, XGBoost, Hyperopt, Optuna, credit card fraud.**

I. Introduction

II. Related Work

The experiment aims to statistically evaluate the performance of XGBoost and its dervative algorithms, LightGBM and CatBoost, when they are optimized with Hyperopt and Optuna.

A. Boosting Tree Methods

Boosting builds models from individual weak learners in an iterative way. These models are not built on completely random subsets of data and features, but sequentially by putting more weight on instances with wrong predictions and high errors. Gradient Boosting is an ensemble of Decision Trees. The gradient is the partial derivative of the loss function, so it describes the steepness of the error function. The gradient directs towards the direction in which to change the model parameters to reduce the error in the next round of training by "descending the gradient". The predictions of multiple models are combined to optimize boosted model predictions[1]. The 3 boosted tree algorithms were part of the experiments, namely, XGBoost, Catboost and LightGBM.

1) XGBoost: XGBoost is an optimized distributed gradient boosting library which implements Gradient Boosting framework. It is an ensemble of a set of classification and regression trees (CART)[2]. XGBoost introduced regularized objective function (Training Loss + Regularization).
The training loss measures how predictive the model is on training data. Regularization term penalizes the complexity of the algorithm to avoid over-fitting[3]. XGBoost incorporates advanced regularization (L1 and L2) to improve model generalization. For learning purpose additive strategy is used where gradients are added to the running training process by fitting the next tree also to these values. In order to optimize the objective function, it computes the second-order gradients by taking Taylor expansion of the loss function up to the second order, which provides more information about the direction of gradients and how to get to the minimum of the loss function. While regular gradient boosting uses the loss function of the base model (e.g. decision tree) for minimizing the error of the overall model, XGBoost uses the 2nd order derivative as an approximation.
XGBoost uses Shrinkage technique [4] and column subsampling[5] to reduce over-fitting. The complexity of the model can be controlled by controlling max_depth, min_child_weight and gamma parameters. Over-fitting can also be avoided by adding randomness i.e. including subsample and colsample_bytree parameters. We can also reduce stepsize eta and increase num_round to control overfitting. For faster computation, we can set tree_method parameter to hist or gpu_hist. To handle imbalance data, we can balance the positive and negative weights via scale_pos_weight parameter or we can set parameter max_delta_step to a finite number (say 1) to help convergence.

2) LightGBM: LightGBM was developed with aim of making training faster. The conventional GBDT implementation needs to scan all the data points to estimate the information gain for splitting, thus increasing the complexity with the increase in number of features and instances. To overcome this challenge the developers introduced two techniques to sample the data:

- Gradient Based One-Side Sampling (GOSS): It kept instances with large gradients and those instances with smaller gradients are randomly dropped or sampled down.
- Exclusive Feature Bundling (EFB): This algorithm tackles the sparse feature space of mutually exclusive features (like one hot encoded representation) by bundling them together to reduce the number of features[6]

The main improvements that LightGBM has over XGBoost are:

- Histogram based algorithm: Each continuous feature is bucketed into discrete bins. The number of bins thus created is much smaller to iterate over as compared to number of points. Unlike other histogram based algorithms, LightGBM rely on EFB for optimization of sparse space.
- The trees are grown with depth first approach: Keeping the presorted state, LightGBM chooses the leaf with maximum delta loss to grow and does not have to grow the whole level as shown in figure 1.

3) CatBoost: Unlike XGBoost and LightGBM, CatBoost samples a new permutation of the dataset at each step of boosting to prevent prediction shift overfitting[7]. CatBoost maintains a set of models differing by permutations used for their training. Then, for calculating the residual of a particular permutation, CatBoost uses a model trained without that particular permutation. And it repeats the same methodology for each model, so each model is learned using only the 1_{st} i examples in the permutation. At each step, in order to obtain

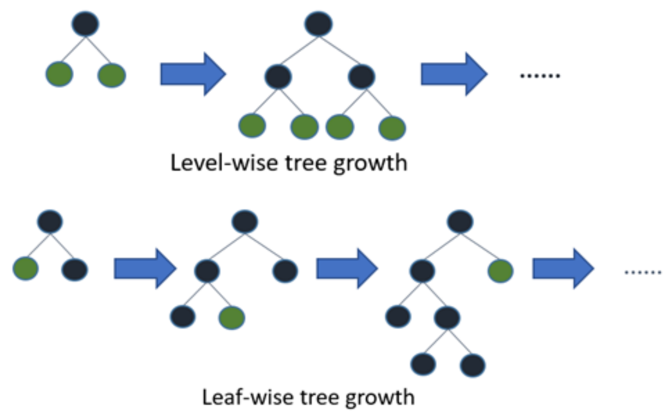

Fig. 1. XGBoost and other tree learning algorithms grow by level while LightGBM grows by leaf-wise

the residual for j_{th} sample, CatBoost uses M_{j-1} models. This ensures that the obtained residuals remain unshifted[7], when the current model is exposed to new training examples. This method is called as *ordered boosting*. The approach is the main differentiator between Catboost from its counterparts such as XGboost and LightGBM.

B. Optimization Methods

The experiments have taken 2 approaches for optimizing the hyperparameters. One approach is the commonly used random search. The other approach is using Bayesian optimization, more specifically Hyperopt and Optuna libraries which are classified as Sequential model-based optimizers(SMBO).

SMBO is a Bayesian Optimization (BO) method which has been shown to be effective and sample efficient in the case of black-box expensive objective functions. Bayesian Optimization is currently the standard approach in the machine learning community for solving automatic algorithm configuration, hyperparameter tuning[8][9]. In Sequential model-based optimization, the selection of new design points (new parameter configurations to be evaluated) can depend on the response values observed at previous design points.

The Bayesian optimization framework has two key components. The first ingredient is a probabilistic surrogate model—typically a Gaussian Process (GP) — which consists of a prior distribution that captures our beliefs about the behavior of the unknown objective function and an observation model that describes the data generation mechanism. The second ingredient is a loss function that describes how optimal a sequence of queries are; in practice, these loss functions often take the form of regret, either simple or cumulative. Ideally, the expected loss is then minimized to select an optimal sequence of queries. After observing the output of each query of the objective, the prior is updated to produce a more informative posterior distribution over the space of objective functions[10].

1) Hyperopt: Hyperopt is a Python library for SMBO and is a tool for hyperparameter optimization of machine learning algorithms. Its developers claim that it can increase efficiency of hyperparameter tuning for difficult search settings (i.e. many hyperparameters and a small budget for function) by defining where the function is defined and where you think the best values are. The reuierments for implementation of Hyperopt mthod are: the objective function (to be minimized), search space (configuration space), a trials database and the search algorithm[11].

