Høgskolen i Østfold

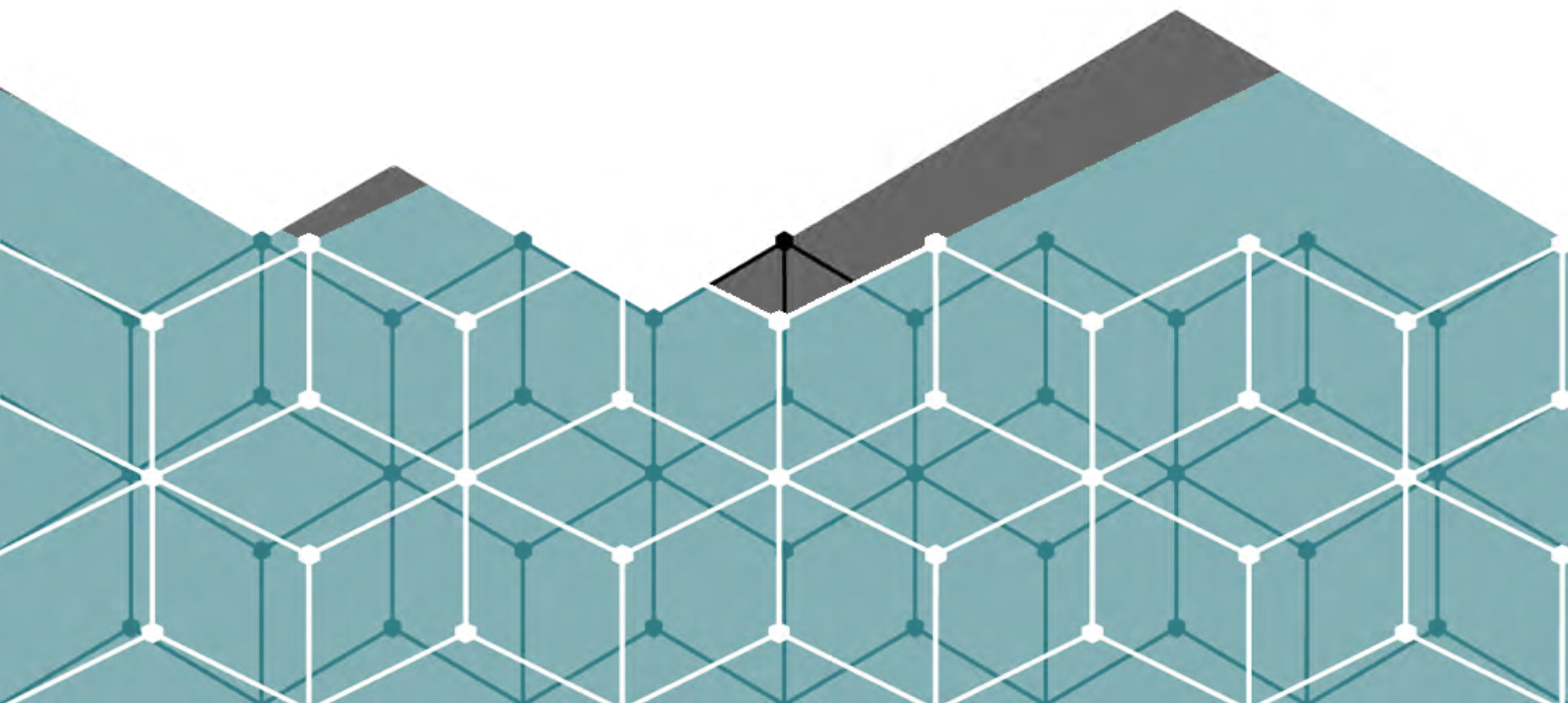
Comparative Analysis of:

- classification accuracy for XGBoost, LightGBM, CatBoost, H2O, and Classifium

Master's Thesis

Shubodha Acharya

Master thesis project· 60 hp
Østfold University College
Faculty of Computer Science, Engineering and Economics
Department of Computer Science and Communication
Halden, Norway.
2021



MASTER'S THESIS
ON
Comparative Analysis Of Classification Accuracy For XGBoost, LightGBM, CatBoost, H2O, and Classifium.

Shubodha Acharya

A thesis presented for the degree of Master in Applied Computer Science

School of Computer Sciences
Østfold University College
Halden
October 18, 2021

About this document

This document contains the work done for master's thesis. It is implemented in LaTeX, the most popular, advanced, flexible, and comprehensive documentation system in natural sciences. However, the document as such could be implemented with other tools, as well.

The thesis covers the topic of machine learning, its types, application, and advance algorithms that has been used for classification task. Thesis presents the extensive work done for addressing the scope of its topic. Also, codes, pipelines and mathematical calculation done during the course of work are presented in the Appendix of this document.

The structure followed for building this document is based on the widely used model (Introduction, Methods, Results, And Discussion)[1].

The document is developed using a standard template designed by Gunnar Misund, associate professor at HIOF (Østfold University College) and also, it is recommended by HIOF[2]. The template is designed on the basis of Gunnar Misund 25 years of teaching experience, and "The Mayfield Handbook of Technical and Scientific Writing" [10][3].

Finally: Review, Comments, bug reports, and suggestions are highly welcome[4]!

Shubodha Acharya
Halden, October 18, 2021

[1] en.wikipedia.org/wiki/IMRAD
[2] www.ia.hiof.no/~gunnarmi
[3] www.mhhe.com/mayfieldpub/tsw/home.htm
[4] subodhrajacharya@gmail.com

Abstract

Hyperparameter tuning for any machine learning algorithms can play significant role on determining the algorithm's performance in terms of classification accuracies or regression analysis. This research work presents a comparative study between the automatic machine learning algorithm Classifium, and manually tuned gradient boosting decision tree-based machine learning algorithms, XGBoost, LightGBM, and CatBoost implemented with the hyperparameter tuning pipeline, and the other automatic machine learning library H2O, for their classification accuracy. We used the 7 different data sets, which were originally taken from UCI repository, and developed the classification models with all our algorithms. And when compared for their classification accuracy, we observed that in majority of classification task with all of the 7 data sets, Classifium outperformed manually and carefully tuned XGBoost, LightGBM and CatBoost, following the hyperparameter tuning pipelines, which we proposed for this study based on relevant work and official documents of algorithms. When we ran H2O by assembling algorithms like deep learning, gradient boosting, random forest, and only with XGBoost separately, we found that the classification accuracies obtained from Classifium was better than the classification accuracies obtained from H2O with its own hyperparameter tuning pipeline for the both approaches. From this study, we found that the automatic hyperparameter tuning pipeline implemented in Classifium gives better or similar results compared to the carefully tuned gradient boosting algorithms, like XGBoost, LightGBM and CatBoost implemented with the manual hyperparameter tuning pipelines, and H2O with the automatic hyperparameter tuning pipeline.

Keywords: Machine Learning, Gradient Boosting, XGBoost, LightGBM, CatBoost, AutoML, Hyperparameter

Acknowledgments

I would first like to thank my supervisor, Dr. Roland Olsson for his invaluable guidance, continuous support and supervision during my thesis. It was my honour to conduct this work under his immense knowledge and expertise, which has contributed a lot to sharpen my knowledge.

I am also thankful towards head engineer Perparim Mustafa and associate professor Gunnar Misund at Østfold University College, Halden, For their kind technical support during execution and documentation of this thesis work.

I am grateful towards Dr. Monica Kristansen Holone, head of department of Information Technology and Communication, Østfold University College, Halden, for encouraging and supporting me during my master's study.

In addition, I am thankful towards my family and friends for their continuous love, support and encouragement.

This would may not have been possible without their contribution.

Prerequisite

Since this thesis covers the theoretical and technical aspect of machine learning and some advance algorithm. So it is advised for reader to have basic knowledge about the machine learning, automatic machine learning, gradient boosting and hyperparameter tuning to get the most clear understanding from this document.

Contents

List of Figures

List of Tables

List of Code

Chapter 1

Introduction

1.1 Machine Learning

The term Machine Learning was coined by Arthur L. Samuel in 1959 [87], which means that the machines performing the task autonomously by learning from the past experience and improving its performance. In 1959 Arthur L. Samuel published his work on "Some Studies in Machine Learning using Game Checkers" [1], in which he has described the concept of how a computer can be trained to play a game by itself better than a person who develop that game. Also, in the early 1950s, Alan Turing developed a machine which could imitate the human voice for his imitation game [14] and through which he raised the question "can machines think?" [29] According to Mitchell [9] the process of machine learning can be defined as, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E".

In other words, machine learning can be defined as the domain of computer science in which the computer programs are fed with data, and the computer programs analyse those data and find the patterns from that data using some set of algorithms. When new similar kind of data is fetched in same program, the algorithm responds in the same way that it has learned from its past experience. For analysing the data, machine learning algorithm depends on probability and statistics which are the domains of mathematics [68]. Just like human brain learns from the past experience, in machine learning, computers or machines are supposed to learn from the data in the form of experience. The machine learning technique has been applied in engineering, economics, medical research, automobiles and in every other sectors which are growing in today's world. For example, machine learning technique is used to classify the email as spam or non-spam [23], classifying the likelihood of person as diabetic or non-diabetic [72]. Also, this technique is used to predict the stock price [39], housing price [46] and et cetera. In automobile industry machine learning technique is applied to produce the cars which can automatically avoid collision, change the driving lanes and navigate in urban area [25]. Beside these, machine

learning technique is used for advertisement recommendation, movie recommendation, face detection, person detection in security camera, and many more.

1.1.1 Machine Learning Methods

According to [78], [31], and [66] machine learning methods can be generalised into four categories based on the data from which they are learning and the algorithms on which they are learning.

1. Supervised Learning
   In supervised learning methods, the machines are trained with the set of examples containing the labeled outputs for corresponding input data sets or features. From those examples they learn the relationship between input features and labeled output. On the basis of their learning experience, machine performs the task of classification or regression on the data which are fed without the labeled output. In abstract form, machine is trained with set of input features 'X' and their corresponding set of output label 'y'. It maps the relations of each input attribute of 'X' with output 'y'. Later, when a set of new data is added in 'X' without labeled output 'y', machine will perform classification or regression on the basis of experience that it has gained by learning the input set 'X' with labeled output 'y'. Let us consider the examples of classification of emails and prediction of car prices. In classification of emails, machine is trained with the features of emails like domain name, IP address, host name, format, and et cetera with the corresponding output label as spam or non-spam [71]. In future, the trained machine can classify new emails as spam or non-spam based on its previous experience. Similarly the price of car can be predicted with supervised machine learning method, where machine is first trained with the example containing features and corresponding price of car and in future only features are given as input to predict the price. According to [75], supervised learning methods best fit in scenarios where the user is aware about the output as categories or continuous values, but is unaware about the corresponding output. However, to find the patterns in unlabeled data set using machine learning technique, we need to explore other dimension of machine learning methods like Unsupervised Learning.

2. Unsupervised Learning
   In unsupervised machine learning methods, the machines are trained with the set of data without labeled output or target class. In this method machines learns the pattern by finding relation among the features of the data set. The data used in unsupervised machine learning have no predefined output attribute. In the book Hands-on Machine learning with Scikit-Learn, Keras and Tensorflow, [78] has explained the unsupervised learning using the example of dividing the visitors on some blog post into clusters based on their sex and age group. In this example [78] explains that with unsupervised learning methods, machines are introduced only with the features of the visitors along

with their sex and age included in the feature set. Required target features for classification are not explicitly defined. Machines learns the data of visitors and finds how many of visitors were male and female, how many of visitors were young, adult or old and so on. According to [78] other relevant examples of unsupervised learning can be detecting anomaly in credit card transaction, catching manufacturing defects and association rule learning [20]. The most general type of unsupervised learning methods are clustering and association [68] [66].

3. Semi-supervised Learning
   Semi-supervised learning method uses both the principles of Supervised learning and Unsupervised learning methods partially. In this learning methods the data set have both labeled and unlabeled data, but comparing to unlabeled data, labeled data are very few in volume [78]. To create labeled training data set, it can take lot of time and man power resulting the process to be expensive. For example in computer-aided diagnosis, drug discovery and part of speech tagging, it can be difficult to obtain labeled data [86]. Semi-supervised learning methods can be useful in areas where labeled data are very few and unlabeled data are enormous. [78] has given the example of photo hosting service, Google Photos, as use case scenario of semi-supervised learning. In Google Photo while uploading our family photo if we label the members of the family with names in few pictures, the system will automatically recognize them in rest of the uploaded pictures and will assign their name as label.

4. Reinforcement Learning
   In reinforcement learning the machine or learning agent is set in its learning environment and is exposed to different tasks. While performing those tasks learning agent can make mistakes or it can do that task correctly. The basic concept of reinforcement learning is that the learning agent learns by trial and error. It is warned when it makes mistakes and it is rewarded when it performs the task correctly and the objective of the learning agent is to maximize its reward [68]. [78] gives the example of Deepmind's Alphago program as an example of reinforcement learning, which learned to play game of Go by trail and error method and succeeded to beat human.

1.2 Classification

Classification is the act of separating the instances from a collection and assigning those instances, having the same features or characters, together, to a group or class. For example, if 'D' represents a huge collection of numeric and alphabetical values, the task of classification can separate those values into their distinctive classes. According to [2] a classification allocates entities to initially undefined classes so that individuals in a class are in some sense close to one another. Classification is a useful tool for making decisions because different outcomes of a decision can be categorized

into different classes and study can be done on the course of each outcome, which is one application classification [4]. During the task of classification, if the instance is classified under supervision it is called supervised classification and if the instance is classified without any supervision it is called unsupervised classification. Here supervision refers to the preliminary knowledge assigned to the instance about its attributes and its respective output class. The relevant example of knowledge can be labeled data with its output class. Classifying emails into spam and non-spam class, classifying customers as 'bad debtor' or 'good debtor' based on their economic status, classifying if the person is prone to diabetes or not, based on his/her genetic information, are supervised classification. And categorizing visitors of an online store based on their sex, age, interest can be unsupervised classification. For performing the classification on big data, a machine learning approach is used which also automates the process of classification [6]. In machine learning methods, different algorithms are used for performing supervised and unsupervised classification. According to [4], in machine learning methods, algorithms that produce logistic solutions are used, like decision tree, which is one of the best examples for supervised classification. In the decision tree approach, algorithms like C5.0, CART, Random Forest, Gradient Boosting are used for performing classification and regression. According to [4] and [6], generally in machine learning there are also other different approaches used for performing classification and they are the Statistical method, and Neural Network method.

1. Statistical method Statistical method is purely based on mathematics. According to [6], statistical methods are generally characterised by having an explicit underlying probability model, which provides a probability of being in each class rather than simply a classification. In addition, it is usually assumed that the technique will be used by statisticians, and hence human intervention is assumed. Statistical methods are adopted to develop machine learning classification algorithms like ID3, C4.5, CART and other. Some methods, based on statistical classification, used for classification of data are linear discriminant, quadratic discriminant, nearest neighbours, Bayes independence, Bayes second order [4].

2. Neural Network method The artificial neural network or neural network consists of neurons or perceptrons [8], which receives information as input data from source or other neurons and gives the information to other neurons or to receiver by analysing and adjusting the information it receives. The neurons are placed vertically in a row and there can be more than two vertical row of neurons connected to each other. The working mechanism of neuron network is influenced with human biological nervous system [19]. Artificial neuron network have been vastly used for pattern recognition and classification [35].

1.3 Classification algorithms

Here we will discuss about the algorithms that has been used for the classification and regression task since the beginning. In 1985, Quinlan proposed an algorithm, ID3 [3], which is used to generate simple decision tree [5] from a set of training data. The algorithm is used to perform classification of discrete data. For classification and regression, Breiman et al. proposed an algorithm called CART which stands for Classification and Regression trees [28]. To overcome the limitation of ID3 algorithm on processing speed, efficiency, volume of data, Quinlan proposed C4.5 algorithm [43], which was upgraded version of ID3 algorithm. Still c4.5 had some limitation on its predicting accuracy, memory requirements and efficiency. Quinlan proposed most efficient algorithm of that time, C5.0 or See5 [21]. C5.0, Random Forest and Gradient Boosting algorithms are amongst the more popular algorithms for their advanced features and higher efficiency, and are further investigated in this study.

1.3.1 C5.0

John Ross Quinlan proposed the most efficient algorithm of that time, C5.0 or See5 [21]. C5.0 is supposed to be one of the effective classification algorithms among ID3, CART and C4.5 [59]. C5.0 algorithm generates the decision tree using the top-down iteration approach. It splits the variables into two different sets based on their information gain as splitting criteria. Boosting and construction of cost-sensitive trees are two important features implemented in C5.0 which makes it an efficient classification algorithm. In boosting, all the instances are assigned certain weights and all instances from the training data set are used to generate the decision tree classifier in each round. The process of generating classifiers is repeated to certain previously declared number. As the algorithm repeats generating new classifiers from the same data, the previously classified result will be changed. The weights of instances that get misclassified are increased in every round of generating decision tree so that algorithm can give special attention to those instances having some extra weight while performing classification in next round [13]. Until the error of classification gets lower than 0.5, the process of generating classifier by initializing adjusted weight on the instances is repeated [3]. Like boosting, construction of cost-sensitive tree is another important feature implemented in C5.0 algorithm. In C4.5, all misclassified instances are treated with same importance level. But according to [27], if considered in real life, misclassifying the diagnosis of any disease can lead to serious result. The cost of misclassifying a patient, for example, diabetic as a non-diabetic can generate serious consequences. So it should have higher importance. This concept is implemented in C5.0 by assigning the weights to the instances. These weights are considered as the cost for misclassifying the instances present in the data set [27]. Also, C5.0 was incorporated with the ability to process data not only numeric or categorical but also data that represent dates, time, timestamps, etcetera [24]. Another significant feature in C5.0 algorithm was that it gives efficient results also on those data set which are imbalanced. C5.0 is a simple and efficient

classification algorithm. It can be used for almost any kind of data. C5.0 provides a robust and highly efficient model for prediction and classification tasks.

1.3.2 Random Forest

The Random forest algorithm was proposed by L. Breiman in 2001, as a general-purpose method for classification and regression task [50]. It is an ensemble learning method that generates many decision trees as classifiers and combine their result to give final prediction [36]. There are two methods applied to generate ensemble learning models described by [37] and they are sequential ensemble methods and parallel ensemble methods. In the sequential ensemble method, base learner's result is used as feedback to generate learner in next step using the same data, for example AdaBoost [13]. In parallel ensemble methods, the learning models are generated in parallel [37], where the base learner is build independent of each other. In random forest, a parallel ensemble learning method is used to generate the decision trees which are independent of each other. The process of generating a small subset of the final learning model, which are the decision tree, is also known as bagging. Bagging is a concept that is applied using two different techniques, namely bootstrap and aggregation [37]. Bootstrap is a process of generating separate data set from the existing one, by randomly picking the rows of data from the original data set. During this process, the same row can be picked and placed in new data set multiple times, which is called sampling with replacement [37]. And at the same time, some rows may not be picked at all, or they are left in the original data set untouched. And those data are said out-of-bag data or sometimes held-out data. Now, for the process of generating multiple trees as learners, a different subset of features and instances are chosen randomly from the bootstrap data set. For example, some trees may be created using only 2 features and some trees may be created using more than 2 features from multidimensional bootstrap data set. For selecting the best split attribute among the randomly selected subset of data from bootstrap data set, gini impurity is calculated, the method used in CART [15]. After generating the learners or classifiers of the tree with a randomly chosen subset of data set, those trees are tested for their efficiency using out-of-box data [37]. Out of bag data are those data which were left in the original data set without being included in the bootstrap data set. According to [7], there can be 36.8% of out-of-bag data in each base learner. The efficiency of base learner is estimated by feeding those out-of-bag data into the base learner. Now when all learners or classifiers are tested with the out-of-bag data set, by the process of aggregation, that class is chosen as the final predicted class, which has been predicted by the majority of base learners or classifiers in the model. At this point, the trees generated cast their vote for specifying any instances belonging to any class. Random forest is a popular algorithm for performing classification tasks on large and multidimensional data sets, because of features like being non-parametric, interpretable, efficient, and also having high prediction accuracy and low processing speed [49].