2) Optuna: Optuna is another SMBO method for hyperparameter optimization of machine learning algorithms. It features a very efficient pruning strategy, because in practice, both parameter searching strategy and performance estimation strategy are important for high-performance optimization[12],[13],[14]. Unlike other hyperparameter-optimizers (define-and-run APIs), new generation hyperparameter optimizers such as Optuna apply a define-by-run API[12]. There are generally two types of sampling method:

- Relational sampling that exploits the correlations among the parameters
- Independent sampling that samples each parameter independently.

Optuna can handle various independent sampling methods including Tree Parzen Estimtors (TPE) as well as relational sampling methods like or Gaussian Process-Baysian Optimization(GP-BO)[15],[10]. Instead of using old-techniques of early-stopping, Optuna deploys Asynchronous Successive Halving(ASHA). In this technique each worker is allowed to asynchronously execute aggressive early stopping based on provisional ranking of trials. Therefore, the parallel computation can process multiple trials simultaneously without delay, and one worker does not have to wait for the results from other workers at each round of pruning. Such advanced pruning can significantly accelerate the optimization speed, by conducting more trials and exploring a broader space[12].

3) Random Search: Random Search evaluates a given number of random combinations of hyperparameters by random selection of values for each hyperparameter at every iteration[16]. Random search has been used as a baseline for keeping the advantages of implementation simplicity and reproducibility.

C. Statistical Comparison of Algorithms

Statistical significance tests are designed to compare the performance of Machine Learning (ML) models and quantify the likelihood of the samples of performance score being observed, given the assumption that they were drawn from the same distribution. If this assumption, i.e. null hypothesis, is rejected, it suggests that the difference in performance scores is statistically significant. With the limited amount of data in the test set, we cannot rely just on the evaluation parameter like accuracy to understand the nature of different models. A statistical test is a technique that relies on the probability distribution, for reaching the conclusion concerning the reasonableness of the hypothesis. Hypothetical tests related to spotting differences are classified as parametric and nonparametric tests. Parametric tests make the assumption that the sample comes from a distribution that belongs to a known family, usually the Gaussian family, and is described with a number of parameters of that distribution[17]. Nonparametric statistical tests rely on no or few assumptions about the shape or parameters of the population distribution from which the sample was drawn[18].

1) T-test: The Parametric test provides the generalisations about the mean of the parent population. The T-test assesses whether the means of two groups are statistically different from each other[19]. There are certain assumptions to be checked for this test. For example:
i. If the two samples are independent of one another
ii. If the two populations have equal variance or spread
iii. If the two populations are normally distributed
The input required for this test is the group of observations of the different runs of the pair algorithms that will be compared for a single problem. The null hypothesis in this test is that there is no difference between the performances of two applied ML models. A p-value smaller than the considered significance level rejects the null hypothesis in favor of the alternative hypothesis, which assumes the ML models perform differently.

2) Wilcox Rank Sum: Wilcoxon Rank Sum Test does not assume known distributions, it does not deal with parameters (mean or variance), and therefore it is a non-parametric version of the two-sample t-test[20]. It is based solely on the order in which the observations from the two samples fall.
The null hypothesis to be tested is:
H_0: the observations of the two samples are from the same distribution and the alternative hypothesis is
H_1: the observations of the two samples are from two distributions that have the same shape, but there is a shift in location.
The two samples are combined and rank ordered together. The strategy is to determine if the values from the two samples are randomly mixed in the rank ordering or if they are clustered at opposite ends when combined. A random rank order would mean that the two samples are not different, while a cluster of one sample values would indicate a difference between them[21]. The Wilcoxon test assumes continuous differences, therefore they should not be rounded to, say, one or two decimals since this would decrease the power of the test due to a high number of ties.
This test is often performed as a two-sided test and, thus, the research hypothesis indicates that the populations are not equal as opposed to specifying directionality. A one-sided research hypothesis is used if interest lies in detecting a positive or negative shift in one population as compared to the other.

3) Kappa Test: Cohen's Kappa is a quantitative measure of reliability for two raters that are rating the same thing, corrected for how often that the raters may agree by chance. It is a very good measure that can handle very well both multi-class and imbalanced class problems.
Cohen's kappa can be estimated as follow:

$$k = \frac{P_o - P_e}{1 - P_e} = 1 - \frac{1 - P_o}{1 - P_e}$$

Where P_o is the observed agreement and P_e is the expected agreement. It basically tells how much better the classifier is performing over the performance of a classifier that simply guesses at random according to the frequency of each class. Cohen's kappa is always less than or equal to one. Values of zero or less, indicate that the classifier is not useful. There is no standardized way to interpret its values. Landis and Koch (1977) provide a way to characterize values. According to their scheme, a value < 0 is indicating no agreement, 0–0.20 as slight, 0.21–0.40 as fair, 0.41–0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1 as almost perfect agreement[22].

4) Comparing Roc: Receiver Operating Characteristics (ROC) is a probability curve, and AUC i.e. Area Under Curve represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes[23]. The ROC curve is plotted with True Positive Rate (TPR) against the False Positive Rate (FPR). AUC helps us in assessing the performance of the classifier over its entire operating range. A single threshold can be selected to compare the classifiers' performance at that point, or the overall performance can be compared by considering the AUC.

D. Feature Importance

In machine learning, the interpretability of models is as important as their prediction accuracy. In recent years, large amount of model agnostic methods to improve the transparency, trustability, and interpretability of machine learning models have been developed. These are all grouped under the feature importance umbrella, but each pick certain characteristics of feature-outcome interaction. Recursive feature elimination (RFE) methods are a greedy search where the algorithm is trained on an iteratively altered subset of features. Recursive feature elimination does not scale to high dimensional datasets and is susceptible to distribution shift [24].Remove And Retrain (ROAR) is a modified version of RFE and allows for better control. On every iteration, a new model is re-trained from random initialization on the new dataset and evaluated on the new test data. Since re-training can result in slightly different models, it is essential to repeat the training process multiple times to ensure that the variance in accuracy is low [25]. Without re-training, it is unclear whether the degradation in model performance comes from the distribution shift or because the features that were removed are truly informative [26],[27]. Therefore, retraining better controls for artefacts introduced by the data modification.
However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, such as ensemble or deep learning models, creating a tension between accuracy and interpretability. In response, various methods have recently been proposed to help users interpret the predictions of complex models, but it is often unclear how these methods are related and when one method is preferable over another. To address this problem,

we present a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations)[28] .

III. Comparing the ML Algorithms

There are published papers which present the comparison for GBDT algorithms in terms of CPU runtime and accuracy. Some papers highlight faster CPU runtime for LightGBM [29] or greater reduction in training time by XGBoost or better performance by CatBoost with categorical features [30]. However the common critique is that ML parameters are not comparable with each other and to make a fair comparison we need to solve hyper parameter optimization problem i.e. find an optimal set of parameters that will allow achieving this fixed quality in the minimum amount of time [31]. Considering this, the paper examines the algorithm performance with two optimization techniques – Hyperopt and Optuna and take Random Search performance as a baseline. The approach taken in this paper is to run the experiments multiple time to address the following questions:
i. How do the time to hyperparameter optimization and best loss statistically compare for each algorithm and optimization
ii. How statistically comparable is the ROC curve and Kappa score for each scenario
iii. How robust is each model to generate feature importance
The multiple experiments with random seed values would provide the comparison from sample distribution of model performance. Statistical benchmarking is becoming increasingly important. Statistical evaluation of experimental results has been considered an essential part of validation of new machine learning methods for quite some time. Selecting the appropriate test is crucial. Ben David in his paper [32] used Weighted Kappa to effectively deal with cost-sensitive classification. The approach used in this paper compensates for classifications that may be due to chance. Provost et al. point out in their paper [33] that comparison of algorithms based on accuracy is unsatisfactory. Classification accuracy implies equal misclassification costs (i.e. false positive and false negative errors). They presented a methodology that if there is no dominating classifier and cost and class distribution information is not available, conclusions can be drawn based on the superiority for specific regions of ROC space. For example, if few false positive errors can be tolerated, algorithm that is superior at the "far left" edge of ROC space can be considered better than other algorithms. Wilcoxon-signed rank test was used for performing pair wise comparison for the results obtained by the Genetic Based Machine Learning (GBML) algorithms for classification[34].

IV. About Dataset

The dataset used in this paper has been taken from Kaggle, it was collected and analysed during a research collaboration of Worldline and the Machine Learning Group ($http://mlg.ulb.ac.be$) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. The dataset contains transactions made through credit cards in the duration of two days in the month of September, 2013 by European cardholders. For the purpose of confidentiality, the details of all cardholders i.e. most of the features have been transformed via a principal component analysis (PCA) transform. It contains only numerical input variables as a result of PCA transformations. In total, there are 28 transformed features (V1, V2.....V28). Remaining features, as shown in Table I, are non-PCA applied features.

TABLE I
Description of non-PCA features

S.No	Feature	Description
1	Time	Number of seconds that elapsed between the current transaction and the first transaction
2	Amount	Transaction amount
3	Class	0 - Normal; 1 - Fraud

The dataset is highly imbalanced, the positive class (frauds) account for 0.167% of all transactions as shown in Table II.

TABLE II
Distribution of target class

Class	Count	Percentage
0	283253	99.833
1	473	0.167

From the distribution shown in figure 2, we can see that the distribution of most of the PCA components is Gaussian, and many may be centered around zero, suggesting that the variables were standardized as part of the PCA transform.

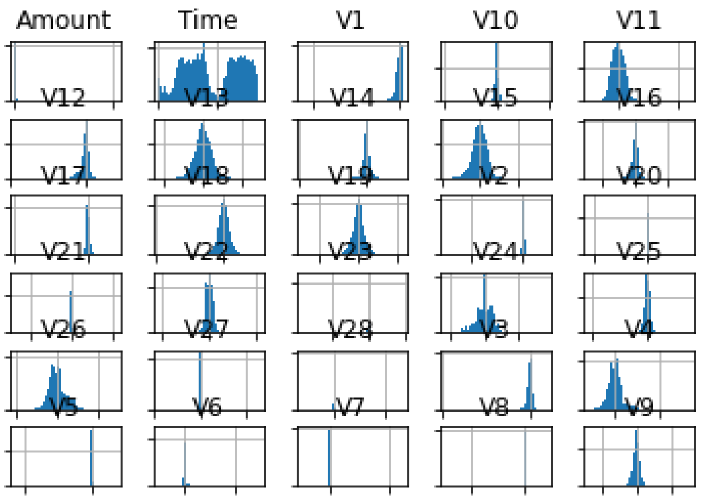

Fig. 2. Distribution of the features of data set

The Amount feature does not appear on the histogram. Most values for 'Amount' are small, with a mean of about 88 and the middle 50 percent of observations between 5 and 77. The largest value is about 25691, which is pulling the distribution up and might be an outlier (e.g. someone purchased a car on their credit card). The correlation matrix as shown in figure 3 explains that feature 'Class' is independent of both the 'Amount' and 'Time' features. It is dependent on PCA applied attributes[35].

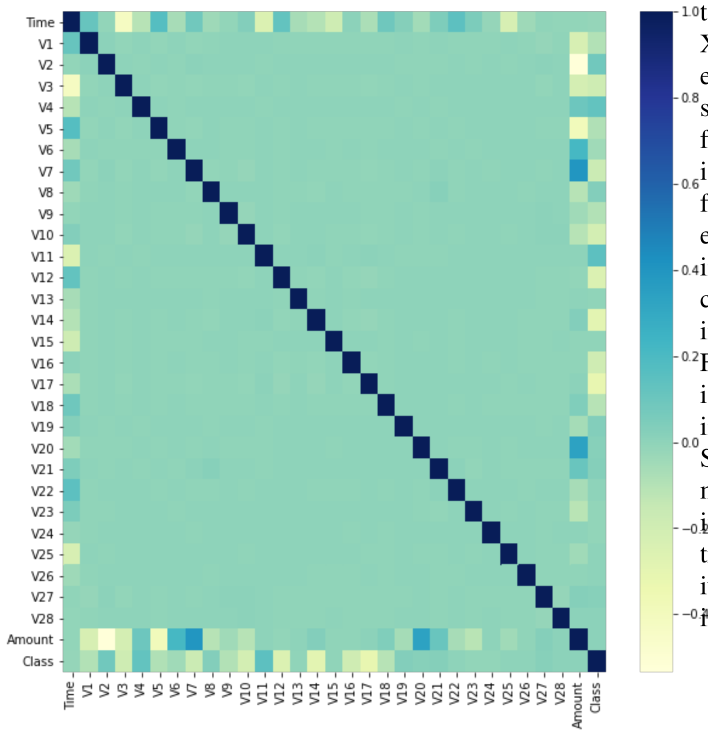

Fig. 3. Correlation Matrix for Features

The challenges with the dataset are as listed below:

- Imbalanced nature: The Credit card dataset is highly imbalanced because there will always be more legitimate transaction when compared to fraudulent ones[35]. This causes the detection of fraud transactions to be difficult and imprecise.
- Overlapping data: many transactions may seem fraudulent, when actually they are normal (false positive) and conversely, a fraudulent transaction may be considered legitimate (false negative). Hence, obtaining low rate of false positive and false negative is a key challenge in fraud detection systems[36]
- Lack of adaptability: Detection system should be able to detect new types of frauds. Classification algorithms usually face the problem of detecting new types of normal or fraudulent patterns[37].
- Fraud detection cost: The system should take into account both the cost of fraudulent behavior that is detected and the cost of preventing it. For example, no revenue is obtained by stopping a fraudulent transaction of a few dollars[38]
- Lack of standard metrics: There is no standard evaluation metric available for assessing and comparing the results of fraud detection systems.