1.3.3 Gradient Boosting

Gradient Boosting is a machine learning algorithm which is based on ensemble approach of base learner. It was first introduced in 1999 by Stanford University Professor Jerome H. Friedman [82]. According to Jerome H. Friedman [16], Gradient Boosting is an algorithm that is used to produce competitive, highly robust, interpretable procedures for both regression and classification. Gradient boosting was derived from AdaBoost [51]. And it has been proved that gradient boosting produces most the efficient results for classification and regression tasks, due to which it has become one of the most popular algorithm in machine learning domain for classification and regression tasks on complex data. Gradient boosting algorithm follows the boosting approach while generating the final model by combining many base learners [30]. The results from each base learner is boosted by multiplying with fixed learning rate [16] and added to the previously predicted value by the previous base learner. The result from each base learner is added to generate the predicted value of the model. The base learners can be defined as the basic building blocks of the final model. Base learners are also called weak learner as their individual prediction is assumed to be more than random guess [16]. Gradient boosting model can be designed with different base learners. According to [40], there are three categories of base learners, namely Linear models, Smooth models [52], and Decision tree models. Since decision tree models are widely used as the base learner of gradient boosting, we will focus on the decision tree based learning model. As the gradient boosting is used for classification and regression task, it uses different algorithms for both kind of tasks. To understand the working of gradient boosting algorithm, we will analyze the algorithm proposed by Jerome H. Friedman for regression and classification tasks separately. Below is the Gradient Boosting algorithm for regression proposed by J.H. Friedman [16]. The algorithm is derived from [16].

- **Gradient Boosting Algorithm for Regression:**

For the convenience, the algorithm is divided into seven steps, step 1, step 2, step 3, step 4, step 5, step 6, and step 7. We will analyse the each step based on [16].

Input: Data $(x_i, y_i)_{i=1}^{n}$ and a differentiable loss function $L(y_i, F(x))$

The above expression defines about the data set with of 'x' variables and corresponding 'y' output variable or target class or predicted value of any data set ranging from 'i' to 'n' number. The loss function is used to measure the accuracy of predicted value. Though there are several loss function used in gradient boosting and according to [17], the popular ones are least-squares, least absolute deviation, Huber, and logistic binomial log-likelihood. Here, for regression task least squared method is used to determine the loss between observed and predicted value.

Step 1 : $F_0(X) = argmin_\rho \sum_{i=1}^{N} L(y_i, \rho)$

In the above equation, we defined function $F_0(X)$, which actually represents the first predicted values from the introduced training example. This function can be said as the function which maps each 'x' to their corresponding 'y'. Here $L(y_i, \rho)$, is a loss function. The $\sum$ means that we need to find the sum of residual by subtracting each observed value,y_i with each predicted value ρ. And argmin means that the minimum value for ρ, which helps to minimize the total sum of residuals. This minimized value is assigned to $F_0(X)$, which will be treated as leaf. Gradient boost will start inducing the decision tree based on this first minimum predicted value. This is just a single prediction, and it is treated as a single leaf, which will assign the same value to all other samples.

Step 2 : For $m = 1$ to M do :

As we know gradient boosting is an ensemble learning model, and it uses several base learners, here we need to declare the number of base learners that will be generated through out the process. Here 'm' refers to the individual tree generated in each iteration and 'M' is the number of trees or base learners that will be generated through out the process.

Step 3 : $\tilde{y_i} = -[\frac{\partial L(y_i, F(X_i))}{\partial F(X_i)}]_{F(X)=F_{m-1}(X), i=1,N}$

In this step, negative gradient is calculated which gives the best steepest-descent direction [16]. However this gradient is defined only at the data points X_{i1}^{N} and cannot be generalised to other X-values [16]. The gradient boost is named after this gradient. To find the steepest slope and the minimum intercept, the function minimum intercept the $-[\frac{\partial L(y_i, F(X_i))}{\partial F(X_i)}]_{F(X)=F_{m-1}(X), i=1,N}$ is partially differentiated with respect to slope and intercept of all samples in the data set.

Step 4 : $a_m = argmin_a \beta \sum_{i=1}^{N} [\tilde{y_i} - \beta h(X_i; a)]^2$

In this step, a regression tree is generated using the residuals value obtained in step 3, and using them as predicted values. β is the some constant defined as learning rate for generalization of value at each leaf of tree a_m. If β is not used then the new value obtained can over-fit to the training value.

Step 5 : $\rho_m = argmin_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(X_i) + \rho h(X_i; a_m))$

In this step, the output values of regression tree, a_m, generated at step 5 is determined. $argmin_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(X_i) + \rho h(X_i; a_m))$ expression says that for each leaf in new tree we compute an output value ρ, that minimizes the summation at each leaf.

Step 6 : $F_m(X) = F_{m-1}(X) + \rho_m h(X; a_m)$

At this step, we update the approximation $F_m(X)$. Here, $h(X, a_m)$ is added with the approximation generated by F_{m-1} in previous step. This illustrates the sequential ensemble learning process. The process from step 3 to step 6 repeats until the value of m does not equals to M.

Step 7 : end

Next, we will illustrate the gradient boosting algorithm for k-class classification. The algorithm is taken from [16]. This algorithm is implemented to develop a gradient-descent boosting algorithm for the k-class problem [16].

- **Gradient Boosting Algorithm for Classification**

Step 1: $F_{K0}(X) = 0, k = 1, K$

Step 2: For $m = 1\, to\, M$ do :

Step 3: $p_k(X) = \frac{exp(F_k(X))}{\sum_{l=1}^{k} exp(F_l(X)), k=1,K}$

Step 4: For $k = 1\, to\, K$, do:

Step 5: $\tilde{y}_{ik} =$ J - terminal node tree $(\tilde{y}_{ik}, X_i)_1^N$

Step 6: $\gamma_{jkm} = \frac{(k-1)\sum X_j \in R_{jkm} \tilde{y}_{ik}}{\sum X_j \in R_{jkm} |\tilde{y}_{ik}|(1-|\tilde{y}_{ik}|)}, j = 1, J$

Step 7: $F_{km}(X) = F_{k,m-1}(X) + \sum_{j=1}^{J} \gamma_{jkm} 1(X \in R_{jkm})$

Step 8: Stop Algorithm

In the above given algorithm for multi class classification, in step 1, $F_{K0}(X)$ denotes a function of "X" for "K" classes. In the step 2, number of iteration is declared as M. Initially the value of m = 1, represents that it is the first tree that will be induced. In step 3, negative binomial log-likelihood function is used as loss function. At step 4, probabilities of K classes are calculated at each iterations, m, to predict the corresponding current residuals for each class on the probability scale [16]. Each of these tree has J - terminal nodes, and their corresponding region is calculated by using $\tilde{y}_{ik}$ at step 5. Step 6 updates the values calculated for each region of tree. The final estimated value at step 7 is used for performing classification task.

From the above two algorithms, gradient boosting algorithms for regression and classification, we found that gradient boosting algorithms sequentially produce a prediction model in the form of linear combinations of base learners, i.e, m trees. The prediction accuracy of Gradient boosting algorithm is high due to the boosting technique applied at each base learner generated in every iteration of the process. According to [48] gradient boosting algorithm provides state-of-the-art prediction results on several challenging data set and it is regarded as one of the best classification algorithm we have today.

Generally, there are three types of gradient boosting algorithms categorised on the basis of data volume they are supplied during the calculation of gradients: a)Batch gradient boosting b)Mini Batch gradient boosting, and c) Stochastic gradient boosting.

In batch gradient boosting, the gradient of the cost function with respect to the parameters of whole training data is calculated in every iteration of tree formation [56]. This method requires more computational time and resources due this property. So, this method requires more time during the implementation phase in large data sets. Where as, in Stochastic gradient descent, this problem is addressed by only calculating the gradients of the cost function with respect to the parameters of certain sample of whole data set. But according to [56], Stochastic gradient boosting also can not converge towards global minima when learning rate is decreased. And, Mini-batch gradient descent calculates the gradient of the cost functions with respect to the parameters of "N" samples from whole data set. This helps to decrease the computational time and resources required for algorithm.

1.3.4 C5.0 Versus Random Forest versus Gradient Boosting

Here we will compare and contrast the algorithms C5.0, Random Forest and Gradient Boosting on the basis of their working principle and features. The work of compare and contrast of algorithms is provided in the table below. We have used following criteria to compare and contrast the algorithms:

- Proposed By : This parameter provides the information about the person who proposed the algorithm.
- Proposed For : This parameter provides the general information for which the algorithm was proposed.
- Working Principle : This parameter provides the information about the principle followed by algorithm to generate the classifier.
- Classifier Produced : This parameter provides the information about the type of classifiers produced by each algorithm.
- Ensemble Approach : This parameter provides the information about the ensemble technique used by the algorithms to generate final predicting model.
- Data Input Strategy : This parameter provides the information about the strategy used to fed data to algorithm.
- Root Node : This parameter provides the information about the technique used to select the root node for splitting the attributes in data set.
- Efficiency : This parameter gives the information about the general efficiency of the model (processing time and prediction) generated by using the algorithms.

Based on the above discussed criteria, we have listed the differences between C5.0, Random Forest, and Gradient Boosting in table 1 below.

C5.0 versus Random Forest versus Gradient Boosting

Algorithms:	C5.0	Random Forest	Gradient Boosting
Proposed By:	John Ross Quinlan	Leo Breiman	Jerome H. Friedman
Proposed For:	C5.0 is proposed for performing classification task.	Random Forest is proposed for classification and regression task.	Gradient Boosting is Proposed for classification and regression task.
Working Principle:	C5.0 is based on the information theory in which it calculates the information gain ratio of the data and performs the classification task. It is an iterative model, which means this algorithm grows the tree sequentially.	In Random Forest, data are sampled randomly. Gini index is calculated from the set of randomly selected data while generating a tree. And several trees are produced. Each one with its predicted result. The result which is predicted by most of the trees is considered as the final one.	In Gradient boosting gradients are calculated such that the difference between the true value and the predicted results become minimum.
Classifier Produced:	C5.0 generates decision tree classifiers.	Random Forest generates the decision tree based classifiers and regressors.	Gradient Boosting can be used to generate tree-based, and linear model based classifiers and regressor.
Ensemble Approach:	C5.0 follows the boosting technique for generating decision trees one after another in sequence.	Random Forest follows bagging technique for generating the decision trees.	Gradient Boosting follows the sequential ensemble approach and uses both boosting and bagging technique to generate the final model.

Table 1.1 continued from previous page

Algorithms:	C5.0	Random Forest	Gradient Boosting
Data Input Strategy:	In C5.0, the whole training data is supplied to the algorithm at the beginning.	Random Forest supports sampling technique. So, it trains each trees with randomly selected certain por - tion of data from training data set.	Gradient boosting does not support bootstrap model, so all data are provided to algorithm at once. However stochastic gradient boosting supports bootstraping. So, subsampling feature is available in stochastic gradient boosting.
Node splitting criteria:	C5.0 calculates the information gain ratio of attributes present in data set to decide the best splitting attribute at each node.	Random forest calculates the gini index of features, to decide the best splitting attributes present in the data set.	Gradient boosting uses the least square method to find the attribute having minimum squared error while selecting the nodes of the decision tree.
Efficiency:	C5.0 cannot handle big data set. Its classification accuracy is lower than that of Random Forest and Gradient Boosting Machines.	Random Forest is considered to be more efficient than C5.0 in terms of prediction accuracy. But it is less accurate than Gradient Boosting Machine.	Gradient Boosting is the most advance algorithm in terms of handling big data, training speed, and performance.

Table 1.1: Table summarizing the differences between C5.0, Random Forest, and Gradient Boosting.

1.4 XGBoost

XGBoost is an open-source package implementing a gradient boosting principle for the classification, regression, and ranking tasks in machine learning. It was proposed by Chen and Gustrin [54] in 2016. XGBoost stands for extreme gradient boosting algorithm [53] which is an extended version of gradient boosting algorithm and it is based on the same principle as gradient boosting algorithm for creating base learners and developing them into assembled model for classification or regression. XGBoost uses a sequential ensemble approach to generate an additive classifier which is also called boosting technique and it is used for supervised machine learning problems. The XGBoost's learning function can be represented as the regularized objective function. This function is the minimized form of functions used to create a model using XGBoost. The objective function is defined to measure how well the model performed on the prediction task. The regularized objective function [54] is given as:

$$L(\phi) \; = \; \sum_i l(\tilde{y}_i, y_i) + \sum_k \Omega(f_k)$$

Where $l(\tilde{y}_i, y_i)$ represents the loss function. Though there are many loss functions like the hinge loss, the logistic loss, the cross entropy loss, and exponential loss [80], that can be used in XGBoost to measure the residuals, but usually mean squared error and logistic loss functions are used [45]. And $\sum_k \Omega(f_k)$ represents the regularization term that helps to prevent the over-fitting in training data set. By this function, the predicted value at each leaf is minimized by some scale, which creates the opportunity for future models in ensemble to predict the correct value. The goal of the learning model is to minimize the loss and hence increase the efficiency of the model.

XGBoost is considered to be much more efficient than gradient boosting machines, in terms of processing speed and prediction accuracy. According to [44] it is ten times faster than gradient boosting due to the feature scalability. There are several other features [54] in XGBoost which provides support for high scalability. Here we have discussed some features provided by [54] in short. XGBoost has shrinkage and column sub-sampling features, which helps to prevent over-fitting. Shrinkage scales newly added weights, at leaf after each step of tree boosting, by some factor η [54]. When the final weight at a leaf, which is the predicted weight of each tree, is reduced, the residual increases and this leads new trees to improve the future prediction. In column sub-sampling, subsets of features are randomly picked so that algorithms don't need to consider all features present on the data set while training the model. And column sub-sampling also contributes on decreasing the training time required. For finding the best split candidate in a data set, XGBoost implements an exact greedy algorithm and approximate algorithm. The exact greedy algorithm is used when a data set is located as a whole inside a single location. And the approximate algorithm is used when computation for the best splitting feature is needed to perform in distributed locations. Also, another important feature of XGBoost is that it is aware of sparse data type [54]. XGBoost is best suitable for high volume complex data with multidimensional feature vector. And in real life these type of data set have more sparse data. When XGBoost is fed with sparse data like

data sets having multiple missing attributes, frequent zero values, a separate path is declared from the root node of the tree to pass those instances in relevant direction. Lastly, the most important feature of XgBoost due to which its performance speed is also high is that it supports out-of-core computations. In this feature, those data which cannot be fitted into computer primary memory at once are divided into blocks and are stored in the different locations of secondary memory locations. When they are needed during the process of execution, they are loaded from the secondary memory locations. According to [54], XGBoost package is widely popular in machine learning and data mining applications for classification and regression tasks since it provides a state-of-the-art solution using a minimal amount of resources.

1.5 LightGBM

LightGBM is a machine learning software package for implementing the gradient boosting algorithm for classification, regression and ranking tasks. It was proposed by Guolin Ke et al. [60]. It also works on the same principle of gradient boosting, in which XGBoost works. It is also assigned a loss function whose gradient is calculated with respect to the features to determine the new prediction.

It is the improved version of XGBoost in terms of speed and accuracy [60]. When a large volume of data is needed to handle with multiple features, in XGBoost to determine the best split point among all feature, every data present in the particular features are needed to be scanned which is very time-consuming [60]. To eliminate this problem Guolin Ke et al. proposed two novel techniques as Gradient-based one-side sampling (GOSS) and Exclusive Feature Bundling (EFB) for LightGBM. In GOSS for calculating the information gain to find the best split point, only those data points whose gradients are higher are considered. The other data with lower gradients are randomly sampled and they are multiplied with a constant value so that data having lower gradients can influence in finding the best split point. This approach results in higher speed in the process of finding best the split point. Another improvement made by Guolin Ke et al. for LightGBM is EFB. In high dimensional data set, there can be huge sparsity. Features with high sparsity can only increase the resource requirements while training. To eliminate this problem, if such features are excluded from data set, then it may also influence during computation of the best split point. In LightGBM, this problem is tackled by implementing EFB, in which mutually exclusive features are bundled together [60] to reduce the number of features, which adds to the speed and efficiency of the algorithm. Due to these features in LightGBM, according to [60], the overall training speed of LightGBM is 20 times faster than other conventional gradient boosting decision trees. LightGBM is also a new gradient boosting based algorithm which is gaining huge popularity in the machine learning community for classification and regression tasks.

1.6 CatBoost

CatBoost is an open-source machine learning library implemented with the gradient boosting principle for the classification, regression and ranking task. CatBoost or Categorical Boosting is proposed by Dorogush et al. [67] for classification, regression and ranking task, with our user requirement to change categorical features into numerical during data pre-processing by using the encoding technique. CatBoost can be used with data sets having both numeric and categorical features. According to Dorogush et al. [67], to deal with the categorical value CatBoost uses a strategy, which helps to reduce over-fitting. In this strategy in CatBoost, features are permuted or shuffled randomly. And for each example, an average label value for example with the same category value, placed before the given one in the shuffled list, is calculated. During the training of model, each example is replaced with the average label value. Since it follows ensemble approach, while generating a new decision tree, permutation is performed again on the data set and whole process of of calculating the numerical value of categorical example is repeated. According to [67], another powerful and effective method implemented in CatBoost for converting categorical features to numerical is by calculating the frequency of examples present in the data set and replacing it with counted value in training process. This feature also enhances the performance of the algorithm.

Another important feature of CatBoost is Feature Combination [67]. In Feature combination, CatBoost generates more powerful features by combining the features available in the data set. Since all present features can not be combined with each other, it will be computation overhead, CatBoost combines the feature in greedy way. That means CatBoost does not combine the feature for the first split in the tree. For the next split, CatBoost combines all combinations and categorical features present in current tree with all categorical features in data set [67]. According to [67], another improvement made in CatBoost which made it better than other gradient boosting decision trees like XGBoost and LightGBM is "Fighting Gradient Bias". In CatBoost, unlike XGBoost and LightGBM, for building the tree structure, modified version of gradient based decision trees is used and for setting the leaf values of built tree, traditional gradient boosting decision trees is used. It is explained in brief in [67]. Due to these features, Dorogush et al. claims that CatBoost outperforms the existing state-of-the-art libraries like XGBoost and LightGBM in terms of computational speed and accuracy, in their work published in [67].

1.7 Hyper-parameters

In machine learning algorithms, hyper-parameter is a parameter whose value is used to control the learning process [89]. In other word, hyperparameter determines the architecture of the machine learning model. In case of decision tree based model, parameters like number of leaves, depth of tree, loss function used to measure the efficiency of prediction, and et cetera determines the model architecture. For example, during the process of generating decision tree, if it becomes too wide or too deep, then there can be chances of over fitting tree on training instances. To minimize such challenges, size of the tree can be controlled by using the pruning method [11]. Here, size of tree, number of leaves, depth of tree and pruning factor can be termed as parameters of decision tree. Machine learning algorithms like C5.0, Random Forest and Gradient Boosting have a large set of hyperparameter from which best set of parameters are needed to be chosen for the high performance of algorithm [69][88][38]. Some of the hyper parameters used in C5.0 are maximum tree depth, number of leaf. Similarly some hyper parameters used in Random Forest are booststrap model, split criteria, maximum features used to train a tree, minimum number of leaf and like wise, some hyper parameters used in Gradient Boosting are number of trees or estimator in the model, the learning rate of the model, the maximum tree depth, the minimum tree weight and et cetera. We will discuss about the some important hyperparameters of xgboost, LightGBM and CatBoost, which we have selected for tuning our models. These parameters are given below.

1. **Learning Rate (learning_rate, eta, shrinkage_rate)**

   As we know xgboost, LightGBM and CatBoost are implemented with gradient boosting algorithm [54], so they uses sequential ensemble method to generate the final prediction. In sequential ensemble method, many base learners or trees are generated. The result or prediction given by each tree for some specific class is multiplied with some fixed rate and added to another tree's result. The rate by which the result of each tree is multiplied is known as the learning rate. Learning rate is used to specify the steps at which gradients are calculated. It can also be understood as the ratio or the interval after which the algorithm checks for the total loss obtained after making prediction at that point. Learning rate for xgboost, LightGBM and CatBoost is selected between 0 and 1. Generally, the default learning rate of xgboost and LightGBM is 0.1, and default learning rate of CatBoost is 0.03 except for multiclass, logloss and root mean squared error loss function. If the learning rate is set to be low, it increases the training period. So, to decrease the time, we can increase the learning rate and decrease the number of trees.

2. **Number of Trees (n_estimators, num_iterations, iterations)**

   Gradient boosting algorithms generate a number of sequential base learners, and multiply the output of each learners with fixed learning rate and gives the output as predicted value. If the gradient boosting algorithm follows tree based module, then those base learners are called trees. In other words, number

of trees simply denotes the number of boosting iterations or number of trees build during the whole boosting process. In xgboost, these number of trees are represented as 'n_estimators'. The default number of trees generated in xgboost is 100. Generally, We can select the number of trees in xgboost from 100 to 5000. But in LightGBM there is no limit in maximum number of trees. Similarly, in CatBoost, the default number of trees is 1000 and it also does not have any maximum limit for this parameter.