V. Methodology

A set of experiments were run for Credit Card Fraud data which is an imbalanced data set. The comparison is made for Hyperopt vs Optuna vs random search optimization techniques for the 3 algorithms – CatBoost, LightGBM and XGBoost. At random 15 seeds were selected to run the experiments using Hyperopt and Optuna and 5 seeds were selected for random search optimization. The experiment had following aspects:
i. The optimization was done on cross validation with 5 folds, logloss as objective function and F1 score as custom evaluation metric.
ii. The range and space for hyper parameters was kept constant in all the experiments.
iii. Each experiment was run on 300 trials.
Following information was recorded:
i. *Train time* - Time taken for overall run of the experiment
ii. *F1 "loss"* - Since the dataset was highly imbalancedF1 Score was selected as evaluation metric and the loss to be minimized for optimization was set to "1 - F1 score"
iii. *Hyperparameters* - The value of hyperparameters for each trial in every experiment were stored to observe convergence
iv. *Estimators* - The best boosting round trees were stored independently for each experiment trial

VI. Results

A. Comparing Algorithm-wise Optimization Techniques

The search space for all 3 algorithms was kept very broad in order to ascertain the strength of each optimization technique. The learning rate for tuning was kept at 0.3 for 5 fold cross validation, however, during the training on optimized hyperparameters the learning rate was set to 0.5. The objective for the experiments was *binary logloss* and a custom evaluation meteric was create for XGBoost and LightGBM for F1-score and inbuilt F1 metric was used for CatBoost. The machine was run in parallel with *nthread* set at 8 on Google Cloud Platform (GCP) deep learning virtual machine.

1) XGBoost: This algorithm had the ranges and choices for different parameters as shown in table III. All optimization techniques had similar search space with the same ranges. As evident from figure 4, Random search on an average took 7 hours more than Hyperopt and Optuna for optimization. Values

TABLE III
Hyperparameter Range for XGBoost

Hyperparameters	Choices
$booster$	$gbtree$
$gamma$	0.001–100
$lambda\_l1$	3–7
max_depth	0.7–1
$subsample$	0.7–1
$sampling\_method$	$Uniform$
$colsample\_bytree$	0.7–1
$colsample\_bylevel$	0.001–100
reg_lambda	0.001–100
reg_alpha	0.001–100
$tree\_method$	$approx, hist$
$grow\_policy$	$depthwise, lossguide$
max_leaves	max_depth^2–1
$predictor$	$cpu\_predictor$

obtained for ‘minimum loss’ are also more as compared Hyperopt and Optuna as shown in figure 5. However, ‘minimum loss’ values for Hyperopt and Optuna are approximately in same range for all the experiments. For corresponding time,

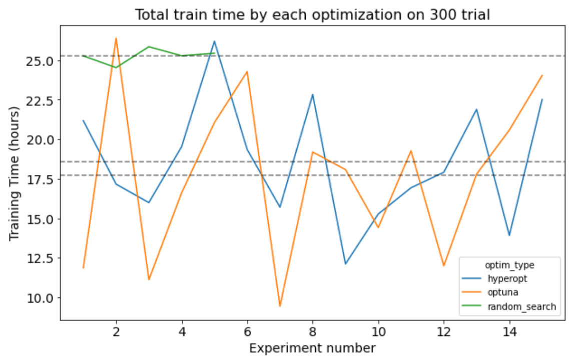

Fig. 4. Total optimization time taken by each experiment for Hyperopt, Optuna and random search techniques to run 300 trials for XGBoost

the best loss for 15 experiments each in Hyperopt and Optuna and 5 in Random Search are shown in figure 10

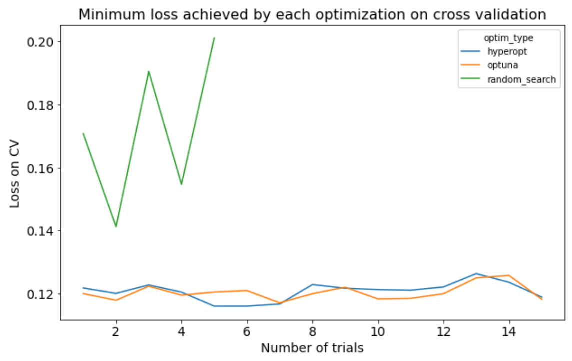

Fig. 5. Minimum loss values obtained for each experiment for Hyperopt, Optuna and random search techniques during the run of 300 trials for XGBoost

The distribution of losses for each experiment during the 300 trials is shown for Hyperopt and Optuna in figure 6 and 7 respectively. The dots indicate the best loss for each experiment and the correspondin x-axis value indicates the number of trial at which it was achieved. For both the techniques, minimum loss is usually attained beyond 130 trials.

It is evident from figure 8 that for most of the experiments Hyperopt took more than 250 iterations and 12 hours to obtain minimum loss. While Optuna took less than 12 hours and 200 iterations.

2) LightGBM: This algorithm had 8 important features to optimize. The table IV shows the parameter and selected ranges.

Despite the similar search space for all optimization techniques, Hyperopt optimization took a great time for training when compared to Optuna or Random Search as shown in figure 9.

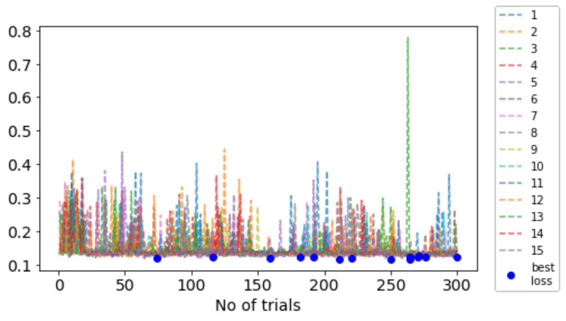

Fig. 6. XGBoost: Distribution of loss recorded while running for 300 trials with Hyperopt optimization.

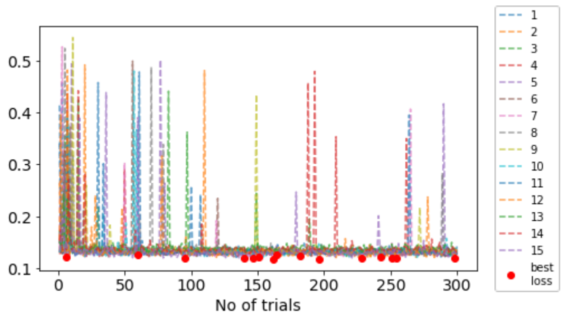

Fig. 7. XGBoost: Distribution of loss recorded while running with Optuna optimization while running for 300 trials with Optuna optimization.