3. **Maximum depth (max_depth, depth)**

   The depth of any decision tree can be defined as the distance from root node to end leaves. The distance basically should be measured in terms of levels from where the decision branches split up. In other words, the total number of splitting nodes from root node of to the final classification leaf of the decision tree. For example, lets consider that there are 3 predictors (X, Y, Z) and one target feature (P) having binary target class as 0 and 1. Now lets say we found X to be the best attribute to be a root node and remaining Y and Z are split from the node X. Now again, Y and Z will be split into two leaf node predicting the class 0 or 1 as their final outcome. Here the total length or depth, from root node to leaf will be 2. So, it can be said that the maximum depth of tree can be one less than the total number of features. But practically it is not best practice to increase the depth of tree, which increases the complexities and risk of over-fitting. So, the maximum depth needs to be find out by cross validation method or tuning process. In xgboost, maximum depth is represented as 'max_depth'. In xgboost and CatBoost, the default value of depth is 6.

   In LightGBM tree grows leaf-wise. This means that instead of growing tree in layer-wise like in XGBoost and CatBoost, tree grows from the leaf of decision trees. So depth is determined by the number of leaf as well. In LightGBM the default value of depth is -1. Here -1 denotes that tree can grow to any depth according to data it is being trained with, if it is not controlled by assigning some integer value to its parameter max_depth [61]. The more tree grows depth-wise, there will be more chances of overfitting. So, this parameter is tuned during the training process in our experiment.

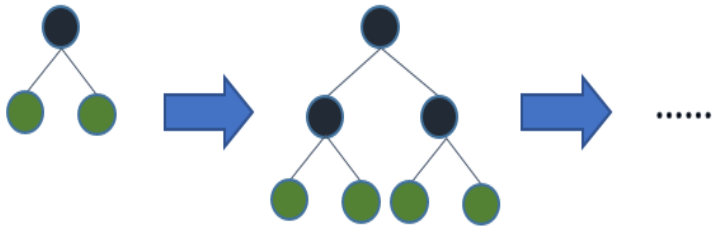


Figure 1.1: An image of a level-wise tree growth

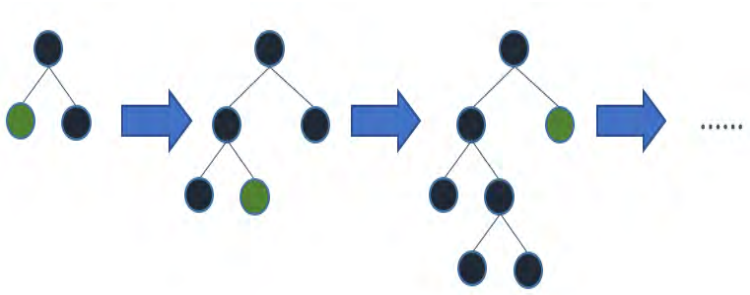


Figure 1.2: An image of a leaf-wise tree growth [62]

4. **Minimum child weight**

   While forming decision tree, after root node, many different child nodes are formed. After child nodes, leaf nodes are formed. If no threshold is set for further splitting of child node, the tree will grow until all target attributes become leaf node. This may increase the complexity of tree and results in overfitting. So we can set some threshold to further split the node, if it has minimum number of instances equal to the threshold set. This is called to be minimum child weight.

5. **Gamma**

   Gamma is a regularization parameter that is used in xgboost [54] for determining whether to prune the particular leaves from tree or not. In xgboost, for pruning, a random value for gamma (γ) is set. This value is subtracted with the gain, which is previously calculated, to find out the best split. If the difference turns negative, then that particular leaves is pruned, otherwise, it is left as it is. Gamma helps to control the overfitting of tree by pruning and making tree less sensitive to new instance in data set.

6. **Subsample and Colsample**

   In normal gradient boosting algorithm,the whole data set needs to be fetched at once. This increases the training time and also decreases the efficiency of model by over generalizing the model. According to [18], this can be improved by inserting only some random chunk of data from whole data set in each iteration at a time. The process of dividing the data set into smaller sets or chunks is called subsampling in xgboost. Similarly, colsample means the randomly picked set of column from the subsample while generating a particular base learner (tree).

   We will tune both subsample and colsample in the range of 0.1 to 1. Here, the float number declared for the range, represents the ratio of randomly selected set of data to the whole data set assuming 1 for the whole data set.

7. **Regularization parameters (α and λ)**

Alpha (α) and Lambda (λ) are another set of regularization parameters after gamma. These parameters help to control the overfitting problem in xgboost model. If the data set that is used to train the model has multiple features, then those features can highly influence the model while training which can lead to overfitting. To avoid this problem, if some of the features are dropped, then some vital information may get lost which can results to under fitting. In this case, we can decrease the weights by penalizing the multiple features by decreasing their value in same proportion. This is called regularization. For decreasing the weights of node, xgboost uses λ to divide the square of the sum of residuals ending in a node for finding the weight of that node. score or weight of node $= \frac{(sum\ of\ residuals)^2}{number\ of\ residuals + \lambda}$

The total gain or weight is the sum of residuals divided by their number by adding with λ as shown in the above equation for every node, in root node, left node and right node. λ in the equation helps to minimise the weight or score of node and hence decreases the total gain. When gain is decreased, the pruning decision at each node become less complex for algorithm. The equation for calculating gain is given as: Gain = weight of left node + weight of right node - weight of root node.

Alpha (α) and Lambda (λ) are also called L1 regularization and L2 regularization terms respectively. L1 and L2 both penalises the weights in xgboost.

8. **Number of leaves (num_leaves)**

   LightGBM adds nodes to trees based on the gain from adding that node, regardless of depth. This parameter is responsible for deciding the number of nodes in a tree. The range of this parameter is from 1 to ∞. Its default value in gradient based LightGBM is 31.

9. **Minimum gain to split (min_gain_to_split)**

   When adding a new tree node, LightGBM chooses the split point that has the largest gain. Gain is basically the reduction in training loss that results from adding a split point. By default, LightGBM sets min_gain_to_split to 0.0, which means "there is no improvement that is too small". However, in practice you might find that very small improvements in the training loss don't have a meaningful impact on the generalization error of the model. Increase min_gain_to_split to reduce training time.

10. **Minimum data in a leaf (min_data_in_leaf)**

    Minimum number of observations that must fall into a tree node for it to be added. Its default value is 20 in LightGBM.

11. **Random strength (random_strength)**

    Random strength is the amount of randomness to use for scoring splits when the tree structure is selected. This parameter is used to control the overfitting of CatBoost model. The value of this parameter is used when selecting splits.

On every iteration each possible split gets a score (for example, the score indicates how much adding this split will improve the loss function for the training dataset). The split with the highest score is selected. The scores have no randomness. A normally distributed random variable is added to the score of the feature. It has a zero mean and a variance that decreases during the training. The value of this parameter is the multiplier of the variance.

12. **Bagging temperature (bagging_temperature)**

    This parameter defines the settings of the Bayesian bootstrap to assign random weights to objects. It is used by default in classification and regression modes in CatBoost.

13. **Border Count (border_count)**

    This parameter determines the number of splits for numerical features in CatBoost. This helps to control the overfitting of model.

14. **Tree growing policy** By default, CatBoost uses symmetric trees, which are built if the growing policy is set to Symmetric Tree. Such trees are built level by level until the specified depth is reached. On each iteration, all leaves from the last tree level are split with the same condition. The resulting tree structure is always symmetric. Symmetric trees have a very good prediction speed (roughly 10 times faster than non-symmetric trees) and give better quality in many cases. However, in some cases, other tree growing strategies can give better results than growing symmetric trees.

1.8 Hyperparameters Search Method

The performance of machine learning algorithms depends on the set of hyper parameters used [63]. So, selection of appropriate hyper parameter values is a crucial task during the learning model generation. The process of finding the best set of hyper parameters that produces the best result is also called hyper parameter optimization. So, for finding the best value of any hyperparameters requires multiple trial with different values. But for performing multiple trial, it requires lot of human effort and time. To overcome this inefficient approach of selecting hyper parameters, methods like grid search, random search, and bayesian optimization are used [65]. We will discuss about each in brief below.

1.8.1 Grid Search

In grid search method [78], each and every hyperparameters declared with values are tested in combining each other. That means for example M and N are two hyper parameters. M is declared with 3 different values and N is declared with 4 different values. Now In grid search all these values are combined turn by turn and the train and test is performed at least for 12 times. This process gives the

best set of hyper parameters but it requires a lot of time since after combining the hyperparameters, many different sets are generated. This method is costly in terms of computational resources and time, but it checks each and every values from search space. So, for finding the best value of hyperparameters, we followed this method in our experiment.

1.8.2 Random Search

In random search method, [34] explains that the time for selecting best combinations of hyper parameter is less due to its character of randomly selecting the values from hyper parameter declared space. Unlike grid search method, random search randomly selects the combination of hyperparameters and perform operation on it. According to [34] random search yields the same results like grid search in much less time, but sometime it can miss the value from declared space.

1.8.3 Bayesian Optimization

Bayesian optimization [12] [47] is also used for finding the best set of hyper parameters. This technique creates the probabilistic model of the function being optimized and then leverages this model to decide where next to evaluate the function [65]. In this method information about the past output of combinations of hyper parameters can be incorporated which increases the computation speed of algorithm, since it don't need to combine those hyper parameters that have already been combined.

1.9 Automated Machine Learning (AutoML)

In the automated machine learning (AutoML) approach, feature engineering, base learner selection and the hyperparameter optimization of base learner are done automatically and this approach has been proven to be efficient on those models which require a wide range of hyperparameter [79]. Unlike in AutoML, algorithms like Random Forest, XGBoost, LightGBM, and CatBoost, require human supervision for data processing, feature engineering, selecting base model, optimizing hyperparameters and prediction result analysis of the model. For optimizing hyperparameters, mostly Grid search and Random search are used, which requires human effort for declaring hyperparameter values in search space and selecting the best result from their output. Some common automatic machine learning tools which are in practice are Auto Weka [79], Auto-Sklearn [79], H2o [84], Hyperopt-Sklearn [42], TPOT (The Tree-Based Pipeline Optimization Tool) [57], AzureML [58], and et cetera.

So, Automatic hyperparameter optimization is useful in reducing human effort to tune the hyperparameters of the model, which enhances the performance of machine learning algorithms, and also hyperparameter optimization is more reproducible than a manual search [79]. Also, in automated machine learning, it is not required to explicitly perform data preparation tasks like one-hot-encoding, and filling missing values. These sorts of tasks are automatically handled by an algorithm implementing

series of steps for the task through pipeline Another important benefit of the automated machine learning process is that it can be used by anyone from machine learning experts to naive users. Automated machine learning makes it possible to generate the best machine learning model with little knowledge about the domain.

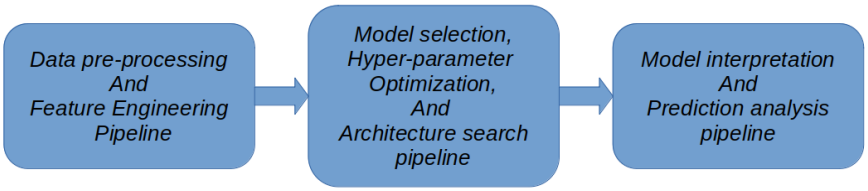


Figure 1.3: The common automatic machine learning pipeline [81]

According to [81] most automatic machine learning techniques are implemented through a pipeline having different components as shown in figure 3. In each pipeline series of actions are done. In the figure, data pre-processing, and feature engineering is implemented in the first component. Task such as identifying data types, data size, target attribute, imputing missing values, calculating the relevancy of any feature, addition or removal of features, et cetera are done. Based on the information gathered from data pre-processing and feature engineering component, tasks such as model selection, hyperparameter optimization, are done in the second component. For hyperparameter optimization different automatic machine learning framework follows different technique. For example, Auto-Weka uses sequential model-based algorithm configuration (SMAC) [33], Auto-sklearn utilizes SMAC3, H2O-AutoML utilizes random search for selecting hyperparameters from search space. After data pre-processing, feature engineering, model selection, hyperparameter optimization and modelling, the performance results are visualized using different visualization methods. For example, H2O visualizes its results through a graphical user interface in a dashboard.

Chapter 2

Analysis

2.1 Research Question

The topic of our study was "Comparative Analysis Of Classification Accuracy For XGBoost, LightGBM, CatBoost, H2O, and Classifium."

For getting the optimal accuracy from manually tuned XGBoost, LightGBM and CatBoost we implemented hyperparameter tuning pipeline, which we will discuss in the Implementation chapter of this document. For automatic machine learning algorithm H2O and Classifium, they are provided with their built-in pipeline for data pre-processing and hyperparameter tuning. During our study we were interested to find out how well the automatic hyperparameter tuning pipeline of Classifium performs when compared with the manually tuned XGBoost, LightGBM, and CatBoost in terms of classification accuracy. And also, how well it can perform compared with another automatic machine learning algorithm like H2O provided with its own pipeline for tuning hyperparameters of algorithm.

Based on the scope of our project, the following research question is formulated:

- **RQ1:** Can an automated machine learning approach like Classifium generates the classification results comparable with the results generated by the manually and carefully tuned gradient boosting machine learning libraries like xgboost, LightGBM, CatBoost and automatic machine learning library H2O?

2.2 Related Work

For making feature extraction, feature engineering, model building and tuning, automated machine learning approaches provide a platform, where these tasks are automatically done by the algorithm itself through pipelines. It provides ease for novice users to work with machine learning problems and in that way democratize the machine learning solutions for everyone. Automated machine learning has been in practice and much research work is going on in this field. Here we have briefly discussed some automated machine learning platforms that are in practice.

Auto Weka [79] is an automated machine learning library in which machine learning model selection and hyperparameter optimization of those models is done automatically. It uses the Combined Algorithm Selection and Hyperparameter optimization (CASH) [79] algorithm for model selection, hyperparameter optimization and feature selection. In Auto Weka, [79] authors implemented the Bayesian optimization technique to automatically select base learners and hyperparameters already available in the library of WEKA [26] which is a data mining software package. The base learner algorithms in auto weka are Bayes net, Decision stumps, Decision table, J48, Linear regression M5P and multilayer perceptron. But it does not have gradient boosting based algorithms as base learners.
H2O [84] is also an auto-machine learning tool which is used to select the best machine learning algorithms like linear regression, gradient boosting, random forest, deep neural network and et cetera and tune their hyperparameters automatically for given data. Along with other algorithms, H2O has incorporated xgboost as its base learner, but not LightGBM and CatBoost.

Auto-Sklearn [79]is another AutoMl method that searches for the best combinations of algorithms and hyperparameters from the search space. According to [79], this method is better than Auto Weka and Hyperopt-Sklearn in terms that in this method results of tuning performed on similar data in past can be incorporated.
But Auto-Sklearn does not have xgboost, LightGBM and CatBoost as its base learners.

Auto hyperparameters tuning algorithms are essential components of any automated machine learning library. Addressing this requirement, for finding the best configurations of the algorithms and hyperparameters of Scikit-learn machine learning library, Hyperopt-Sklearn [42] is proposed. It automatically searches the algorithms with their hyperparameters in the search space and combines them to represent a single hyperparameter optimization problem [79].

For auto-tuning the hyperparameters [77] proposed a simple adaptive random search strategy. Authors compared this strategy with other optimization algorithms like L-SHADE and random search algorithm by running 11 different data set on XGBoost algorithm and found that this strategy equally optimized the hyper parameters of algorithm like L-SHADE and random search algorithm does.
Autoxgboost [73] is also an automl method that follows the series of steps to perform auto-tuning of hyperparameters. This method or pipeline is implemented to tune hyperparameters, threshold optimization and encoding of categorical features.
Likewise for tuning the hyperparameter of XGBoost, [70] used a framework based on Bayesian optimization principle, called Hyperopt.

Till now we have discussed about some of the research work done in the field of automatic machine learning algorithm and automatic hyperparameter searching and optimization algorithms. Since, gradient boosting decision trees are more

effective in solving machine learning problems [22], [32], our study is more focused on comparing manually tuned gradient boosting decision tree-based libraries, XGBoost, LightGBM and CatBoost, with the automated machine learning methods like H2O and Classifium. XGBoost, LightGBM and CatBoost are the latest state of the art algorithms implementing gradient boosting techniques.

2.3 Related Comparative Work

To our best knowledge, very less study has been done on comparative analysis between new gradient boosting libraries, XGBoost, LightGBM and CatBoost for their performance. Here we are discussing few research works which have been done comparing XGBoost, LightGBM and CatBoost and automated machine learning library in terms of classification accuracy.

In "CatBoost : Gradient boosting with categorical feature support" [67], Dorogush et al., performed a comparative analysis between the latest implementation of gradient boosting decision tree algorithms, CatBoost versus XGBoost and LightGBM. For the comparison task, [67] had taken 8 different public data set from the UCI repository. Those data set had a sufficient amount of categorical attributes and CatBoost was purposed to handle the categorical attributes, without converting them to numerical using an encoding technique. While comparing the performance between these algorithms, they have developed two sets of models for each algorithm. One with hyperparameters tuned and another with default hyperparameters set. During this work, they have not explained about the implementation of hyperparameter optimization pipeline in [67].
From comparison results, [67] claims that CatBoost performed better than XGBoost and LightGBM in terms of lower log loss error.

Similarly, in "A comparative analysis of gradient boosting algorithms" [90] has performed extensive comparison tests among XGBoost, LightGBM and CatBoost to analyze their performance accuracy using 28 different data sets from the UCI repository. From this comparative study, they conclude that CatBoost outperformed XGBoost and LightGBM in terms of accuracy. LightGBM outperformed XGBoost and CatBoost in terms of training time. And XGBoost performance accuracy and training time was found to be in between of CatBoost and LightGBM result.
Also, in [90] authors mention that the hyperparameters of comparative models were tuned using the grid search method. But they have not mentioned the implementation of hyperparameter tuning pipeline.

In another similar work by Essam AL Daoud, "Comparison between XGBoost, LightGBM and CatBoost using home credit data set" [74], the author concludes that LightGBM outperformed XGBoost and CatBoost in terms of accuracy and training time. He also argues that the results obtained from his work can not be generalised

for other data sets.

In "Comparative analysis of gradient boosting algorithm for landslide susceptibility mapping", [85] performed a comparative study on four different gradient boosting algorithms, Gradient Boosting Machine, XGBoost, LightGBM and CatBoost for modelling landslide susceptibility. This study [85] concludes that CatBoost model had the highest prediction accuracy. After CatBoost, XGBoost, LightGBM and GBM were ranked from top to bottom according to their performance accuracy. During this work [85] mentioned that he has applied the grid search method for tuning parameter of GBM and XGBoost. But parameters of LightGBM and CatBoost were tuned by trial and error method.

In [83] the author evaluated CatBoost and XGBoost on the task of Medicare fraud detection and report performance in terms of running time and Area Under the Receiver Operating Characteristic Curve (AUC). And concludes that with highly imbalanced and numerical data set CatBoost and XGBoost performed nearly the same in terms of AUC.

Similar to latest gradient boosting algorithms, there has been very few comparative analysis between different automatic machine learning tools. Here we have discussed only one analysis study conducted to compare different automatic machine learning algorithms.
In "Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools", [81], a comparative analysis is conducted to evaluate the different automatic machine learning approaches like Auto-WEKA [41], auto-sklearn [76], TPOT [57], and H2O AutoML [84] with 300 different data sets. This study [81] has concluded that H2O AutoML performed slightly better in terms of accuracy for binary classification and regression than other automl tools.

To our best knowledge, there has not been any comparative study done between manually tuned XGBoost, CatBoost and LightGBM, implemented with hyperparameter optimization pipeline, and AutoML H2O with automatic machine learning algorithm Classifium.
Our research will be focused on fulfilling this gap.

Chapter 3

Planning

3.1 Methodology

In this section, we present the methodology we followed to carry out our experimentation in details.
According to the objective of our study, we first manually tuned xgboost, LightGBM and CatBoost following the hyperparameter tuning pipeline and using the gridsearchcv() function of scikit learn library with seven different data sets. After obtaining the results from manually tuned algorithms, we then ran Classifium, which is an automated machine learning algorithm used for classification tasks, with the same training and testing data sets and obtained the result. Again, we ran another automated machine learning algorithm called H2O with the same data sets. Then we tabulated the results of all algorithms and compared the findings.
Throughout the process of manual tuning xgboost, LightGBM and CatBoost, we followed a series of steps where different kinds of hyperparameters were tuned. We call these steps as pipeline. The same pipeline was used by [55] on their work and found impressive results. For finding the best value of hyperparameters from declared search space, we used the gridsearchcv() function of the scikit learn library. We declared the values in the grid. When the gridsearchcv() function returned the value, we again searched for best fit declaring a new grid containing the last best value and its closest lower and upper value.
We took seven different data sets from the HIOF repository maintained by Prof. Roland Olsson for building classification modules. These data sets are explored in detail in a later section. Here, we will discuss the hyperparameter pipeline that we followed to tune xgboost, LightGBM and CatBoost.