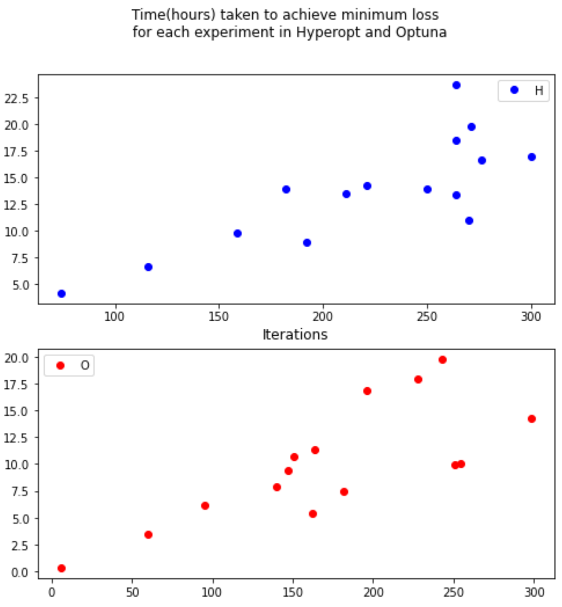

Fig. 8. Comparison of total time and number of iterations taken by XGBoost to reach best loss in each experiment with Hyperopt and Optuna seperately.

For corresponding time, the best loss for 15 experiments in each Hyperopt and Optuna and 5 in Random Search are shown in figure 10

The distribution of losses for each experiment during the 300 trials is shown for Hyperopt and Optuna in figure 11

TABLE IV
HYPERPARAMETER RANGE FOR LIGHTGBM

Hyperparameters	Choices
$boosting\_type$	$gbdt/goss$
num_leaves	$31 - 1023$
$lambda\_l1$	$0.0 - 0.5$
$lambda\_l2$	$0.0 - 0.5$
$min\_gain\_to\_split$	$0.0 - 5.0$
$min\_data\_in\_leaf$	$20 - 500$
min_child_weight	$0.1 - 10$
$feature\_fraction$	$0.4 - 1.0$

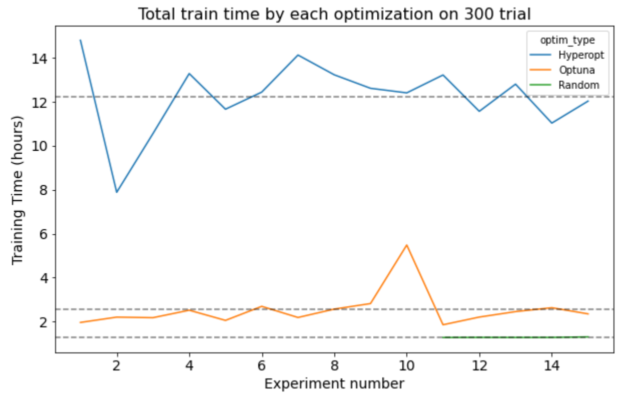

Fig. 9. Total training time take by each experiment for Hyperopt, Optuna and random search techniques to run 300 trials for LightGBM

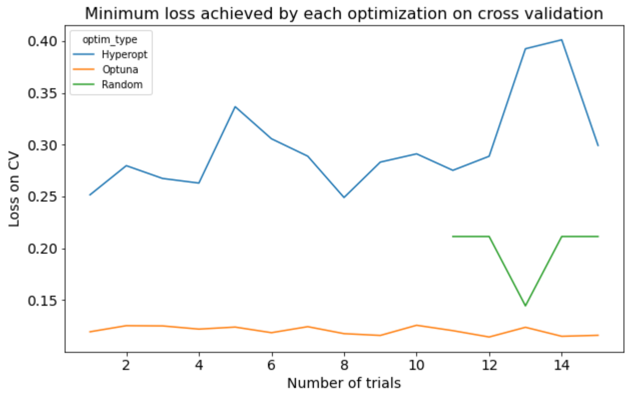

Fig. 10. Best loss for each experiment for Hyperopt, Optuna and random search techniques during run of 300 trials for LightGBM

and 12 respectively. The dots indicate the best loss for each experiment and the corresponding value on the x-axis which indicates the number of trial at which it was achieved. The trials indicate that the loss for Hyperopt did not converge and remained of same width during the trials while in Optuna the loss continuously diminished after first 100 trials. The reason for this is because the Optuna module is integrated with lightgbm and sequentially tunes the module.

The best trials for both hyeropt and Optuna came after running for more than 150 trials out of 300 for most of the

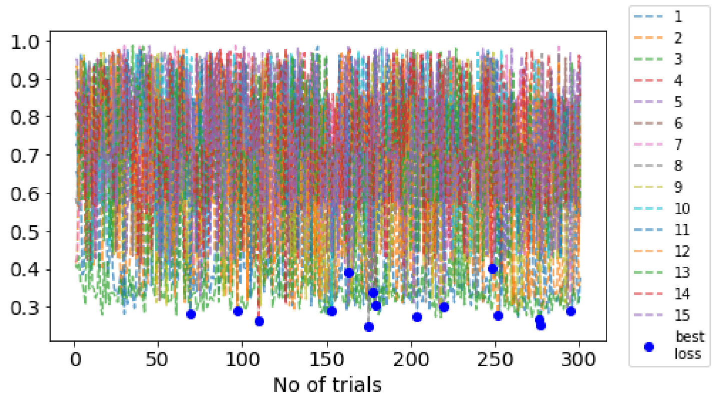

Fig. 11. Distribution of loss recorded while running 300 trials using Hyperopt optimization. The loss value resembles noise and does not converge even towards the end.

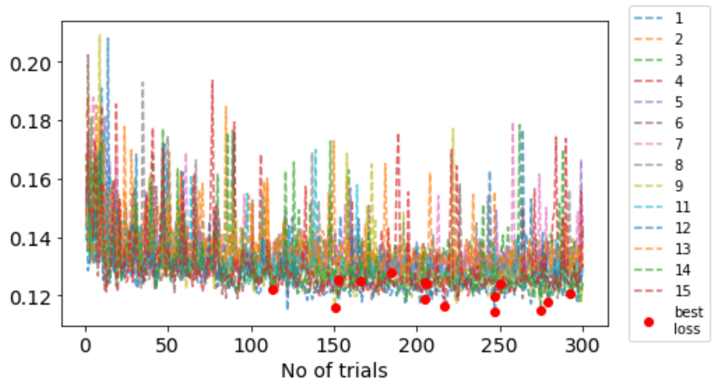

Fig. 12. Distribution of loss recorded while running with Optuna optimization. It is noticeable that the best loss was mostly achieved after 150 trials from total of 300 trials.

experiments. While the time taken is highly dispropotionate. The average time taken for Hyperopt to optimize is about 5 times more than Optuna as shown in figure 13 and still the best loss is about 2.5 times of worst loss by Optuna as shown in figure 10 before.

3) CatBoost: This algorithm had 8 important features to optimize. The table V shows the selected hyperparameter and its ranges.

TABLE V
HYPERPARAMETER RANGE FOR LIGHTGBM

	Low Range	High Range
$boosting\_type$	gbdt	goss
num_leaves	31	1023
$lambda\_l1$	0.0	0.5
$lambda\_l2$	0.0	0.5
$min\_gain\_to\_split$	0.0	5.0
$min\_data\_in\_leaf$	20	500
min_child_weight	0.1	10
$feature\_fraction$	0.4	1.0

The amount of time required to complete 300 trials for each experiment is depicted in figure 14. There is no noticeable performance difference between Optuna and Hyperopt, however,

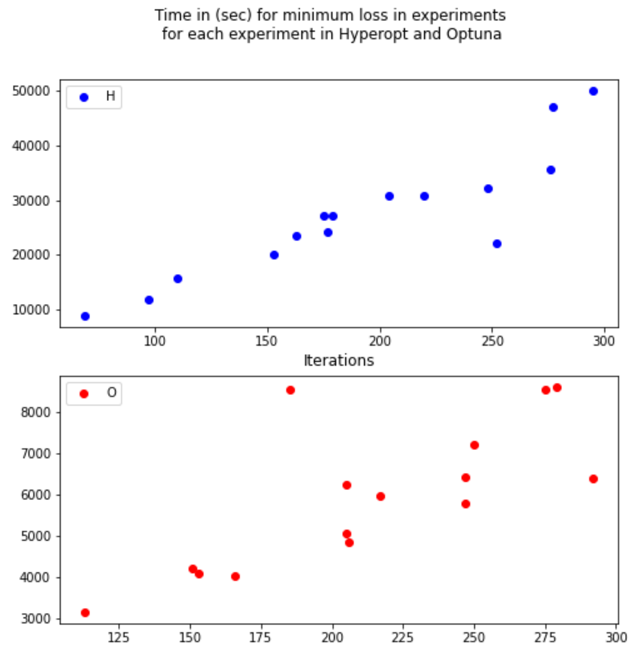

Fig. 13. Comparison of total time and nuber of iterations taken to reach best loss by each experiment in Hyperopt and Optuna tuning for LightGBM

Random Search needs significantly longer time to complete the 300 trials of each experiment.

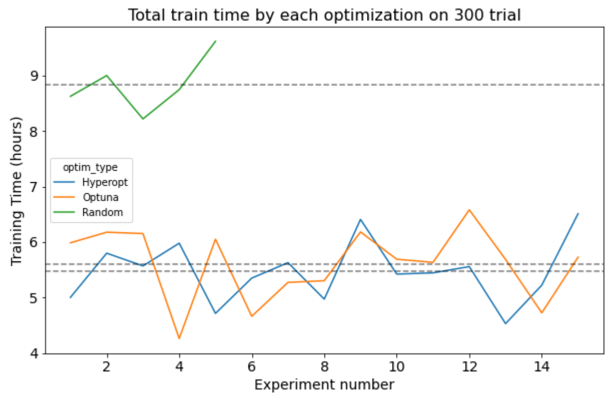

Fig. 14. Total training time taken by each experiment for Hyperopt, Optuna and random search to run 300 trials on CatBoost

In terms of best loss, Optuna usually outperforms the Hyperopt, whereas random search is the worst performer of all the three as shown in figure 15

It is also evident that Hyperopt typically converges to the best loss after the 150th trial, in comparison, Optuna can achieve the best loss at earlier trials as shown in figure 16 and 17. Another important factor is that the distribution of losses in every iteration are bulkier for Hyperopt than Optuna. In other words, high amplitude losses are more common in Hyperopt.

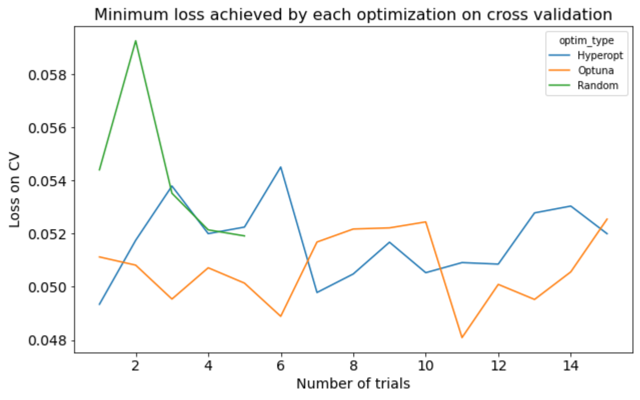

Fig. 15. Best loss for each experiment for Hyperopt, Optuna and random search techniques during run of 300 trials for CatBoost

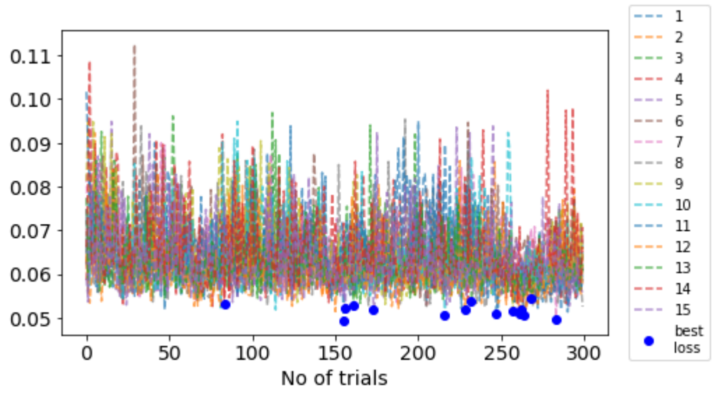

Fig. 16. CatBoost: Distribution of loss recorded for 15 with each running 300 trials using Hyperopt optimization.

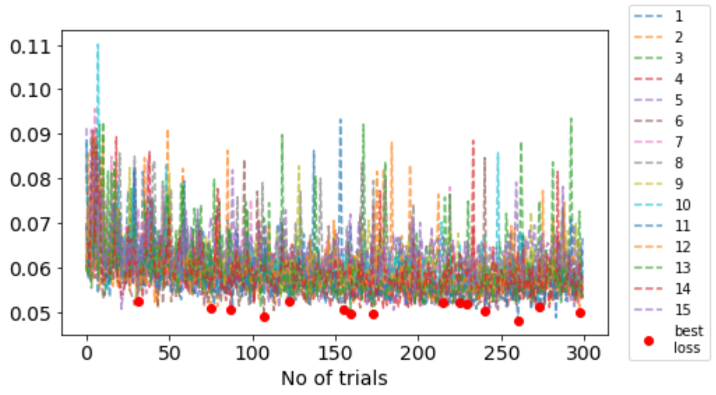

Fig. 17. CatBoost: Distribution of loss recorded for 15 Optuna experiments each with 300 trials.