3.2 Hyperparameter optimization pipeline for Xgboost

At the beginning, we declared a xgboost classifier function with hyperparameters having default values. The detail about the function and its hyperparameters are given in the appendix of this document. The default values were adjusted throughout the tuning process.
Series of steps or pipeline we followed to tune xgboost:

- **Step 1 :** Tune the number of iterations
  The number of iteration was tuned along with the learning rate.
  The function was updated with the new value of estimators and learning rate and proceeded to step 2.

- **Step 2 :** Tune max_depth and min_child_weight
  In the second step of pipeline, we tuned max_depth and min_child_weight. The best-fitting value was updated in the function. And proceeded to step 3.

- **Step 3 :** Tune Gamma
  In the third step, the value of gamma was tuned and the best value was updated in the function.

- **Step 4 :** Re-calibrate the number of iterations
  After updating the xgboost classification function with the best value of hyperparameters obtained so far from steps 1, 2 and 3, we again tuned the number of iterations.
  After tuning the number of iterations from this step, we updated the function once again with a new value for the number of iterations and moved to step 5.

- **Step 5 :** Tune regularization parameters
  At this step, we tuned regularization parameters (l1 and l2 parameters). And updated the function with a new value of regularization parameters and moved to step 6.

- **Step 6 :** Reducing learning rate
  This was the last step of our pipeline for tuning hyperparameters of xgboost. When all hyperparameters were tuned up to step 5, we again tuned the learning rate. At this time, we checked whether it best fit will lower value than present value or not.

3.3 Hyperparameter optimization pipeline for LightGBM

At the beginning, we declared LightGBM classifier function with hyperparameters having default values. For tuning the LightGBM we followed the similar pipeline which we had followed for tuning xgboost. We will discuss the series of steps or the pipeline in this section that we have proposed here based on the documentation and the related work.

- **Step 1 :**Tune number of estimators and learning rate
  The number of estimators were tuned along with the learning rate.
  The classification function was updated with the new value of estimators and learning rate and proceeded to step 2.

- **Step 2 :** Tune maximum depth and minimum gain to split
  After tuning the learning rate and the number of estimators, we tuned maximum depth and minimum gain to split.
  The best value of these two hyperparameters obtained at step 2 is updated in the classification function and moved to step 3.

- **Step 3 :** Tune minimum data in leaves and number of leaves
  At step 3, we tuned minimum data in leaves and number of leaves.
  After tuning these two parameters, the classification function was updated with the best value of these two hyperparameters.

- **Step 4 :** Re-calibrate the number of estimators
  After step 3, we re-tuned the number of estimators of classification models which were trained with big data sets.

- **Step 5 :** Tune bagging fraction and feature fraction
  After step 4, we tuned the bagging fraction and feature fraction of our classification model with the updated classification function.

- **Step 6 :** Tune regularization parameters (lambda_l1 and lambda_l2). After step5 we tuned regularization parameters. This was the last step of our pipeline for tuning the LightGBM.

3.4 Hyperparameter optimization pipeline for CatBoost

From the official documentation of CatBoost, https://CatBoost.ai/docs/concepts/parameter-tuning.html, we followed the guidelines for tuning CatBoost classification models. It is claimed on their documentation that CatBoost can be tuned to an optimum level with few sets of hyperparameters. Its default value for other hyperparameters is set such a way that they applicable for any data sets. So we proposed the series of steps as pipeline for tuning the hyperparameters of CatBoost as described below.

- **Step 1 :**Tune number of estimators and learning rate
  The number of estimators was tuned along with the learning rate.
  The classification function was updated with the new value of estimators and learning rate and proceeded to step 2.

- **Step 2 :** Tune depth of the tree
  After tuning the learning rate and the number of estimators, we tuned the maximum depth of the tree.
  The best value of depth hyperparameter obtained at step 2 is updated in the classification function and moved to step 3.

- **Step 3 :** Tune l2 leaf regularization function
  At step 3, we tunedthe regularization function named as l2_leaf_reg.
  After tuning these parameters, the classification function was updated with the best value of these parameters.

- **Step 4 :** Tune random strength
  After step 3, we tuned the random strength parameter of CatBoost. And after obtaining the best value of this hyperparameter, we updated the classification function and moved to step 5.

- **Step 5 :** Tune border count
  In this step, we tuned the border count parameter. Generally, the default value of this parameter gives the best result. But we chose this parameter because of its properties; since its value could influence the training time.

- **Step 6 :** Tune bagging temperature
  After step5 we tuned bagging temperature parameters.

- **Step 7 :** Re-calibrate iterations and learning rate
  After tuning all hyperparameters from step 1 to step 6, we re-calibrate these two hyperparameters again at last.

3.5 Data Sets

In our experiment, we used seven different data sets from UCI data repositories, available in csv (Comma Separated Value) format. Below is the table illustrating the brief information about the data sets we have used.
Each data set was divided into two equal portions as training and testing sets for training a classification model and testing its classification accuracy in unseen data sets.

Data Sets	Number of Training		Number of Testing		Classification Type
	Rows	Columns	Rows	Columns	
Adult	24420	15	24420	15	Binary
Bank	20593	21	20593	21	Binary
Dota	51471	117	51471	117	Binary
Flavours of Physics	33775	51	33775	51	Binary
Forest Cover Type	290505	55	290505	55	Multiclass
MiniBooNE	65031	51	65031	51	Binary
Porto Seguro	21695	59	21695	59	Binary

Table 3.1: Table illustrating the information about the data sets used.

Out of the 7 data sets, described in table 3.1, only Adult, Bank, and the Porto Seguro data set has missing value. Similarly, Adult, Bank, Dota, Forest Cover Type, and Porto Seguro data sets has both categorical and numerical types attribute. And, Flavours of Physics and MiniBooNE data sets has only numerical type attributes or columns.

3.6 Cross Validation

After developing a machine learning model, it is a crucial task to find out its reliability. A model trained with data sets will perform acceptably on that particular data set. For finding its reliability we need to test the model in a data set which was not used in its training phase. There are different approaches for feeding unseen data set to model. We followed the simple approach and that was splitting the whole data set into two parts. One part was used for training the model and the other part was used for testing.
For making the model less unbiased, we implemented a 5-fold cross validation technique where the whole data set was split into 5 different blocks. Each block was used as a validation set and remaining 4 block was used for training the model. The model was trained on the training set of 4 blocks and later tested on the validation set of 1 block. The accuracy of model was calculated for the test on the validation set. The process was repeated until all of the five blocks were treated as a validation data set and for each repetition, accuracy of the model in validation data set was calculated. After the training and later the testing of the model for all iterations, the overall accuracy of the model was calculated as the mean of accuracy obtained from 5 rounds of training and testing process.
While training and testing the model, its accuracy can also be highly influenced by imbalanced classes. For example, if there are 100 binary classification examples, and 80 examples represent positive outcome and 20 examples represent negative outcome, there is always 80% chance for an example to be classified as positive during the testing phase. To overcome this problem, along with cross validation, we created strata from the training data set and performed cross-validation on those strata. This whole process was implemented by using RepeatedStratifiedKFold() function from the scikit learn library.

3.7 Repeated Stratified Cross Validation

For solving the problem of imbalanced classes, different strata are created, containing the same proportion of examples from all classes. These strata are treated as a block in the k-fold cross validation technique.
In our experiment, we used the following function from Scikit Learn library, to perform the repeated stratified k-fold cross validation.

```
RepeatedStratifiedKFold(n_splits=num_folds, n_repeats=num_repetition,
                        random_state=Int_value)
```

		Training Set								Validation Set	
		Fold-1		Fold-2		Fold-3		Fold-4		Fold-5	
Repetition: 1	Iteration: 1	0	1	2	4	5	6	8	9	3	7
	Iteration: 2	0	1	2	3	5	6	7	8	4	9
	Iteration: 3	1	2	3	4	6	7	8	9	0	5
	Iteration: 4	2	3	0	1	4	5	7	9	6	8
	Iteration: 5	3	4	1	2	6	7	8	9	0	5
Repetition: 2	Iteration: 1	0	1	2	3	4	5	8	9	6	7
	Iteration: 2	0	1	2	4	5	6	7	8	3	9
	Iteration: 3	3	5	0	1	6	7	8	9	2	4
	Iteration: 4	6	7	3	4	0	2	8	9	1	5
	Iteration: 5	1	2	3	4	5	6	7	9	0	8

Table 3.2: Table illustrating the distribution of data clusters during the repeated stratified k-fold cross-validation process.

In the function, RepeatedStratifiedKFold(), declared above, parameter 'n_splits' denotes the integer value according to which the whole data set will be divided into folds or blocks. Similarly, n_repeats denotes the integer value for which the whole k-fold cross validation process will be repeated. And random_state also takes any integer value, which is declared to keep track records of the data randomly distributed in each iterations and throughout whole repetitions.
To demonstrate the concept of Repeated Stratified K-Fold cross validation, we have made a table which represents the distribution of imaginary data having only 10 rows. We used 5-fold cross validation repeating for 2 times.

```
RepeatedStratifiedKFold(n_splits=5, n_repeats=2,
                        random_state=36)
```

Let us suppose there are 10 data examples with features and target variables. We are performing the Repeated Stratified K-Fold cross-validation and we have declared in our function to create 5 folds with two repetitions, so 5 different folds were created, each containing 2 examples or data and the process was repeated twice. For implementing the stratification sampling process, the function takes the sample

from all classes in equal proportions,with each class having equal probabilities of being selected during the training and testing process. So, each fold or block created during the process can be understood as strata. In each strata, equal proportion of samples from all classes are included. In the second repetition, the entire blocks of strata were shuffled randomly. Here, the random_state parameter is defined with some integer value, to keep a record of the generated strata throughout the process. The accuracy of the model trained with 4 different strata is tested with 5th strata declared as a testing set. This process is repeated during all 5-folds and the accuracy of the model is calculated in each iteration. When accuracy is obtained from 10 iterations , throughout the two repetitions, average accuracy is calculated. For this reason, the accuracy calculated is believed to be more unbiased and reliable. This is because the model gets the opportunity to test itself with 10 different validation or testing sets.

We assigned different values for repetition while training the model with different data sets. We performed the following steps with every data sets before deciding the value for repetition in repeated stratified cross validation technique.

- Step 1: Declare 200000 as a constant.
- Step 2: Obtain the total number of rows in the training data set (r).
- Step 3: Divide 200000 by the total number of rows (r).

If l is the value for repetition then,

$$l = \frac{200000}{r}$$

The result, "l", if obtained as floating point, then it was converted into integer by rounding off technique. And the value of "l" was used to determine how many times the whole stratified cross validation technique would be repeated for the particular data set. In this way the number of rows or the data set size influenced the times the stratified cross validation process was repeated for that particular data set.

3.8 Evaluation metric

We tested our trained model with testing data sets. The volume of training data set and testing data set is described in section 3.4. As a performance measure we used accuracy. Accuracy is calculated by:

$$\text{Accuracy} = \frac{\text{Total number of correct predictions made}}{\text{Total number of all predictions made}}$$

For example, let us consider the following imaginary confusion matrix. May this matrix represents the results of binary classification of 100 samples. Lets consider 60 samples belong to positive case and 40 samples belong to negative case.

Predicted Versus Actual	Predicted Positive	Predicted Negative
Actual Positive	50 TP	10 FN
Actual Negative	5 FP	45 TN

Table 3.3: Confusion matrix.

In above confusion matrix, TP represents True Positive, FP represents False Positive, FN represents False Negative and TN represents True Negative. Now we can calculate the accuracy of the model from the above confusion matrix.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$

After replacing the variable with constant values from table 4 in above equation, we will get:

$$\text{Accuracy} = \frac{50 + 45}{50 + 45 + 10 + 5}$$

$$\text{Accuracy} = \quad 0.863636$$

So, we obtained accuracy as 0.863636. This means the model classifies the examples with 86.36% accuracy or any instance has a probability of 86.36% of being correctly classified.

For evaluating the performance of our model we used accuracy as evaluation metrics.

Chapter 4

Implementation

In this chapter we will discuss our approach for performing the comparative analysis of manually tuned gradient boosting algorithms, XGBoost, LightGBM, and CatBoost, and automated machine learning algorithm, Classifium and H2O in terms of classification accuracy. For this, we performed the task given in different sections below.

4.1 Implementation of XGBoost

XGBoost or Extreme Gradient Boosting library is based on gradient boosting technique. It is available as a free open-source software package. Its software package is available in Python, R and Java languages. We used XGBoost package implemented in the Python programming language. Details about this package can be found at: https://xgboost.readthedocs.io/en/latest/index.html

4.1.1 XGBoost modelling

XGBoost has a wide range of hyperparameters, categorised as general parameters, booster parameters, learning task parameters, and command-line parameters. We have briefly described these hyperparameters in Introduction section 1.7. Details about the tuning process and steps are provided in the appendix of this report.
For modelling XGBoost with different data sets, we first declared a function XGBClassifier() imported from XGBoost library. At first, all hyperparameters of this function were left with their default values. Their values were tuned by following the hyperparameter tuning pipeline described in section 3.1.
Here we have listed those hyperparameters of XGBClassifier() which we used and tuned during experimentation:

1. booster : XGBoost provides different options for boosting techniques. We used 'gbtree' booster, which is a default booster in XGBoost.

2. learning_rate : We have tuned the learning rate of XGBoost during our experiment because it determines the next range at which gradients are

calculated during the gradient boosting process. So, this parameter influences the accuracy of the machine learning model.

3. n_estimators : We have tuned the number of trees that were built in XGBoost during our experiment. This also influences the accuracy of the machine learning model.

4. max_depth : We have tuned max_depth of XGBoost tree. Because depth can control the overfitting issue of the machine learning model.

5. min_child_weight : We have tuned the parameter min_child_weight, which can control the overfitting problem of the machine learning model.

6. gamma : Gamma is one of the regularization parameters, which helps to adjust the weight of gradients during the training process. So we tuned its value during our experiment.

7. subsample : Since XGBoost supports stochastic gradient boosting technique, we optimized this hyperparameter, which controls the amount of data from the whole data set to be used for building trees each time. So, it also helps to determine the accuracy of the model.

8. colsample_bytree : Sampling can be done column-wise. Selecting the important features while generating a decision tree is controlled by this hyperparameter. So we tuned this parameter to get the best accuracy from model.

9. reg_alpha : It is also a regularization parameter that controls the overfitting problem of the model during training. We have tuned this parameter during our experimentation.

10. reg_lambda : It is also another regularization parameter that controls the overfitting problem like gamma and alpha of the model during training. We have tuned this parameter during our experimentation.

11. tree_method : This hyperparameter determines the method used for building a tree. We have used gpu_hist method, which is a gpu implementation of hist method.

We used 7 different data sets described in section 3.4 for tuning xgboost classifier model. For the training and testing propose, whole data sets were divided into half, one half used for training and other half was used for testing. For those features which had categorical values, we used one-hot encoding technique to change it from categorical to numerical. For handling missing values, XGBoost has a built-in feature. In our data set, we replaced missing strings denoting missing values with np.NAN. So while declaring the XGBoost classifier function, we used a parameter 'missing' and assigned np.NAN as value to it.
For objective function, we used 'binary:logistic' function for binary classification,

which outputs the probability of each samples belonging to a certain class. And for multi-class classification, we used 'multi:softprob' as an objective function.
We have used RepeatedStratifiedKFold() function from scikit learn library to perform stratified repeated cross-validation. This technique helped to tackle the problem of imbalanced data sets. Details about this strategy are given in section 3.5.1.
With all configurations mentioned above, we performed hyperparameter tuning of XGBoost with 7 different data sets following the hyperparameter tuning pipeline described in section 3.1. The result obtained as accuracy after training and testing xgboost classifier model with 7 different data sets is tabulated in Results section.

4.2 Implementation of LightGBM

Light Gradient Boosting Machine or LightGBM is also a software package implementing gradient boosting framework for classification, regression and ranking tasks in the machine learning domain. It is available in different versions which supports Python, R and C programming language. We used LightGBM package built for Python programming language. Details about this package or library can be found at https://lightgbm.readthedocs.io/en/latest
/index.html

4.2.1 LightGBM modelling

LightGBM also has a wide variety of hyperparameters categorized under different categories like Core parameters, Learning Control parameters, IO parameters, Predict parameters, Objective parameters, Metric parameters, Network parameters and GPU parameters.
But from different categories, we have selected a limited number of parameters for modelling LightGBM with all 7 data sets explained in section 3.4. And those hyperparameters are given below. Details about these hyperparameters is provided in Introduction section 1.7. Details about the tuning process and steps of LightGBMClassifier() with all 7 data sets are provided in the appendix of this report.
For modelling LightGBM with all 7 data sets we used LightGBMClassifier() function from LightGBM library. At first, all hyperparameters were left unchanged with their default value. And following the hyperparameter tuning pipeline, described in section 3.2, we only tuned below listed hyperparameters of LightGBMClassifier().

1. boosting : LightGBM uses different boosting techniques, gbdt (gradient boosting decision tree), rf (random forest), dart and goss (gradient one side sampling). For boosting technique, we used gbdt which is also a default boosting technique in LightGBMClassifier().

2. learning_rate : Learning rate is categorized as a core parameter of LightGBM. It controls the gradient calculated in different steps. It has a high influence in the performance of LightGBM in terms of speed and accuracy.

3. n_estimator : It is another important parameter of LightGBM which controls the total number of trees built during the boosting process. It influences the model performance in terms of training speed and classification accuracy. So, we tuned this parameter through our pipeline.

4. max_depth : We tuned this parameter because it is used to control the overfitting problem of model.

5. min_gain_to_split : This parameter sets the threshold for performing further splits from a leaf in a tree. It helps to increase the training speed and the accuracy. So, we tuned this parameter.

6. min_data_in_leaf : This parameter sets the threshold for minimum number of data to be present in every leaf before splitting. This helps to control overfitting and boosts the accuracy of model.

7. num_leaves : This parameter sets the maximum number of leaves in one single tree built during the boosting process. We tuned this parameter.

8. bagging_fraction : LightGBM also supports stochastic gradient boosting technique. This parameter randomly selects the rows of data for training in one iteration. This helps to control overfitting problem and speeds up the training process.

9. feature_fraction : Like bagging_fraction, feature_fraction is used to randomly select part of data from the whole data set, but column wise. This parameter selects a bunch of column randomly and helps to increase training speed and accuracy. So, we tuned this parameter as well.

10. regularization_parameters : LightGBM also has regularization parameters named as reg_alpha and reg_lambda or lambda_l1 and lambda_l2. We tuned this parameter to tune the weights of gradients and control overfitting.

After declaring the function LightGBMClassifier(), including the above listed function we also included objective function for LightGBM. For binary classification, we used 'binary' as an objective function and 'multiclass' for multi-classification.
For handling the missing values, like in XGBoost, LightGBM also has implemented a default mechanism, which treats all values represented by np.NAN in Python as missing values. So, during our experiment, we replaced all missing values with np.NAN.
For handling categorical data in LightGBM, we used one-hot encoding technique, which gives the binary representation for categorical variables.
For performing cross-validation and address the problem of imbalanced target classes in data sets, we used RepeatedStratifiedKFold() function from scikit learn library.

With all configurations mentioned above, we performed modelling of LightGBM using 7 different data sets described in section 3.4. While modelling, we used separate data subsets for training and testing. The results obtained as accuracy after cross-fold training and testing are listed in the table presented in Result section.

4.3 Implementation of CatBoost

Categorical Boosting or CatBoost is another software package that implements the gradient boosting framework for classification, regression and ranking tasks in a machine learning domain. It is available for Python and R language as different packages. We used Python package implementing CatBoost. Details about this package or library can be found at https://catboost.ai/docs.

4.3.1 CatBoost modelling

Unlike XGBoost and LightGBM, it has a very limited number of hyperparameters for tuning. According to the official documentation of CatBoost, https://cat boost.ai/docs, its parameters' default values are tuned to fit various data sets. The analysis result is presented in this paper [67]. Here, we have listed few hyperparameters that we have selected for tuning CatBoost algorithm with our 7 different data sets tabulated in section 3.4. Details about these parameters are provided in section Introduction section 1.7. Detailed procedure followed during hyperparameter optimization process is provided in Appendix section of this report. For modelling CatBoost with all 7 data sets we used CatBoostClassifier() function from CatBoost library. At first, all hyperparameters were left with their default values. And following the hyperparameter tuning pipeline, described in section 3.3, we tuned only those hyperparameters listed below.

1. grow_policy : CatBoost has different techniques for building trees, like, Symmetric, Depthwise, and Lossguide. We have used Symmetric tree growing policy, which is also a default policy for growing trees in CatBoost.

2. learning_rate : Learning rate is used for controlling the width of intervals in which gradients are calculated. This algorithm has a high influence on classification accuracy and training time. So, we have tuned this algorithm during our experiment with 7 different data sets.

3. iterations : This parameter controls the number of trees built during the training process. So, it also affects the performance accuracy and training time. We have tuned this parameter during our experiment.

4. depth : Depth of tree is highly sensitive with overfitting problems. So, we have tuned this parameter during our experiment.

5. l2_leaf_reg : We have tuned the regularization function of CatBoost.

6. random_strength : This parameter is used for controlling the overfitting problems. This parameter assigns a score to every possible split during tree building and the split with the highest score based on loss function is selected. So, we have tuned this parameter to boost up the performance accuracy.

7. bagging_temperature : This parameter assigns random weights to objects. We have tuned this parameter during our experiment.

8. border_count : This parameter controls the number of splits required for numerical features. We have tuned this parameter during our experiment.

After declaring the function CatBoostClassifier() with the above-mentioned hyperparameters, we also added an objective function. For binary classifications, the 'Logloss' was used as the objective function and for multi-class classification 'MultiClass' was used as the objective function.
For handling the missing values, CatBoost has built-in features which treat float NAN (Not a Number) value in data sets as missing attributes. Any values represented by a string of 'NAN' or 'null' or 'none' is also treated as a missing value. So, to handle the missing values of our data sets, we simply replaced the 'NAN' string in missing values.
Since CatBoost is built for handling categorical data, we did not perform any encoding technique to convert our categorical features to numerals. This technique is briefly explained in section 1.6. Details about this are presented at [64].
For addressing the issues of imbalanced data sets and for cross-validation, we have performed a repeated stratified cross validation technique with the function RepeatedStratifiedKFold() implemented from the scikit learn library.
With all technical configurations and settings, we performed the modelling of the CatBoost algorithm with all 7 different data sets described in section 3.4. The results obtained as classification accuracy after cross-fold training and testing are listed in the table presented in Result section.

4.4 Implementation of H2O

H2O is an automatic machine learning algorithm, that performs data processing, modelling and evaluation by itself [91]. It makes it easy for beginner users with little knowledge in machine learning to generate a machine learning model using H2O. H2O supports Supervised and Unsupervised machine learning techniques. For Supervised learning approach H2O supports different algorithms like Deep Learning (Neural Networks), Distributed Random Forest (DRF), Generalized Linear Model (GLM), Maximum R Square Improvements (MAXR), Generalized Additive Models (GAM), ANOVA GLM, Gradient Boosting Machine (GBM), Naïve Bayes Classifier, RuleFit, Stacked Ensembles, Support Vector Machine (SVM), and XGBoost as base learner. Using these algorithms, we can perform classification and regression tasks in H2O. For Unsupervised learning, H2O supports different algorithms like Aggregator, Generalized Low Rank Models (GLRM), Isolation Forest, Extended Isolation Forest, K-Means Clustering, and Principal Component Analysis (PCA) In H2O, it is not required to separately impute the missing values, perform data encoding technique for categorical attributes, and hyperparameter tuning of any model.
We have used H2O to model with our 7 different data sets. Details about its modelling are explained in the next section.

4.4.1 H2O modelling

Unlike XGBoost, LightGBM, and Classifium, in H2O we do not need to perform data processing, hyperparameter optimization and evaluation.
Since H2O supports many machine learning algorithms for its base learner, we modeled H2O with 7 data sets in two different approaches; a) At first by allowing H2O to select the either single best algorithm as its base learners or allowing it to create the best classification model by assembling more than one algorithms that it supports as its base learner by itself. b) And, since we were comparing the performance of different gradient boosting algorithm with different settings of hyperparameter tuning pipeline, we allow H2O to use only XGBoost as for its base learner, and perform classification task on all of the 7 data sets.

Following task were performed throughout the both approaches.

1. We loaded the data into the H2O frame work using python script.

2. Data set for the training and the testing were loaded separately as two different data frames.

3. Data attributes were divided into predictors and target groups using the same approach which we have used earlier in XGBoost, LightGBM and CatBost using python.

4. By default, H2O performs regression. To denote classification tasks, we must have to denote our target column by categorical data type. Attributes can be changed from any form to categorical by using asfactor() in H2O. So, to perform the classification task, we denote our target attribute only as categorical by using the asfactor() function in H2O.

5. For handling imbalanced data set, we used 'balanced_class' feature of H2O.

6. By default, H2O performs 5-fold cross validation technique. So we did not assign any function externally to perform cross-validation. So, during the training, the data set was divided into 5 folds by the H2O default algorithm.

7. In H2O we can set the training time. But we did not set any constraints for training time. Every classification model was trained with a default time limit, which is 3600 seconds, besides a model trained with Forest Cover data set. This is because the Forest Cover data set is the biggest data set that we have used in our experimentation. So in the model, we set 4500 seconds for models trained with the Forest Cover data set.

8. The list of models were published on leader board, based on the evaluation metric score. We have set accuracy as our evaluation metric.

9. We selected two different models. One of which is developed using XGBoost and another was developed using assembling approach.

10. Both models were tested with testing data sets for determining their testing accuracy, separately.

The cross-validation training accuracy and testing accuracy of classification models generated with H2O and all 7 different data sets are tabulated in a table as result. The table is presented in the Result section of this report.

4.5 Implementation of Classifium

Classifium is an automatic machine learning algorithm for classification. Unlike in XGBoost, LightGBM and CatBoost,it is not required to perform hyperparameter tuning by the user in Classifium. Hyperparameter tuning is done by the built-in algorithms automatically. Also, it is not essential to perform the data pre-processing like imputing missing values and changing categorical attributes to numerical in Classifium. Simply missing values are represented by the special letter "?", and all the categorical and numerical columns are represented as "nominal" and "ordinal". Columns having name "nominal" are treated as categorical column, and built-in mechanism is used to convert them into numerical. For representing target column or feature, string "output" is assigned as a name of column. Also, if any columns or features are not important, then those are simply denoted by the string "Ignore", and they are dropped while training.

4.5.1 Classifium modelling

We followed the following steps for generating classification models using Classifium with all of the 7 data sets. The steps are described in series below.

1. We installed the executable files of Classifium from its source into our implementing machine.

2. We prepared separate data set for training and testing as discussed in above section 4.5.

3. Unlike other algorithms that we used for our experiment, Classifium requires extra name file, containing the names of columns in exact order in which they are presented in the data files.

4. We supplied the training data set in a csv file and name file of the data set in the same directory where Classifium was installed as a executable file.

5. We run the following command to perform the classification task:
   ./cl -f filestem –numFolds –numReps –numThreads
   Here, filestem means the name of data file representing training data and name files of that training and testing data. And numFolds represents the number of folds that will be made for performing cross validation. Similarly, numReps means, the parameter numReps will perform the repeated stratified

cross validation for that number of iterations as its value is assigned. And numThreads means the number of cpus in which the algorithm will run in parallel.

6. After training the model with the data set, Classifium generated the training model with its name followed by an extension "forest".

7. That forest file is used to perform the prediction on test data set by executing following code in the same directory where the Classifium is installed and where the name files, and testing data set is present along with the "forest" file.
   ./predict filestem

8. After running the above code, we obtained the classification accuracy on training data set and testing data set along with the confusion matrix.

4.6 Implementation platform

The technical specifications of the machine on which we ran all of our algorithms are presented below.
Model Name: AMD Ryzen Threadripper 2990WX 32-Core Processor
Architecture: x86_64
Hyperthreading: Enabled
Number of CPUs: 64
RAM: 128 GB DDR4
Graphics: NVIDIA 10.8 GB

Chapter 5

Results

In this chapter, we will illustrate the results from our experiment to address the goal of our study. For comparing the accuracy of automatic machine learning algorithm, Classifium, with gradient boosting decision tree based algorithms like XGBoost, LightGBM, and CatBoost, and automated machine learning tools H2O, we performed the modelling of these algorithms as described in sections 4.1, 4.2, 4.3, 4.4, and 4.5. We followed the hyperparameter tuning pipeline as described in sections 3.1, 3.2, and 3.3 for modelling our classification algorithms with all 7 data sets. We implemented 5-fold cross validation technique with the repeated stratified sampling method for generating each classification model for all algorithms with the 7 data sets, and obtained cross-validation training accuracy from training models. The cross validation training results is represented as CV training accuracy in the table below. The models were trained and tested with the separate set of training and testing data sets for their classification accuracy. Data sets used for training and testing are listed in a table in section 3.5. The results obtained from each algorithm with all 7 data sets are tabulated in table 5.1 and 5.2 which are presented below.

In table 5.1, we have listed the cross-validation training and testing accuracy of XGBoost, LightGBM, CatBoost, Classifium and H2O. In the table 5.1, column H2O represents the results which are generated using all of its base learners, Deep learning, Distributed Random Forest, XGBoost, Gradient Boosting Machine, and others. It tested all its base learners against a particular data set. It assembled all base learners and stacked them, and performed training and testing.
In table 5.2, we have tabulated the training and testing results from all algorithms along with H2O with only XGBoost as its base learner. Our focus was to test for performance of hyperparameter tuning pipeline of H2O in two different configuration. One, only with gradient boosting algorithm like XGBoost, and another by the ensemble of many base learners with hyperparameter tuning pipeline. We found that the classification accuracy of H2O generated with stacked ensemble model and only with XGBoost were very much close to the results of Classifium and other algorithms in the same data set.

We have compared the classification accuracy of Classifium with every other algorithms used for our study. The comparison results are presented in the graphs,

in section 5.3. Also, we performed the statistical significant test for the classification results obtained from XGBoost, LightGBM, CatBoost and H2O against the results obtained from Classifium. The result from statistical significance test is presented in section 5.4.

5.1 Results obtained from XGBoost, LightGBM, CatBoost, Classifium, and H2O implemented with stacked ensemble models.

Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O	Classifium
Adult	CV Training Accuracy: 0.871785 Testing Accuracy: 0.875225	CV Training Accuracy: 0.871334 Testing Accuracy: 0.875389	CV Training Accuracy: 0.871021 Testing Accuracy: 0.874815	CV Training Accuracy: 0.871007 Testing Accuracy: 0.876577	CV Training Accuracy: 0.8703 Testing Accuracy: 0.87523
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O	Classifium
Bank	CV Training Accuracy: 0.917015 Testing Accuracy: 0.916525	CV Training Accuracy: 0.917297 Testing Accuracy: 0.915893	CV Training Accuracy: 0.916988 Testing Accuracy: 0.917399	CV Training Accuracy: 0.91837 Testing Accuracy: 0.915894	CV Training Accuracy: 0.9164 Testing Accuracy: 0.915995
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O	Classifium
Dota	CV Training Accuracy: 0.596279 Testing Accuracy: 0.594859	CV Training Accuracy: 0.594451 Testing Accuracy: 0.593751	CV Training Accuracy: 0.590784 Testing Accuracy: 0.592838	CV Training Accuracy: 0.599075 Testing Accuracy: 0.597268	CV Training Accuracy: 0.5982 Testing Accuracy: 0.597529

Table 5.1 continued from previous page

Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O	Classifium
Flavours of Physics	CV Training Accuracy: 0.894265 Testing Accuracy: 0.892701	CV Training Accuracy: 0.892622 Testing Accuracy: 0.890747	CV Training Accuracy: 0.894414 Testing Accuracy: 0.892790	CV Training Accuracy: 0.892287 Testing Accuracy: 0.889297	CV Training Accuracy: 0.8929 Testing Accuracy: 0.89158
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O	Classifium
Forest Cover	CV Training Accuracy: 0.963794 Testing Accuracy: 0.968155	CV Training Accuracy: 0.90808 Testing Accuracy: 0.919009	CV Training Accuracy: 0.956348 Testing Accuracy: 0.960062	CV Training Accuracy: 0.951137 Testing Accuracy: 0.954307	CV Training Accuracy: 0.9682 Testing Accuracy: 0.971873
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O	Classifium
MiniBoo NE	CV Training Accuracy: 0.946596 Testing Accuracy: 0.946856	CV Training Accuracy: 0.946974 Testing Accuracy: 0.947624	CV Training Accuracy: 0.946605 Testing Accuracy: 0.947056	CV Training Accuracy: 0.944673 Testing Accuracy: 0.945564	CV Training Accuracy: 0.9455 Testing Accuracy: 0.947057
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O	Classifium

Table 5.1 continued from previous page

Porto Seguro:	CV Training Accuracy: 0.594033 Testing Accuracy: 0.595298	CV Training Accuracy: 0.593906 Testing Accuracy: 0.593455	CV Training Accuracy: 0.593495 Testing Accuracy: 0.592164	CV Training Accuracy: 0.592533 Testing Accuracy: 0.596958	CV Training Accuracy: 0.5936 Testing Accuracy: 0.598544

Table 5.1: Table illustrating the accuracy of 5 different algorithms with 7 different data sets.

5.2 Results obtained from XGBoost, LightGBM, CatBoost, Classifium, and H2O implemented with only XGBoost.

Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O (XGBoost)	Classifium
Adult	CV Training Accuracy: 0.871785 Testing Accuracy: 0.875225	CV Training Accuracy: 0.871334 Testing Accuracy: 0.875389	CV Training Accuracy: 0.871021 Testing Accuracy: 0.874815	CV Training Accuracy: 0.871007 Testing Accuracy: 0.874611	CV Training Accuracy: 0.8703 Testing Accuracy: 0.87523
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O (XGBoost)	Classifium
Bank	CV Training Accuracy: 0.917015 Testing Accuracy: 0.916525	CV Training Accuracy: 0.917297 Testing Accuracy: 0.915893	CV Training Accuracy: 0.916988 Testing Accuracy: 0.917399	CV Training Accuracy: 0.918370 Testing Accuracy: 0.915894	CV Training Accuracy: 0.9164 Testing Accuracy: 0.915995

Table 5.2 continued from previous page

Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O (XGBoost)	Classifium
Dota	CV Training Accuracy: 0.596279 Testing Accuracy: 0.594859	CV Training Accuracy: 0.594451 Testing Accuracy: 0.593751	CV Training Accuracy: 0.590784 Testing Accuracy: 0.592838	CV Training Accuracy: 0.584543 Testing Accuracy: 0.583474	CV Training Accuracy: 0.5982 Testing Accuracy: 0.597529
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O (XGBoost)	Classifium
Flavours of Physics	CV Training Accuracy: 0.894265 Testing Accuracy: 0.892701	CV Training Accuracy: 0.892622 Testing Accuracy: 0.890747	CV Training Accuracy: 0.894414 Testing Accuracy: 0.892790	CV Training Accuracy: 0.891636 Testing Accuracy: 0.887224	CV Training Accuracy: 0.8929 Testing Accuracy: 0.89158
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O (XGBoost)	Classifium
Forest Cover	CV Training Accuracy: 0.963794 Testing Accuracy: 0.968155	CV Training Accuracy: 0.90808 Testing Accuracy: 0.919009	CV Training Accuracy: 0.956348 Testing Accuracy: 0.960062	CV Training Accuracy: 0.949257 Testing Accuracy: 0.955329	CV Training Accuracy: 0.9682 Testing Accuracy: 0.971873
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O (XGBoost)	Classifium

Table 5.2 continued from previous page

MiniBoo NE	CV Training Accuracy: 0.946596 Testing Accuracy: 0.946856	CV Training Accuracy: 0.946974 Testing Accuracy: 0.947624	CV Training Accuracy: 0.946605 Testing Accuracy: 0.947056	CV Training Accuracy: 0.943965 Testing Accuracy: 0.944242	CV Training Accuracy: 0.9455 Testing Accuracy: 0.947057
Algorithms With Data set	XGBoost	Lightgbm	CatBoost	H2O (XGBoost)	Classifium
Porto Seguro:	CV Training Accuracy: 0.594033 Testing Accuracy: 0.595298	CV Training Accuracy: 0.593906 Testing Accuracy: 0.593455	CV Training Accuracy: 0.593495 Testing Accuracy: 0.592164	CV Training Accuracy: 0.58926 Testing Accuracy: 0.595391	CV Training Accuracy: 0.5936 Testing Accuracy: 0.598544

Table 5.2: Table illustrating the accuracy of 5 different algorithms, H2O (XGBoost), with 7 different data sets.

In the table 5.2, the column H2O(XGBoost) represents the classification accuracies obtained by modelling H2O with only XGBoost as its base learner, with all 7 data sets.

5.3 Graphical illustration of the results from experiment.

5.3.1 Classifium versus XGBoost

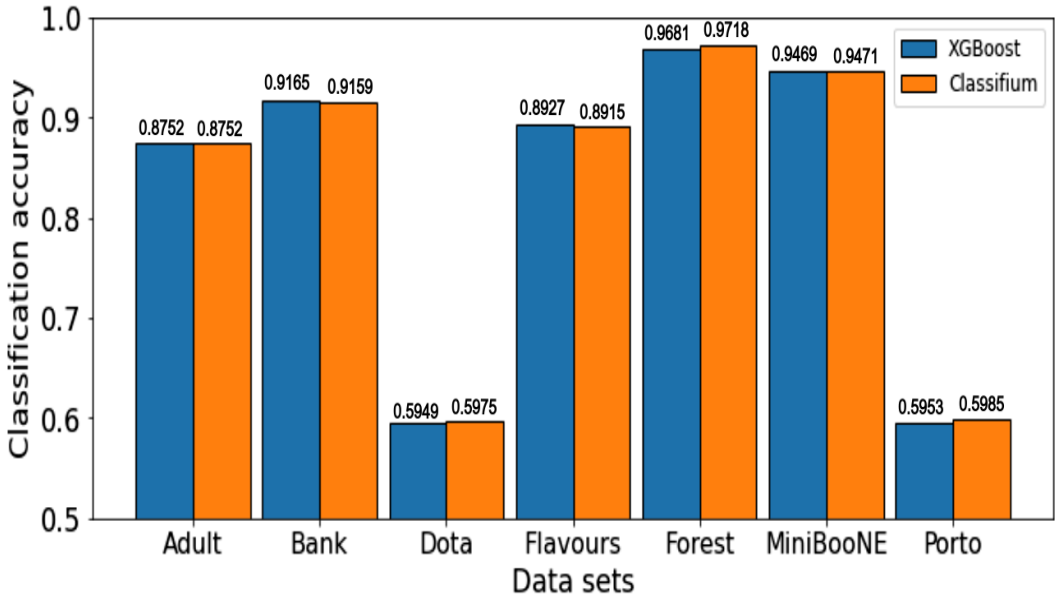


Figure 5.1: Classification accuracy of XGBoost and Classifium with the different data sets.

The graph illustrates the comparative results between Classifium and XGBoost for their classification accuracy with 7 different data sets. The y-axis of the graph represents the accuracy categories in the interval of 10%, and the x-axis represents the distribution of 7 different data sets.

Classifium and XGBoost performed almost same in terms of classification accuracy with Adult data set. It only has difference of 0.000005% in terms of classification accuracy. In this case we can say Classifium performed slightly better than XGBoost. Similarly with the Dota data set, Classifium outperformed XGBoost by 0.00267% of classification accuracy. And also with the Forest Cover Type data set, Classifium outperformed XGBoost by 0.003718% of classification accuracy. Likewise, with the MiniBooNE data set, Classifium outperformed XGBoost by the 0.000201% of classification accuracy. And, with the Porto Seguro data set, Classifium outperformed XGBoost by the difference of 0.003246% of accuracy.

But, with the bank data set XGBoost outperformed Classifium with the slight difference of 0.00053% in terms of classification accuracy. And with the Flavours of Physics data set, XGBoost outperformed Classifium by the difference of 0.001121% in classification accuracy.
The difference between the classification accuracy is calculated based on the classification results presented in the table 5.1.