The aggregated time to achieve the best loss was approximately 62 hours for Hyperopt and 52 hours for Optuna. So, Optuna is approximately 16% faster in finding the parameters of the best model; the ASHA pruning technique, which is a property of Optuna, explains this improvement. Further in figure 18 it can be observed that in many experiments, Optuna reaches the best loss around the 150th trial, whereas Hyperopt depends onto more trials to achieve the same task, and of course requiring more trials translates into longer duration to reach the best loss.

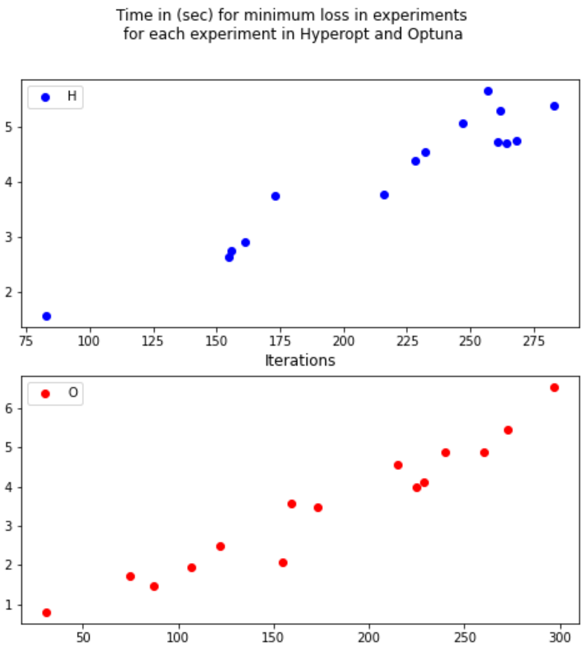

Fig. 18. Comparison of total time and nuber of iterations taken to reach best loss by each experiment in Hyperopt and Optuna tuning for CatBoost

B. Comparing Algorithm Optimization Performance

This section compares the performance of XGBoost, LightGBM and Catboost. The comparison for $ROCAUC$ and $PRAUC$ is done for Hyperopt and Optuna as shown in figures 19 and 20.

Since the number of experiments were 15 each for Hyperopt and Optuna and the distribution was not assumed to be normal, the following 3 tests were performed to compare the algorithm performance:

- Wilcoxon Rank Sum Test: This test was perfomed to measure ROC AUC, PR AUC and F1-Score performance.
- Comparing ROC: Delong and Bootstrap methods were run to compare the ROC AUC for minimum median and maximum value for Hyperopt and Optuna.
- Kappa Coefficient: This test was run on prediction based on default of 0.5 and best cut-off defined by fpr-fnr graph

The table VI presents comparison for algorithms' performance in maximizing ROC AUC, PR AUC and F1-Score. One-sided wilcoxon Rank Sum test was performed to compare if results of left side are better than right side, so for "XGB vs LGB" one sided test was to see if performance of XGBoost

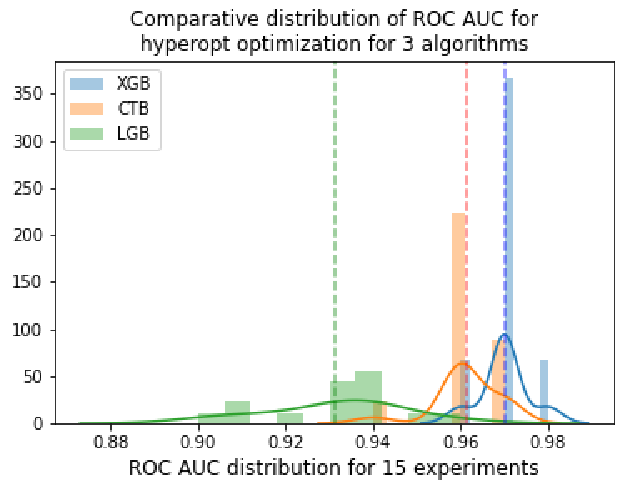

Fig. 19. Comparison of ROC AUC for Hyperopt optimization for 3 algorithms

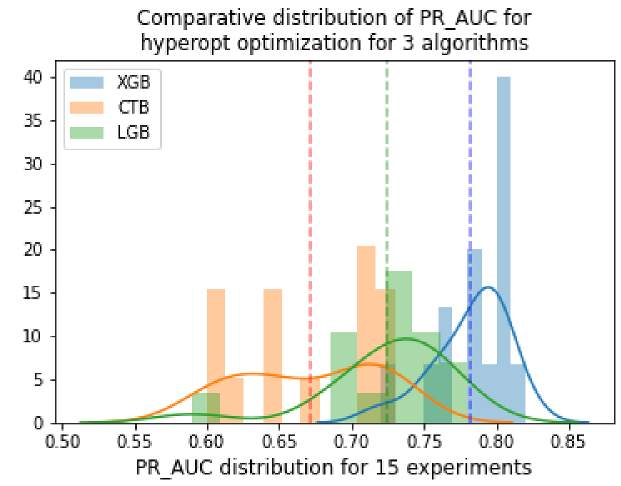

Fig. 20. Comparison of PR AUC for Hyperopt optimization for 3 algorithms

was greater than LightGBM.The results are present for both Hyperopt and Optuna. The reults indicate that with Hyperopt optimization XGBoost performance was greater than LightGBM and Catboost with regards to ROC curve, AUC curve and F1-Score. While using Optuna module performance of LightGBM was better.

TABLE VI
WILCOXON RANK SUM TEST

	XGB vs LGB	XGB vs CTB	LGB vs CTB
Hyeropt			
$ROCAUC$	$1.063e-06$	0.0004582	1
$PRAUC$	$7.236e-05$	$2.44e-06$	0.001662
$F1-Score_{0.5}$	$3.455e-05$	$1.529e-06$	$1.662e-06$
$F1-Score_{best}$	tbc	tbc	tbc
Optuna			
$ROCAUC$	0.9927	0.002472	$1.094e-05$
$PRAUC$	0.9998	$6.991e-06$	$1.372e-062$
$F1-Score_{0.5}$	1	$1.641e-06$	$1.394e-06$
$F1-Score_{best}$	tbc	tbc	tbc

Another crucial test was to compare the ROC curve itself.

suing Delong's and Bootstrap. The results are shown in the table VII

TABLE VII
COMPARING ROC AUC

	XGB vs LGB	XGB vs CTB	LGB vs CTB
Maximum ROC AUC			
$Delong's$	0.0008061	0.247	0.04517
Median ROC AUC			
$Delong's$	0.01895	0.02417	0.9262
Minimum ROC AUC			
$Delong's$	0.008029	0.0084	0.472

Finally, the Kappa Coefficient test is used as it is a very good measure that can handle very well imbalanced class problems. Even though, there is no standardized way to interpret its values. Landis and Koch (1977) provide a way to characterize values. According to their scheme a value < 0 is indicating no agreement , 0–0.20 as slight, 0.21–0.40 as fair, 0.41–0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1 as almost perfect agreement.