5.3.2 Classifium versus LightGBM

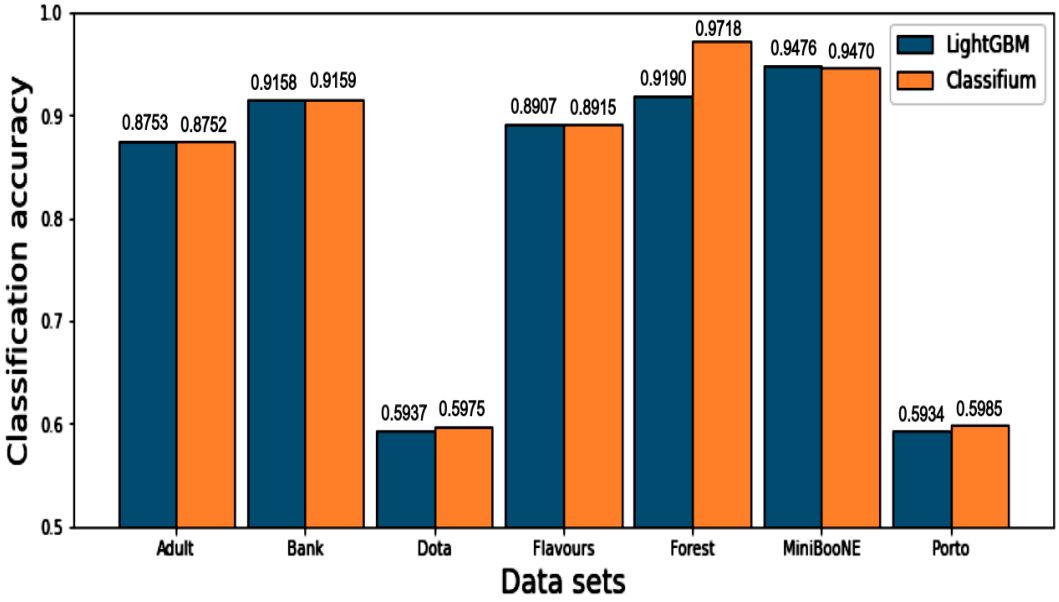


Figure 5.2: Classification accuracy of LightGBM and Classifium with the different data sets.

The graph illustrates the comparative results between Classifium and LightGBM for their classification accuracy with 7 different data sets. The y-axis of the graph represents the accuracy categories in the interval of 10%, and the x-axis represents the distribution of 7 different data sets.

From the graph we can say that, Classifium outperformed LightGBM by 0.000102% of difference in classification accuracy. So, Classifium and LightGBM performed almost same with the Bank data set. With the Dota data set, Classifium outperformed the LightGBM by 0.003778% of difference in classification accuracy. And with the Flavours of Physics data set, Classifium outperformed LightGBM by 0.000833% of difference in classification accuracy. Similarly with the Forest Cover Type data set, Classifium outperformed LightGBM by 0.052864% of difference in classification accuracy, which is the highest difference in classification accuracy made by Classifium against all other algorithms with 7 data sets, in our study. And, Classifium outperformed LightGBM with Porto Seguro data set by the difference of 0.005089% in classification accuracy.

But with the Adult data set, LightGBM outperformed Classifium with the slight difference of 0.000159% in classification accuracy. Similarly LightGBM outperformed Classifium with the MiniBooNE data set by 0.000567% of classification accuracy. The difference between the classification accuracy is calculated based on the classification results presented in the table 5.1.

5.3.3 Classifium versus CatBoost

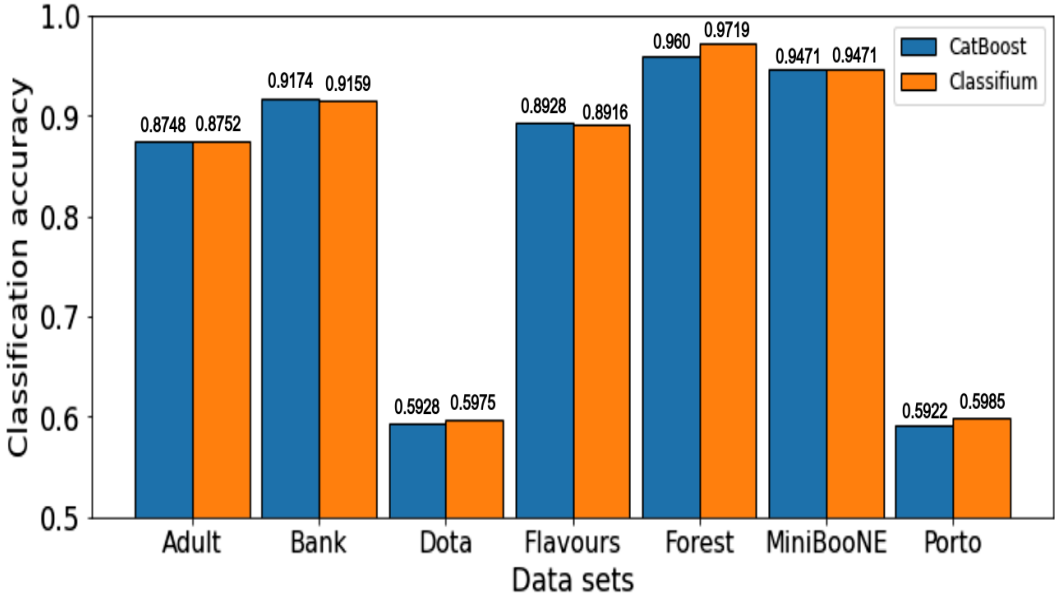


Figure 5.3: Classification accuracy of CatBoost and Classifium with the different data sets.

The graph illustrates the comparative results between Classifium and CatBoost for their classification accuracy with 7 different data sets. The y-axis of the graph represents the accuracy categories in the interval of 10%, and the x-axis represents the distribution of 7 different data sets.

From the graph it is found that Classifium outperformed CatBoost with Adult data set by 0.000415% of classification accuracy. Similarly with the Dota data set, Classifium outperformed CatBoost by 0.004691% of classification accuracy. And with the Forest Cover Type data set as well, Classifium outperformed the CatBoost by the difference of 0.011811%, in classification accuracy. This is the most significant difference in classification accuracy between Classifium and CatBoost that we observed with 7 data set. Similarly, with the Porto Seguro data set, Classifium outperformed CatBoost by 0.00638% of difference in classification accuracy. And with the MiniBooNE data set, both algorithms performed almost equally. Their difference in classification accuracy was found to be 0.000001%.

But with the Bank data set, CatBoost clearly outperformed Classifium in terms of classification accuracy by 0.001404%. Also, with the Flavours of Physics CatBoost outperformed Classifium by the difference of 0.00121% in terms of classification accuracy.

The difference between the classification accuracy is calculated based on the classification results presented in the table 5.1.

5.3.4 Classifium versus H2O(XGB)

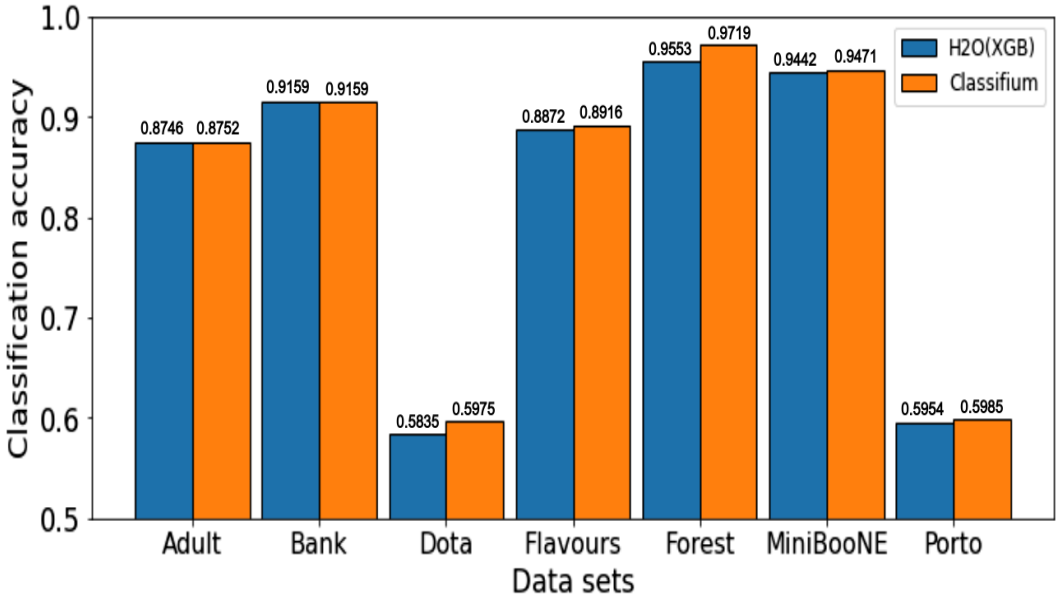


Figure 5.4: Classification accuracy of H2O(XGBoost) and Classifium with the different data sets.

The graph illustrates the comparative results between Classifium and H2O, with XGBoost only as its base learner, for their classification accuracy with 7 different data sets. The y-axis of the graph represents the accuracy categories in the interval of 10%, and the x-axis represents the distribution of 7 different data sets.

H2O model was generated using only XGBoost classifier as its base learner. We were interested to find how XGBoost will perform with the H2O hyperparameter tuning pipeline. From the classification result we observed that Classifium outperformed H2O-XGBoost by 0.000619% of classification accuracy when trained and tested with the Adult data set. Similarly Classifium outperformed H2O-XGBoost with very slight difference of 0.000101% of classification accuracy with the Bank data set. We can say both algorithms performed equally with this data set. Again, Classifium outperformed H2O by 0.014055% of difference with the Dota data set. And similarly, Classifium outperformed H2O-XGBoost by 0.004356% of difference in classification accuracy with the Flavours of Physics data set. Similarly, when tested with the Forest Cover Type data set, Classifium outperformed H2O-XGBoost by 0.016544% of difference in classification accuracy which is a significant difference. Similarly Classifium outperformed H2O-XGBoost by 0.002815% and 0.003153% of difference in classification accuracy when tested with MiniBooNE and Porto Seguro data sets respectively.
The difference between the classification accuracy is calculated based on the classification results presented in the table 5.2.

5.3.5 Classifium versus H2O

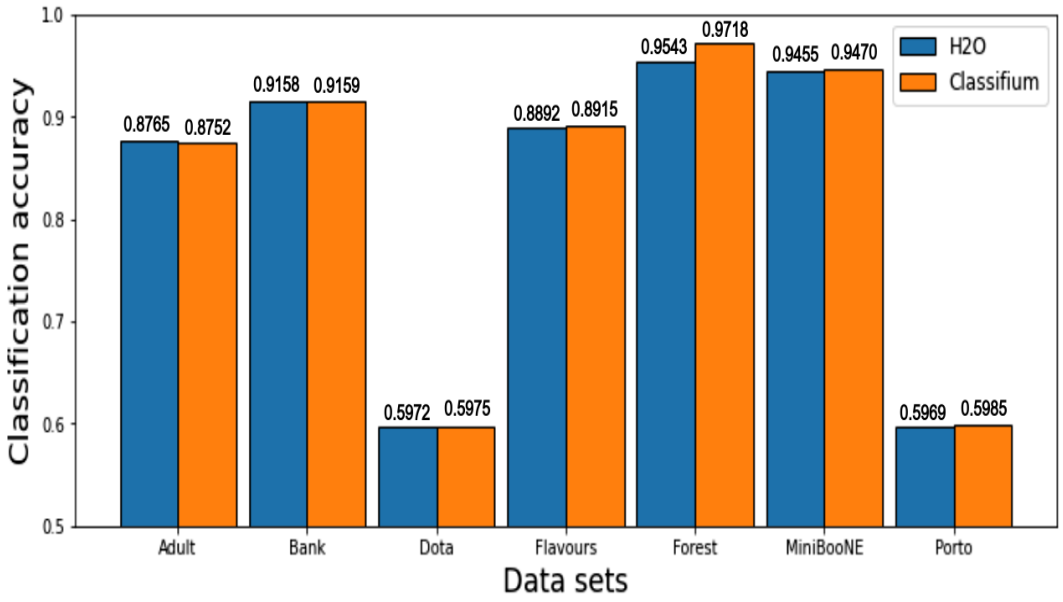


Figure 5.5: Classification accuracy of H2O and Classifium with the different data sets.

The graph illustrates the comparative results between Classifium and H2O, with all possible algorithms ensemble as its base learner, for their classification accuracy with 7 different data sets. The y-axis of the graph represents the accuracy categories in the interval of 10%, and the x-axis represents the distribution of 7 different data sets.

Classification models were generated out of H2O by letting it to run single algorithm and assembling all algorithms that it supports. From the figure 5.5 we can see that H2O beat Classifium for its classification accuracy with the Adult data set by 0.001347% of difference. But with the Bank and Dota data set, Classifium beat the H2O by slight difference of by the slight difference of 0.000101% and 0.000261% respectively in classification accuracy. Similarly, with the Flavours of Physics data set, Classifium outperformed H2O by 0.002283% of difference in classification accuracy. And with the Forest Cover Type data set, Classifium outperformed H2O by the difference of 0.017566% in classification accuracy. And similarly, with the MiniBooNE and Porto Seguro data sets, Classifium beat H2O by the difference of 0.001493% and 0.001586% of classification accuracy respectively.
The difference between the classification accuracy is calculated based on the classification results presented in the table 5.1.

5.4 Results obtained from the statistical Significance test

In the section 5.3, we presented the comparative graph illustrating the difference in classification accuracy made by Classifium and other algorithms. We also discussed the differences between the classification accuracy of Classifium against XGBoost, LightGBM, CatBoost and H2O when implemented with the all 7 data sets. So, far we could say that Classifium outperformed every other algorithm with 7 data sets in majority of classification task. But to further establish the confidence in our results, we performed the statistical significance test between the classification accuracy obtained from Classifium and other 5 different algorithms with the same data set. The table below shows the results obtained from our test for each case. The mathematical calculations performed to calculate the significance test is presented at Appendix B of this document.

Algorithms: with Data sets	Classifium Versus XGBoost	Classifium Versus LightGBM	Classifium Versus CatBoost	Classifium Versus H2O (Ensemble)	Classifium Versus H2O (XGBoost)
Adult	Equal	Equal	Equal	Equal	Equal
Bank	Equal	Equal	Equal	Equal	Equal
Dota	Equal	Equal	Equal	Equal	Better
Flavours of Physics	Equal	Equal	Equal	Equal	Equal
Forest Cover Type	Better	Better	Better	Better	Equal
MiniBoo NE	Equal	Equal	Equal	Equal	Better
Porto Seguro	Equal	Better	Equal	Equal	Equal

Table 5.3: Table illustrating the statistical significance testing results.

In the above given table 5.3, the first horizontal row represents the list of algorithms whose classification results are tested against the results of Classifium with similar data set. And the first vertical column presents the different data sets. The other cells of the table contains the results of statistical significance test performed between the Classifium and each of 5 other algorithms with each data set.

During our test, we had formulated two hypothesis, H0 and H1 as null hypothesis and alternative hypothesis. Our null hypothesis (H0) states that there is no any significant difference between the classification accuracy between two algorithms with the same data set. And our alternate hypothesis states that there is significant difference between the classification accuracy of two algorithms with the same data set. The cell having the value 'Better', denotes that the difference in classification accuracy obtained between Classifium and other algorithm with the same data set is statistical significant. That means every time Classifium will perform better than the other algorithm when tested with the other data samples from the same data set. The cell having value 'Equal' denotes that there is no any statistical significant difference between the results obtained from two algorithms, Classifium and other, with the same data set. That means the difference obtained in their classification accuracy can not generalize that the same kind of results will be obtained when tested with the other set of samples from the same data set.

From the above table we conclude that the difference in classification accuracy obtained between Classifium and XGBoost with Forest Cover Type data set is statistically significant. That means Classifium performed better than XGBoost with Forest Cover Type data set.
Similarly the difference in classification accuracy obtained between Classifium and LightGBM with Forest Cover Type data set and Porto Seguro data set are also statistically significant.
Similarly the difference in classification accuracy obtained between Classifium and CatBoost with the Forest Cover Type data set is statistically significant.
Likewise, the difference in classification accuracy obtained between the Classifium and H2O with Forest Cover Type data set is also statistically significant.
And the difference in classification accuracy obtained between the Classifium and H2O-XGBoost with the Dota and the MiniBooNE data sets are also statistically significant.

Chapter 6

Discussion

In this chapter we will present the discussion of our results on the basis of goal, research question, and results of our study.

6.1 Evaluation

Gradient boosting technique is one of the most effective and efficient techniques for classification and regression tasks in the machine learning domain. XGBoost, LightGBM, and CatBoost are the latest modern implementation of gradient boosting techniques. The automatic machine learning technique, Classifium, is also based on the gradient boosting technique. H2O, which is also an automatic machine learning technique, supports Gradient Boosting Machine and XGBoost algorithms for its base learner. During our study, we were interested in finding how manually tuned gradient boosting algorithms like XGBoost, LightGBM, CatBoost, and automatic machine learning library like H2O with built-in hyperparameter tuning pipeline will perform compared with another automatic machine learning algorithm Classifium with built-in hyperparameter tuning pipeline, in terms of classification accuracy.

From our study, we found that, in the majority of classification tasks with 7 data sets, Classifium performed better than the other algorithms when tested for their classification accuracy with the same set of 7 different data sets. We also realized that the hyperparameter tuning pipeline has a significant role in boosting the classification accuracy of manually tuned algorithms like XGBoost, LightGBM and CatBoost. We also performed a statistical significance test for finding whether our findings were statistically correct or they were correct only by chance. As an answer, we found that Classifium performed better than other algorithms which were tested against it for at least one data set, while with the rest of the other data set, classification accuracy of Classifium was either equal or comparable. The results are presented in tables 5.1, 5.2, and 5.3 in detail. For carrying out our research, we formulated a research question given in Section 2.1. The same research question is presented below:

Research question : Can automated machine learning approach like Classifium generates the classification results comparable with the results generated by the manually and carefully tuned gradient boosting machine learning libraries like xgboost, LightGBM, CatBoost and automatic machine learning library like H2O?

We conducted an extensive study for finding the results of our research question. For this, at first, we performed the hyperparameter tuning of XGBoost, LightGBM and CatBoost carefully following the hyperparameter tuning pipeline, which we have proposed in sections 3.2, 3.3, and 3.4 of this document. Since Classifium and H2O are automatic machine learning algorithms, they are supplied with their own built-in pipelines for optimizing hyperparameters, we did not have to formulate one for them. We used 7 different data sets, presented at section 3.5, to perform the classification using the algorithms XGBoost, LightGBM, CatBoost, H2O and Classifium. The classification accuracies from test data sets were used for comparative analysis.

6.2 Findings

During tuning the hyperparameters of gradient boosting based machine learning algorithms, XGBoost, LightGBM and CatBoost, we found that there exists some difference in classification accuracies, made by algorithms with default hyperparameters and algorithms with optimized hyperparameters. Also, we found that it is very important to find out the right set of hyperparameters of any algorithms for tuning them. We choose the set of hyperparameters for every algorithms based on their official documents and our experience. For XGBoost we choose 9 different hyperparameters, learning rate, number of trees, maximum depth, minimum child weight, sub-sample, column sample, regularization alpha, regularization lambda and gamma. While tuning process we found that learning rate, number of trees and depth had a greater impact on classification accuracies. These hyperparameters also contributed on controlling the overfitting problems. Similarly, other hyperparameters like column sampling, sub-sampling rate, regularization parameters also had an impact on accuracies and overfitting problems. For example when tuning XGBoost with Forest Cover Type data set, at first with only default hyperparameters, we obtained the training accuracy as 0.8975%. But when the learning rate was decreased to 0.1 and the number of trees was increased to 5000, the training accuracy raised to 0.945976%, which is a significant increment in accuracy. Likewise when other hyperparameters were adjusted, the overall testing accuracy hiked to 0.968155%. In another example, when XGBoost was tuned with Dota data set, its training accuracy was increased from 0.587338% to 0.5921801% after adjusting sub-sample and colsample. These are the representative cases that we discussed here regarding the impact of hyperparameters tuning on overall accuracy. Details about the hyperparameter tuning process and results of XGBoost with 7 data sets can be found in Appendix A of this document.

Similarly, we choose 10 different hyperparameters of LightGBM for the hyperparameter tuning process. And they were learning rate, number of trees, maximum depth of tree,

number of leaves, minimum number of data in a leaf, minimum gain to split, bagging fraction, feature fraction, and regularization parameters, lambda l1 and lambda l2. We follow the official document of LightGBM while selecting hyperparameters for the tuning process. We observed that similar to XGBoost, hyperparameter tuning enhanced the accuracy of LightGBM models. For example, with adult data set the training accuracy of model with default hyperparameters was 0.868656%. But when tuned, the training accuracy was raised to 0.871334%. Similarly, the accuracy increased with other data sets as well. An interesting observation we made during tuning process of LightGBM was with the Dota data set. We followed the iterative tuning approach, that means after reaching at the end of hyperparameter tuning pipeline, we began the whole process again from the start of pipeline, but this time with the latest optimized value from previous set of process. This approach helped to increase the testing accuracy of the model. The same approach was tried with the Forest Cover Type data set, but, no significant difference in testing accuracy was found. And also it required extra computational time. Details about the hyperparameter tuning process and results of LightGBM can be found in Appendix A of this document.

Same as for XGBoost and LightGBM, we followed the official document of CatBoost for selecting important hyperparameters for the hyperparameter tuning process. We took only 7 different hyperparameters of CatBoost and performed the hyperparameter tuning with the 7 different data sets. The hyperparameters used were learning rate, number of trees, maximum depth, regularization parameters, random strength, border count and bagging temperature. From our experiment we observed that only learning rate, number of trees, depth and regularization parameter contributed most to increase the accuracy of the model. But the hyperparameter tuning process led to higher classification accuracy with all data sets. Details about the hyperparameter tuning process and results of CatBoost can be found in Appendix A of this document.

When comparing the results only between XGBoost, LightGBM and CatBoost after the hyperparameter tuning process, we found that XGBoost outperformed LightGBM and CatBoost with the Dota, Forest COver Type and Porto Seguro data set. Similarly, LightGBM beat XGBoost and CatBoost with Adult and MiniBooNE data set. And CatBoost beat XGBoost and LightGBM with the Bank and the Flavours of physics data set.
While tuning hyperparameters of XGBoost, LightGBM and CatBoost with 7 different data sets, we also found that it takes XGBoost a longer time for training than it takes for LightGBM and CatBoost with the same data set. For example, it took 165.7 hours to perform complete hyperparameter tuning of XGBoost with the Forest Cover Type data set on GPU. But with the same data set it only took 7.84 hours for LightGBM and 100.96 hours for CatBoost for completing the hyperparameter tuning process. From this observation, we concluded that XGBoost performed better than LightGBM and CatBoost in the majority of classification tasks with a higher classification accuracies. But it took it longest time among the three algorithms for hyperparameter tuning process. After XGBoost, in terms of hyperparameter tuning

process, CatBoost took longest time. Finally, we found that LightGBM took less time to complete hyperparameter tuning process.

When building classification models with H2O, we found that it supports different machines learning algorithms as its base learner. For example, XGBoost, Gradient Boosting Machine, Deep Learning, Naive Bayes, Random Forest and et cetera. It has its own built-in algorithms to select appropriate base learner and its hyperparameter for a any given task. Also, it generates the model assembling possible base learner as a stack. For our experiment we followed two different approach while developing classification models with H2O. At first, we assigned only XGBoost as its base learner. For tuning its hyperparameters, H2O used its own built-in algorithm, which is not published yet. It also performs 5-fold cross validation training with the training data set by default.
Following this approach we obtained the classification models for all the 7 data sets. When comparing its result with the results obtained from Classifium with the same data sets, we found that classification accuracies of models obtained with Classifium were higher. After performing the statistical significance test with the same results, we found that Classifium had performed better than H2O with XGBoost as base learner, with the Dota and the MiniBooNE data set.
As a second approach, we performed modelling H2O enabling it to assemble all of its base learner in a stack. When comparing the classification accuracy of stacked H2O models with Classifium, we found that stacked H2O had higher classification accuracy than Classifium with the Adult data set only. With rest of the other data set, Classifium had higher classification accuracy. When the same results were tested for the statistical significance, we observed that Classifium had performed better than H2O with the Forest Cover Type data set. And, with the rest of the other data set, its classification accuracy were found to be equal with the accuracy obtained with the stacked ensemble H2O models.

After performing the hyperparameter tuning and modeling with XGBoost, LightGBM, CatBoost and H2O, we model Classifium with the same set of 7 data sets. When comparing the classification accuracy of Classifium with the other algorithms with the same data set, we found that in majority of classification task, Classifium outperformed other algorithms. Classifium, with its built-in auto-hyperparameter tuning pipeline, was found to be effective for both, binary and multi-class classification task. For binary classification Classifium outperformed all other algorithms taken for our study, with majority of data set. While performing the multi-class classification task with the Forest Cover Type data set, Classifium gave the accuracy of 0.971873%, which is the highest among the other classification accuracy from all algorithms considered in this study for comparison. Also, when the classification accuracies were tested for statistical significance, we found that Classifium had performed better than other algorithm at least with the one data set out of 7 data set.
But it required a lot of time for Classifium to generate the classification models after data pre-processing and hyperparameter tuning process. For example, with the Forest Cover Type data set, it took 260 hours for Classifium to complete its modeling.

For tuning hyperparameters of XGBoost, LightGBM and CatBoost we had done some data pre-processing. We transformed the categorical columns of all data set into numerical by applying one-hot encoding technique. Also, we replaced the missing value with a special function from Numpy, np.NAN. XGBoost and LightGBM have built-in mechanism to handle np.NAN as missing entity. For CatBoost, we placed a special string "NAN" where values were missing. CatBoost treats this string as a missing value.
Unlike in XGBoost, LightGBM and CatBoost, for handling categorical and numerical data types, it is only required in Classifium to name the columns as "nominal" or "ordinal". Also, missing values can be simply denoted by "?" character.

In the result section of this document, we can see that the testing accuracy of some algorithm with some data set is higher than the training accuracy. This was due to the stratified repeated cross validation technique. Due to this technique, only 80% of data could get trained in each iteration. For example, with the Forest Cover Type data set, we had divided the whole data set into equal two portions, training set and testing set. And we had implemented 5-Fold cross validation, repeated for only one time. Every time, 4 folds were used as training and remaining 1 fold was used for testing. So, 80% of data could get trained in each iteration. Whereas in testing, whole 100% data could get tested, yielding the better result.

From this experiment we observed that, selecting classification models, data pre-processing and manual hyperparameter tuning process following the pipeline are tedious works. It requires direct human involvement throughout the process. So, there always exists a possibility of human error. From our study we found that these limitations of manual hyperparameter tuning process can be mitigated in automatic machine learning algorithms like Classifium and H2O. Further, during experiment we found that Classifium performed better than H2O with almost all data sets. So, from the result, we concluded that we can answer our research question by stating that Classifium can be an easy solution as being automatic machine learning algorithm for classification task.

Chapter 7

Conclusion and Future Work

In this chapter, we present our conclusion about this thesis work and the possible future improvements that we observed during our research.

7.1 Conclusion

The objective of our study is to compare the classification accuracy of the automated machine learning algorithm Classifium with the manually tuned gradient boosting algorithms like XGBoost, LightGBM and CatBoost, and another automated machine learning algorithm H2O. In our experiment, we used seven data sets, from UCI, for training and testing of the five algorithms. We implemented the hyperparameter tuning pipeline for the tuning of hyperparameters of XGBoost, LightGBM, and CatBoost. We modelled the 7 data sets with our 5 algorithms and recorded their classification accuracy. While modelling, we carried out repeated stratified cross validation training for each model with the training data set. We obtained the best set of hyperparameters by fitting these models with the grid search cross validation method. Using the best set of hyperparameters, we performed tests on the testing data sets separately. We observed, after implementing with the hyperparameter tuning pipeline, that XGBoost and CatBoost performed better than LightGBM in terms of classification accuracy, but required more training time than LightGBM. In our next step, we trained and tested the automated machine learning algorithm Classifium for the same seven data sets. For reference, we also trained and tested H2O in two different approaches; a) Since H2O supports many different machine learning algorithms like Distributed Random Forest, Deep Learning, Gradient Boosting Machines, XGBoost, Navie Bayes, and et cetera, H2O models the data set with each algorithm and after, also assemble them together into a stack and perform training and testing with the stacked ensemble model. b) H2O can be run only by selecting specific algorithm as its base learner. So in our second approach we choose XGBoost for its base learner and model the data set using the hyperparameter tuning pipeline of H2O.

After training and testing of the five algorithms, we compared the classification accuracy of Classifium with the others and found that the accuracy of Classifium was

higher than the other algorithms for the majority of the 7 data sets. When compared with the XGBoost for its classification accuracy with the 7 data set, we found that Classifium outperformed XGboost with the Dota, Forest Cover Type, MiniBooNE and Porto Seguro data set. Similarly, when compared with the LightGBM, classification accuracy of Classifium was found higher with the Dota, Flavours of Physics, Forest Cover Type, and Porto Seguro data set. And again, when compared with the classification accuracy of CatBoost, we found that Classifium performed better with the Adult, Dota, Forest Cover Type, and Porto Seguro data set. Likewise we found that the classification accuracy of Classifium was better than the classification accuracy of H2O modeled with the Dota, Flavours of Physics, Forest Cover Type, MiniBooNE, and Porto Seguro data set following approach (a), implemented with its own automatic hyperparameter tuning pipeline. And also, when H2O was modeled following the approach (b), only using XGBoost for its base learner, we found that classification accuracy was lower than that of Classifium when trained and tested with the same data sets, Adult, Bank, Dota, Flavours of Physics, Forest Cover Type, MiniBooNE and Porto Seguro. From this we concluded that the hyperparameter tuning pipeline implemented in Classifium performed better than the hyperparameter tuning pipline implemented in H2O.

For further validation of the results, we also carried out statistical significance test between the obtained accuracies from Classifium with the obtained accuracies from XGBoost, LightGBM, CatBoost, and H2O with the both approaches. The results obtained from the statistical significance test show that the Classifium performed better than the other algorithms in at least one data set while performing equally in others. Based on the results from training and testing of the five algorithms, and further validated by the statistical significance test, we can conclude that the classification accuracy obtained from Classifium, an automated machine learning algorithm, can be better or equal with that from manually tuned algorithms like XGBoost, LightGBM and CatBoost.

Also, we discovered that by implementing stratified repeated K-Fold cross validation techniques, we could overcome the problem of target class imbalance. We implemented this method for every algorithm with all data sets.

7.2 Future Work

During this project, we observed that there are some aspects that can be improved in future. We realised that, since Classifium can only be used for classification task, it should also be able to work for regression task. Also, since Classifium requires a lot of time for training, it can be developed for supporting graphical processing unit.

We have found that XGBoost, LightGBM and CatBoost are implemented in many Kaggle competition in which they are placed best. So it would be more easier to implement XGBoost, LightGBM and CatBoost, if they are provided with the automatic hyperparameter tuning pipeline.

Also, we can raise a question whether the automatic hyperparameter searching algorithm like Hyperband, when implemented with gradient boosting algorithms like XGBoost, LightGBM and CatBoost, can generate the classification results comparable to the results obtained from hyperparameter tuning pipeline of Classifium? This can also be one direction for the future work.

Bibliography

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959. DOI: 10.1147/rd.33.0210.

[2] R. M. Cormack, "A review of classification," *Journal of the Royal Statistical Society: Series A (General)*, vol. 134, no. 3, pp. 321–353, 1971.

[3] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[4] S. M. Weiss and I. Kapouleas, "An empirical comparison of pattern recognition, neural nets, and machine learning classification methods.," in *IJCAI*, Citeseer, vol. 89, 1989, pp. 781–787.

[5] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 660–674, 1991. DOI: 10.1109/21.97458.

[6] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine learning, neural and statistical classification," 1994.

[7] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[8] M. D. Estebon, "Perceptrons: An associative learning network," *Virginia Tech.—1997*, 1997.

[9] T. M. Mitchell, "Does machine learning really work?" *AI magazine*, vol. 18, no. 3, pp. 11–11, 1997.

[10] L. Perelman and E. Barrett, "The Mayfield Handbook of Technical and Scientific Writing," in, 1. New York, NY, USA: McGraw-Hill, Inc., 1997.

[11] J. P. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. E. Brodley, "Pruning decision trees with misclassification costs," in *European Conference on Machine Learning*, Springer, 1998, pp. 131–136.

[12] M. Pelikan, D. E. Goldberg, E. Cantú-Paz, *et al.*, "Boa: The bayesian optimization algorithm," in *Proceedings of the genetic and evolutionary computation conference GECCO-99*, Citeseer, vol. 1, 1999, pp. 525–532.

[13] R. E. Schapire, "A brief introduction to boosting," in *Ijcai*, Citeseer, vol. 99, 1999, pp. 1401–1406.

[14] G. Piccinini, "Turing's rules for the imitation game," *Minds and Machines*, vol. 10, no. 4, pp. 573–582, 2000.

[15] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[16] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001, ISSN: 00905364. [Online]. Available: `http://www.jstor.org/stable/2699986`.

[17] ——, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001, ISSN: 00905364. [Online]. Available: `http://www.jstor.org/stable/2699986`.

[18] ——, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[19] S.-C. Wang, "Artificial neural network," in *Interdisciplinary computing in java programming*, Springer, 2003, pp. 81–100.

[20] C. Zhang and S. Zhang, *Association rule mining: models and algorithms*. Springer, 2003, vol. 2307.

[21] J. R. Quinlan, "Data mining tools see5 and c5. 0," *http://www. rulequest. com/see5-info. html*, 2004.

[22] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161–168.

[23] S. Youn and D. McLeod, "A comparative study for email classification," in *Advances and innovations in systems, computing sciences and software engineering*, Springer, 2007, pp. 387–391.

[24] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[25] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.

[26] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[27] S.-l. Pang and J.-z. Gong, "C5. 0 classification algorithm and application on individual credit evaluation of banks," *Systems Engineering-Theory & Practice*, vol. 29, no. 12, pp. 94–104, 2009.

[28] D. Steinberg and P. Colla, "Cart: Classification and regression trees," *The top ten algorithms in data mining*, vol. 9, p. 179, 2009.

[29] A. M. Turing, "Computing machinery and intelligence," in *Parsing the turing test*, Springer, 2009, pp. 23–65.

[30] Z.-H. Zhou, "Ensemble learning.," *Encyclopedia of biometrics*, vol. 1, pp. 270–273, 2009.

[31] T. O. Ayodele, "Types of machine learning algorithms," *New advances in machine learning*, vol. 3, pp. 19–48, 2010.

[32] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Information Retrieval*, vol. 13, no. 3, pp. 254–270, 2010.

[33] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*, Springer, 2011, pp. 507–523.

[34] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[35] A. Dongare, R. Kharde, and A. D. Kachare, "Introduction to artificial neural network," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 2, no. 1, pp. 189–194, 2012.

[36] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" In *International workshop on machine learning and data mining in pattern recognition*, Springer, 2012, pp. 154–168.

[37] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[38] 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*, Citeseer, vol. 13, 2013, p. 20.

[39] R. Chen and M. Lazer, "Sentiment analysis of twitter feeds for the prediction of stock market movement," *stanford edu Retrieved January*, vol. 25, p. 2013, 2013.

[40] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 2013.

[41] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 847–855.

[42] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn," in *ICML workshop on AutoML*, Citeseer, vol. 9, 2014, p. 50.

[43] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[44] T. Chen and T. He, "Higgs boson discovery with boosted trees," in *NIPS 2014 workshop on high-energy physics and machine learning*, 2015, pp. 69–80.

[45] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, *et al.*, "Xgboost: Extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, 2015.

[46] B. Park and J. K. Bae, "Using machine learning algorithms for housing price prediction: The case of fairfax county, virginia housing data," *Expert systems with applications*, vol. 42, no. 6, pp. 2928–2934, 2015.

[47] 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.

[48] Y. Zhang and A. Haghani, "A gradient boosting method to improve travel time prediction," *Transportation Research Part C: Emerging Technologies*, vol. 58, pp. 308–324, 2015.

[49] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing*, vol. 114, pp. 24–31, 2016.

[50] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.

[51] J. Brownlee. (Sep. 2016). "A gentle introduction to the gradient boosting algorithm for machine learning," [Online]. Available: `https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/`.

[52] R. H. Byrd, J. Nocedal, and F. Oztoprak, "An inexact successive quadratic approximation method for l-1 regularized optimization," *Mathematical Programming*, vol. 157, no. 2, pp. 375–396, 2016.

[53] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. arXiv: `1603.02754`. [Online]. Available: `http://arxiv.org/abs/1603.02754`.

[54] ——, "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.

[55] A. JAIN. (2016). "Hyperparameter optimization xgboost," [Online]. Available: `https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/` (visited on 04/22/2021).

[56] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[57] G. Squillero and P. Burelli, *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30–April 1, 2016, Proceedings, Part I*. Springer, 2016, vol. 9597.

[58] A. Team, "Azureml: Anatomy of a machine learning service," in *Conference on Predictive APIs and Apps*, PMLR, 2016, pp. 1–13.

[59] M. Abdar, M. Zomorodi-Moghadam, R. Das, and I.-H. Ting, "Performance analysis of classification algorithms on early detection of liver disease," *Expert Systems with Applications*, vol. 67, pp. 239–251, 2017.

[60] 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.

[61] ——, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017. [Online]. Available: `https://lightgbm.readthedocs.io/en/latest/Features.html`.

[62] ——, (2017). "Lightgbm: A highly efficient gradient boosting decision tree," [Online]. Available: `https://lightgbm.readthedocs.io/en/latest/Features.html`.

[63] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[64] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," *arXiv preprint arXiv:1706.09516*, 2017.

[65] A. Quitadadmo, J. Johnson, and X. Shi, "Bayesian hyperparameter optimization for machine learning based eqtl analysis," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2017, pp. 98–106.

[66] Ö. Çelik, "A research on machine learning methods and its applications," *Journal of Educational Technology and Online Learning*, vol. 1, no. 3, pp. 25–40, 2018.

[67] A. V. Dorogush, V. Ershov, and A. Gulin, *Catboost: Gradient boosting with categorical features support*, 2018. arXiv: `1810.11363 [cs.LG]`.

[68] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[69] P. Probst, B. Bischl, and A.-L. Boulesteix, "Tunability: Importance of hyperparameters of machine learning algorithms," *arXiv preprint arXiv:1802.09596*, 2018.

[70] S. Putatunda and K. Rama, "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of xgboost," in *Proceedings of the 2018 International Conference on Signal Processing and Machine Learning*, 2018, pp. 6–10.

[71] D. R. Schrider and A. D. Kern, "Supervised machine learning for population genetics: A new paradigm," *Trends in Genetics*, vol. 34, no. 4, pp. 301–312, 2018.

[72] D. Sisodia and D. S. Sisodia, "Prediction of diabetes using classification algorithms," *Procedia computer science*, vol. 132, pp. 1578–1585, 2018.

[73] J. Thomas, S. Coors, and B. Bischl, "Automatic gradient boosting," *arXiv preprint arXiv:1807.03873*, 2018.

[74] 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.

[75] S. K. Chinnamgari, *R Machine Learning Projects: Implement supervised, unsupervised, and reinforcement learning techniques using R 3.5*. Packt Publishing Ltd, 2019.

[76] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Auto-sklearn: Efficient and robust automated machine learning," in *Automated Machine Learning*, Springer, Cham, 2019, pp. 113–134.

[77] M. Geitle and R. Olsson, "A new baseline for automated hyper-parameter optimization," in *International Conference on Machine Learning, Optimization, and Data Science*, Springer, 2019, pp. 521–530.

[78] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[79] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[80] J. Pesantez-Narvaez, M. Guillen, and M. Alcañiz, "Predicting motor insurance claims using telematics data—xgboost versus logistic regression," *Risks*, vol. 7, no. 2, p. 70, 2019.

[81] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss, and R. Farivar, "Towards automated machine learning: Evaluation and comparison of automl approaches and tools," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019, pp. 1471–1479. DOI: 10.1109/ICTAI.2019.00209.

[82] Z. Zhang, Y. Zhao, A. Canes, D. Steinberg, O. Lyashevska, *et al.*, "Predictive analytics with gradient boosting in clinical medicine," *Annals of translational medicine*, vol. 7, no. 7, 2019.

[83] J. Hancock and T. M. Khoshgoftaar, "Performance of catboost and xgboost in medicare fraud detection," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020, pp. 572–579. DOI: 10.1109/ICMLA51294.2020.00095.

[84] E. LeDell and S. Poirier, "H2o automl: Scalable automatic machine learning," in *7th ICML workshop on automated machine learning*, 2020.

[85] E. K. Sahin, "Comparative analysis of gradient boosting algorithms for landslide susceptibility mapping," *Geocarto International*, pp. 1–25, 2020.

[86] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.