TABLE VIII
COMPARING KAPPA COEFFICIENTS

	Hyperopt	Optuna	Random Search
XGBoost	0.84	0.73	.23
	0.79	0.78	0.13
	0.66	0.69	0.15
LightGBM	0.84	0.73	.23
	0.79	0.78	0.13
	0.66	0.69	0.15
CatBoost	0.84	0.73	.23
	0.79	0.78	0.13
	0.66	0.69	0.15

C. Comparing Feature Importance

In our case, we implement the SHAP technique. Shapley feature importance distributes the overall performance of a model among the features according to the marginal contributions. Primarily, SHAP values of a stratified sample of test dataset are extracted. The shap values of each feature vary for every instance of data. A particular feature (i) could produce strong positive impact on output of instance (j), while the same feature (i) produces strong negative impact on output of instance (k). As a result, first, the absolute values of every feature's impact are computed. Then those absolute values are averaged per each feature. The same procedure is repeated for every experiment. The results are illustrated in figure 21
Discussion:
In terms of model stability LightGBM optuna consistently relies on specific features across all experiments. While LightGBM utilizes all the features for prediction, Catboost strongly polarizes the features and avoids using the unimportant ones. This behaviour must be investigated, since LightGBM might be able to detect non-linearities and feature combinations that CatBoost overlooks. One solution is to generate random noise features and observe how our LightGBM and Catboost models

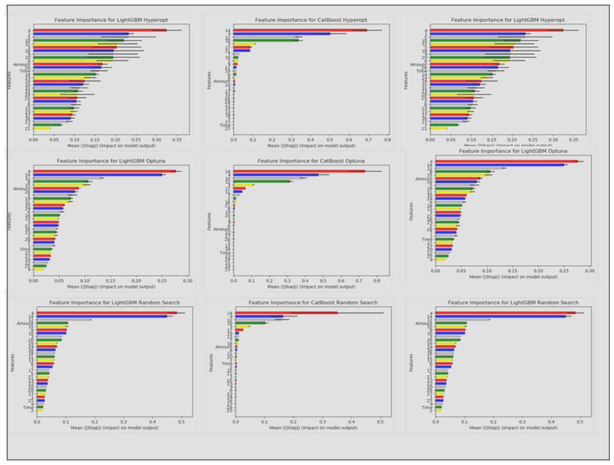

Fig. 21. Feature Importance: Mean and Variance in different GBT algorithms

rank their importance Vs the current features. As a final note, features with importance near zero, are unlikely to generate any strong additive or multiplicative effect when combined with other features, specially that feature correlations are weak, therefore, for CatBoost it is plausible to remove features 25,28,27 and time.

VII. CONCLUSION

To be done[8].

REFERENCES

[1] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.
[2] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
[3] R. Johnson and T. Zhang, "Learning nonlinear functions using regularized greedy forest," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 5, pp. 942–954, 2013.
[4] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.
[5] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in neural information processing systems*, 2017, pp. 3146–3154.
[7] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Advances in neural information processing systems*, 2018, pp. 6638–6648.
[8] C. Antonio, "Sequential model based optimization of partially defined functions under unknown constraints," *Journal of Global Optimization*, pp. 1–23, 2019.
[9] F. Hutter, "Automated configuration of algorithms for solving hard computational problems," Ph.D. dissertation, University of British Columbia, 2009.
[10] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.
[11] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*, vol. 13. Citeseer, 2013, p. 20.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2623–2631.

[13] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1487–1495.

[14] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[15] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[16] A. Géron, "Hands-on machine learning with scikit-learn, keras," in *and TensorFlow: Concepts, Tools, and TEchniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[17] J. Carrasco, S. García, M. Rueda, S. Das, and F. Herrera, "Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review," *Swarm and Evolutionary Computation*, vol. 54, p. 100665, 2020.

[18] T. Hoskin, "Parametric and nonparametric: Demystifying the terms," in *Mayo Clinic*, 2012, pp. 1–5.

[19] Student, "The probable error of a mean," *Biometrika*, pp. 1–25, 1908.

[20] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2020.

[21] G. W. Corder and D. I. Foreman, "Nonparametric statistics for non-statisticians," 2011.

[22] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[23] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[24] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.

[25] S. Hooker, D. Erhan, P. Kindermans, and B. Kim, "Evaluating feature importance estimates," *CoRR*, vol. abs/1806.10758, 2018. [Online]. Available: http://arxiv.org/abs/1806.10758

[26] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *Advances in Neural Information Processing Systems*, 2017, pp. 6967–6976.

[27] R. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," *CoRR*, vol. abs/1704.03296, 2017. [Online]. Available: http://arxiv.org/abs/1704.03296

[28] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in neural information processing systems*, 2017, pp. 4765–4774.

[29] E. Al Daoud, "Comparison between xgboost, lightgbm and catboost using a home credit dataset," *International Journal of Computer and Information Engineering*, vol. 13, no. 1, pp. 6–10, 2019.

[30] A. Anghel, N. Papandreou, T. Parnell, A. De Palma, and H. Pozidis, "Benchmarking and optimization of gradient boosting decision tree algorithms," *arXiv preprint arXiv:1809.04559*, 2018.

[31] A. V. Dorogush, V. Ershov, and D. Kruchinin, "Why every gbdt speed benchmark is wrong," *arXiv preprint arXiv:1810.10380*, 2018.

[32] A. Ben-David, "Comparison of classification accuracy using cohen's weighted kappa," *Expert Systems with Applications*, vol. 34, no. 2, pp. 825–832, 2008.

[33] J. Provost Foster, F. Tom, and K. Ron, "The case against accuracy estimation for comparing induction algorithms," in *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 445–453.

[34] S. García, A. D. Benítez, F. Herrera, and A. Fernández, "Statistical comparisons by means of non-parametric tests: a case study on genetic based machine learning," *algorithms*, vol. 13, p. 18, 2007.

[35] V. N. Dornadula and S. Geetha, "Credit card fraud detection using machine learning algorithms," *Procedia Computer Science*, vol. 165, pp. 631–641, 2019.

[36] S. Maes, K. Tuyls, B. Vanschoenwinkel, and B. Manderick, "Credit card fraud detection using bayesian and neural networks," in *Proceedings of the 1st international naiso congress on neuro fuzzy technologies*, 2002, pp. 261–270.

[37] Z. Zojaji, R. E. Atani, A. H. Monadjemi *et al.*, "A survey of credit card fraud detection techniques: data and technique oriented perspective," *arXiv preprint arXiv:1611.06439*, 2016.

[38] S. Stolfo, D. W. Fan, W. Lee, A. Prodromidis, and P. Chan, "Credit card fraud detection using meta-learning: Issues and initial results," in *AAAI-97 Workshop on Fraud Detection and Risk Management*, 1997, pp. 83–90.