[87] J. Wang, C. Jiang, H. Zhang, Y. Ren, K. .-C. Chen, and L. Hanzo, "Thirty years of machine learning: The road to pareto-optimal wireless networks," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 1472–1514, 2020. DOI: 10.1109/COMST.2020.2965856.

[88] H. J. Weerts, A. C. Mueller, and J. Vanschoren, "Importance of tuning hyperparameters of machine learning algorithms," *arXiv preprint arXiv:2007.07588*, 2020.

[89] Wikipedia contributors, *Hyperparameter (machine learning) — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Hyperparameter_(machine_learning)&oldid=984957886, [Online; accessed 15-January-2021], 2020.

[90] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1937–1967, 2021.

[91] H2O.ai, *H2O AutoML*, H2O version 3.32.1.2, 2021. [Online]. Available: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html.

Appendix A

Hyperparameters tuning pipeline

A.1 Source code

```
def hypertune(model, param):
        cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10,
                random_state=36)
        grid_search = GridSearchCV(model, param, cv=cv,  scoring =
                              ('accuracy'), return_train_score = True,
                             verbose = 2)
        start = time.time()
        grid_search.fit(X_train, y_train)
        stop = time.time()
        sec = stop-start
        minu = sec/60
        hrs = minu/60
        print("\n")
        print('Total time required for execution:', hrs)
        print("\n")
        print('best parameters are:', grid_search.best_params_)
        print('CV accuracy:', grid_search.best_score_)
        print("\n")
        print("The cross validation results are:", grid_search.cv_results_)

hypertune(model1, params1)
//number of repetition, n_repeats, was different for different data sets.
//Criteria of finding the value for n_repeats is explained in section 3.7
//of this document.
```

Listing A.1: Implementation of RepeatedStratifiedKFold() method with gridserach() function while performing hyperparameter tuning of algorithms XGBoost LightGBM and CatBoost.

```
model1 = XGBClassifier(learning_rate = 0.3, gamma = 0,
                       max_depth = 6, min_child_weight = 1,
                       subsample =1, colsample_bytree = 1,
                       reg_lambda=1, objective ='binary:logistic',
                       tree_method='hist', scale_pos_weight=1,
                       n_estimators = 100, n_jobs = -1,
                       use_label_encoder=False, missing = np.NAN)
params1 = {'learning_rate':[0.1,0.2,0.3], 'n_estimators':
                                        range(100,550,50)}
```

Listing A.2: General model of XGBoost used for hyperparameter tuning.

```
model1 = LGBMClassifier(learning_rate=0.1, num_iteration=100,
                        max_depth = -1, num_leaves = 31,
                        min_data_in_leaf = 20, min_gain_to_split = 0,
                        bagging_fraction =1, feature_fraction = 1,
                        lambda_l1 = 0, lambda_l2 = 0,
                        scale_pos_weight =1,n_jobs = -1,
                        objective = 'binary',device_type = 'cpu')
params1 = {'learning_rate':[1, 2, 3], 'num_iteration': range(100, 550, 50)}

hypertune(model1, params1)
```

Listing A.3: General model of LightGBM used for hyperparameter tuning

```
model1 = CatBoostClassifier(iterations=1470,learning_rate=0.1,
                            cat_features = nominal, depth=7,
                           l2_leaf_reg=3.2,random_strength=0.8,
                          border_count=254,bagging_temperature = 1,
                           loss_function='Logloss', eval_metric='Accuracy')
params1 = {'learning_rate':[0.01,0.1,0.2],'iterations':range(1450, 1505, 10)}

hypertune(model1, params1)

//In the above given function, the categorical values are passed to the
//algorithm in an array of categorical feature named 'nominal'
//through a parameter called 'cat_features'.
```

Listing A.4: General model of CatBoost used for hyperparameter tuning

A.2 Hyperparameter tuning pipeline for XGBoost

A.2.1 XGBoost with the Adult data set

Hyperparameter tuning of XGBClassifier() with Adult data set.				
Steps	XGBClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and n_estimators with other default parameters			
XGBClassifier(learning_ rate=0.3,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'learning _rate': [0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	'learning _rate': 0.1, 'n_ estim ators': 150	Accuracy: 0.870209 Time: 0.36 hrs
2	Tune max_depth and min_child_weight			
XGBClassifier(learning_ rate=0.1,n_estimators = 150, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'max_ depth': range (3,10), 'min_ child_ weight': range(0, 8)	{'max_ depth': 6, 'min_ child_ weight': 0}	Accuracy: 0.87045 Time: 0.47 hrs
3	Tune gamma			

Table A.1 continued from previous page

Hyperparameter tuning of XGBClassifier() with Adult data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 150, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'gamma': [0, 1, 2, 3, 4, 5, 6]	'gamma': 0	Accuracy: 0.87045 Time: 0.03 hrs
4	Tune subsample and colsample_bytree		
XGBClassifier(learning_ rate=0.1,n_estimators = 150, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'sub sample': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1] 'col sample_ bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9, 1]	'col sample_ bytree': 0.3, 'sub sample': 1	Accuracy: 0.871785 Time: 0.77 hrs
5	Tune reg_alpha and reg_lambda		

Table A.1 continued from previous page

Hyperparameter tuning of XGBClassifier() with Adult data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 150, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 0.3, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'reg_ alpha': [0.0001, 0.001, 0.01, 0.1, 0, 1, 2, 3], 'reg_ lambda': [1,2,3, 4, 5]}	'reg_ alpha': 0, 'reg_ lambda': 1	Accuracy: 0.871785 Time: 0.09 hrs
6	Tune learning_rate		
XGBClassifier(learning_ rate=0.1,n_estimators = 150, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 0.3, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'learning _rate': [0.001, 0.01, 0.1, 0.2,0.3],	'learning _rate': 0.1	Accuracy: 0.871785 Time: 0.03 hrs
Create the final model with the value of the optimized parameters.			
model = XGBClassifier(learning_ rate=0.1,n_estimators = 150, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 0.3, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.871785 Testing Accuracy: 0.875225
The efficiency of the model with testing data was found to be: 0.875225			

Table A.1 continued from previous page

Hyperparameter tuning of XGBClassifier() with Adult data set.

Table A.1: Hyperparameter tuning pipeline for the XGBClassifier() with the Adult data set.

A.2.2 XGBoost with the Bank data set

Hyperparameter tuning of XGBClassifier() with Bank data set.				
Steps	XGBClassifier()	Parameter Grid	Optimized Value	CV Training:
1	Tune learning_rate and n_estimators with other default parameters			
XGBClassifier(learning_ rate=0.3,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'learning _rate': [0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	'learning _rate': 0.1, 'n_ estim ators': 100	Accuracy: 0.915651 Time: 0.32 hrs
2	Tune max_depth and min_child_weight			
XGBClassifier(learning_ rate=0.1,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'max_ depth': range (3,10), 'min_ child_ weight': range(0, 8)	'max_ depth': 4, 'min_ child_ weight': 2	Accuracy: 0.916734 Time: 0.32 hrs
3	Tune gamma			
XGBClassifier(learning_ rate=0.1,n_estimators = 100, max_depth = 4, min_child weight = 2, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'gamma': [0, 1, 2, 3, 4, 5, 6]	'gamma': 2	Accuracy: 0.916758 Time: 0.02

Table A.2 continued from previous page

Hyperparameter tuning of XGBClassifier() with Bank data set.			
4	Tune subsample and colsample_bytree		
XGBClassifier(learning_ rate=0.1,n_estimators = 100, max_depth = 4, min_child weight = 2, gamma=2, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'sub sample': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1] 'col sample_ bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9, 1]	'sub sample': 0.9, 'col sample_ bytree': 1	Accuracy: 0.916972 Time: 0.29 hrs
5	Tune reg_alpha and reg_lambda		
XGBClassifier(learning_ rate=0.1,n_estimators = 100, max_depth = 4, min_child weight = 2, gamma=2, subsample = 0.9, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'reg_ alpha': [0.0001, 0.001, 0.01, 0.1, 0, 1, 2, 3], 'reg_ lambda': [1,2,3, 4, 5]}	'reg_ alpha': 0.01, 'reg_ lambda': 1	Accuracy: 0.917015 Time: 0.12 hrs
6	Tune learning_rate		

Table A.2 continued from previous page

Hyperparameter tuning of XGBClassifier() with Bank data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 100, max_depth = 4, min_child weight = 2, gamma=2, subsample = 0.9, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'learning _rate': [0.001, 0.01, 0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	'learning _rate': 0.1, 'n_ estim ators': 100	Accuracy: 0.917015 Time: 0.19 hrs
Create the final model with the value of the optimized parameters.			
XGBClassifier(learning_ rate=0.1,n_estimators = 100, max_depth = 4, min_child weight = 2, gamma=2, subsample = 0.9, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))	CV Training Accuracy: 0.917015 Testing Accuracy: 0.916525	
The efficiency of the model with testing data was found to be: 0.916525			

Table A.2: Hyperparameter tuning pipeline for the XGBClassifier() with the Bank data set.

A.2.3 XGBoost with the Dota data set

Hyperparameter tuning of XGBClassifier() with Dota data set.				
Steps	XGBClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and n_estimators with other default parameters			
XGBClassifier(learning_ rate=0.3,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'learning _rate': [0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	'learning _rate': 0.1, 'n_ estim ators': 200	Accuracy: 0.584595
2	Tune max_depth and min_child_weight			
XGBClassifier(learning_ rate=0.1,n_estimators = 200, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'max_ depth': range (3,10), 'min_ child_ weight': range(0, 8)	'max_ depth': 4, 'min_ child_ weight': 4	Accuracy: 0.587201
3	Tune gamma			
XGBClassifier(learning_ rate=0.1,n_estimators = 200, max_depth = 4, min_child weight = 4, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'gamma': [0, 1, 2, 3, 4, 5, 6]	'gamma': 1	Accuracy: 0.587338

Table A.3 continued from previous page

Hyperparameter tuning of XGBClassifier() with Dota data set.			
4	Tune subsample and colsample_bytree		
XGBClassifier(learning_ rate=0.1,n_estimators = 200, max_depth = 4, min_child weight = 4, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'sub sample': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1] 'col sample_ bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9, 1]}	'sub sample': 0.4 'col sample_ bytree': 0.8,	Accuracy: 0.5921801
5	Tune reg_alpha and reg_lambda		
XGBClassifier(learning_ rate=0.1,n_estimators = 200, max_depth = 4, min_child weight = 4, gamma=0, subsample = 0.4, colsample_bytree = 0.8, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'reg_ alpha': [0.0001, 0.001, 0.01, 0.1, 0, 1, 2, 3], 'reg_ lambda': [1,2,3, 4, 5]}	'reg_ alpha': 3, 'reg_ lambda': 4	Accuracy: 0.59347979
5.1	Re-tune reg_alpha andreg_lambda		

Table A.3 continued from previous page

Hyperparameter tuning of XGBClassifier() with Dota data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 200, max_depth = 4, min_child weight = 4, gamma=0, subsample = 0.4, colsample_bytree = 0.8, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'reg_ alpha': [4,5, 6, 7, 8, 9, 10], 'reg_ lambda': [1,2,3,4, 5,6]	'reg_ alpha': 6, 'reg_l ambda': 2	Accuracy: 0.593530
6	Tune learning_rate and n_estimators		
XGBClassifier(learning_ rate=0.1,n_estimators = 200, max_depth = 4, min_child weight = 4, gamma=0, subsample = 0.4, colsample_bytree = 0.8, reg_alpha=6, reg_lambda=2, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'learning _rate': [0.001, 0.01, 0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	'learning _rate': 0.1, 'n_ estim ators': 500	Accuracy: 0.59606
6.1	Re-tune learning_rate and n_estimators		
XGBClassifier(learning_ rate=0.1,n_estimators = 200, max_depth = 4, min_child weight = 4, gamma=0, subsample = 0.4, colsample_bytree = 0.8, reg_alpha=6, reg_lambda=2, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'learning _rate': [0.001, 0.01, 0.1, 0.2, 0.3], 'n_ estim ators': range(450, 1050, 50)	'learning _rate': 0.1, 'n_ estimators': 600	0.5962794

Table A.3 continued from previous page

<table>
<tr><td colspan="3">Hyperparameter tuning of XGBClassifier()
with Dota data set.</td></tr>
<tr><td colspan="3">Create the final model
with the value of the optimized parameters.</td></tr>
<tr><td>XGBClassifier(learning_
rate=0.1,n_estimators
= 600, max_depth = 4,
min_child weight = 4,
gamma=0, subsample = 0.4,
colsample_bytree = 0.8,
reg_alpha=6, reg_lambda=2,
missing=np.NAN, tree_method=
'gpu_hist',
objective = 'binary:logistic')</td><td>model.fit(X_train,
y_train)

y_predict = model.
predict
(X_test)

print('Testing accuracy
score:',
accuracy_score
(y_test, y_predict))</td><td>CV
Training
Accuracy:
0.5962794

Testing
Accuracy:

0.594859</td></tr>
<tr><td colspan="3">The testing accuracy of the model with testing data set was obtained as:
0.59485924</td></tr>
</table>

Table A.3: Hyperparameter tuning pipeline for the XGBClassifier() with the Dota data set.

A.2.4 XGBoost with the Flavours of Physics data set.

Hyperparameter tuning of XGBClassifier() with Flavours of Physics data set.				
Steps	XGBClassifier()	Parameter Grid	Optimized Value	CV Training:
1	Tune learning_rate and n_estimators with other default parameters			
	XGBClassifier(learning_ rate=0.3,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'learning _rate': [0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	learning _rate: 0.1 n_ estim ators: 450	Accuracy: 0.89104 Time: 0.5 hrs
2	Tune max_depth and min_child_weight			
	XGBClassifier(learning_ rate=0.1,n_estimators = 450, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'max_ depth': range (3,10), 'min_ child_ weight': range(0, 8)	'max_ depth': 5, 'min_ child_ weight': 7	Accuracy: 0.89209 Time: 1.85 hrs
2.1	Re-tune min_child_weight			
	XGBClassifier(learning_ rate=0.1,n_estimators = 450, max_depth = 5, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'min_ child_ weight': range(5, 10)	'min_ child_ weight': 7	Accuracy: 0.89209 Time: 0.07 hrs

Table A.4 continued from previous page

<table>
<tr><td colspan="5">Hyperparameter tuning of XGBClassifier()
with Flavours of Physics data set.</td></tr>
<tr><td>3</td><td colspan="4">Tune
gamma</td></tr>
<tr><td colspan="2">XGBClassifier(learning_
rate=0.1,n_estimators
= 450, max_depth = 5,
min_child weight = 7,
gamma=0, subsample = 1,
colsample_bytree = 1,
reg_alpha=0, reg_lambda=1,
missing=np.NAN, tree_method=
'gpu_hist',
objective = 'binary:logistic')</td><td>'gamma':
[0, 1, 2,
3,
4, 5, 6]</td><td>'gamma':
0</td><td>Accuracy:
0.89209
Time:
0.06 hrs</td></tr>
<tr><td>4</td><td colspan="4">Tune subsample and
colsample_bytree</td></tr>
<tr><td colspan="2">XGBClassifier(learning_
rate=0.1,n_estimators
= 450, max_depth = 5,
min_child weight = 7,
gamma=0, subsample = 1,
colsample_bytree = 1,
reg_alpha=0, reg_lambda=1,
missing=np.NAN, tree_method=
'gpu_hist',
objective = 'binary:logistic')</td><td>{'sub
sample':
[0.1, 0.2,
0.3, 0.4,
0.5, 0.6,
0.7, 0.8,
0.9,1]

'col
sample_
bytree':
[0.1, 0.2,
0.3, 0.4,
0.5, 0.6,
0.7,0.8,
0.9, 1]</td><td>'sub
sample':
0.9

'col
sample_
bytree':
0.9,</td><td>Accuracy:
0.89297
Time:
1.30 hrs</td></tr>
<tr><td>5</td><td colspan="4">Tune reg_alpha and
reg_lambda</td></tr>
</table>

Table A.4 continued from previous page

Hyperparameter tuning of XGBClassifier() with Flavours of Physics data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 450, max_depth = 5, min_child weight = 7, gamma=0, subsample = 0.9, colsample_bytree = 0.9, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'reg_ alpha': [0.0001, 0.001, 0.01, 0.1, 0, 1, 2, 3], 'reg_ lambda': [1,2,3, 4, 5]}	'reg_ alpha': 2, 'reg_ lambda': 5	Accuracy: 0.89379 Time: 0.65 hrs
5.1	Re-tune reg_alpha and reg_lambda		
XGBClassifier(learning_ rate=0.1,n_estimators = 450, max_depth = 5, min_child weight = 7, gamma=0, subsample = 0.9, colsample_bytree = 0.9, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'reg_ alpha': [0.0001, 0.001, 0.01, 0.1, 0, 1, 2, 3], 'reg_ lambda': [1,2,3, 4, 5, 6, 7, 8]	'reg_ alpha': 2, 'reg_ lambda': 5	Accuracy: 0.89379 Time: 1.0 hrs
6	Tune learning_rate and n_estimators		

Table A.4 continued from previous page

Hyperparameter tuning of XGBClassifier() with Flavours of Physics data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 450, max_depth = 5, min_child weight = 7, gamma=0, subsample = 0.9, colsample_bytree = 0.9, reg_alpha=2, reg_lambda=5, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'learning _rate': [0.001, 0.01, 0.1, 0.2,0.3], 'n_ estim ators': range(100, 600, 50)	'learning _rate': 0.1, 'n_ estim ators': 550	Accuracy: 0.89383 Time: 0.61 hrs
6.1	Re-tune n_estimators		
XGBClassifier(learning_ rate=0.1,n_estimators = 450, max_depth = 5, min_child weight = 7, gamma=0, subsample = 0.9, colsample_bytree = 0.9, reg_alpha=2, reg_lambda=5, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'n_ estimators': range(450, 850, 50)	'n_ estimators': 600	Accuracy: 0.89390 Time: 0.18 hrs
Create the final model with the value of the optimized parameters.			
XGBClassifier(learning_ rate=0.1,n_estimators = 600, max_depth = 5, min_child weight = 7, gamma=0, subsample = 0.9, colsample_bytree = 0.9, reg_alpha=2, reg_lambda=5, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		Training Accuracy: 0.89390 Testing Accuracy: 0.890895
The testing accuracy of the model with testing data set is: 0.890895			

Table A.4 continued from previous page

Hyperparameter tuning of XGBClassifier() with Flavours of Physics data set.

Table A.4: Hyperparameter tuning pipeline for the XGBClassifier() with the Flavours of Physics data set.

A.2.5 XGBoost with the Forest Cover Type data set

Hyperparameter tuning of XGBClassifier() with Forest Cover Type data set.				
Steps	XGBClassifier()	Parameter Grid	Optimized Value	CV Training:
1	Fix learning_rate at 0.1 and tune n_estimators with other default parameters			
	XGBClassifier(learning_ rate=0.1,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	n_ estim ator: range(100, 5050, 50)	5000	Accuracy: 0.945976 Time: 28 hrs
2	Tune max_depth and min_child_weight			
	XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	'max_ depth': range (3,7), 'min_ child_ weight': range(0, 8)	6 0	Accuracy: 0.954806 Time: 3.4 hrs
2.1	Re-tune max_depth and min_child_weight			
	XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	max_ depth: range (6,11), min_ child_ weight: range (0,3)	10 0	Accuracy: 0.960744 Time: 10.9 hr
2.2	Re-tune max_depth			

Table A.5 continued from previous page

Hyperparameter tuning of XGBClassifier() with Forest Cover Type data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	max_ depth: range (10, 13)	12	Accuracy: 0.961453 Time: 7.5 hr
2.3	Re-tune max_depth		
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	max_ depth: range (12, 15)	14	Accuracy: 0.962062 Time: 11 hrs
2.4	Re-tune max_depth		
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	max_ depth: range (15, 19)	18	Accuracy: 0.962193 Time: 17 hrs
2.5	Re-tune max_depth		

Table A.5 continued from previous page

Hyperparameter tuning of XGBClassifier() with Forest Cover Type data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	max_ depth: range(19, 20)	19	Accuracy: 0.962276 Time: 7 hrs
2.6	Re-tune max_depth		
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 6, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	max_ depth: range(20, 23)	20	Accuracy: 0.962262 Time: 21 hrs
3	Tune gamma		
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 20, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	gamma: [0, 1, 2]	0	Accuracy: 0.962276 Time: 8 hrs
3.1	Re-tune gamma		

Table A.5 continued from previous page

Hyperparameter tuning of XGBClassifier() with Forest Cover Type data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 20, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	gamma: [0, 0.5, 0.75]	0	Accuracy: 0.962276 Time: 8 hrs
4	Tune subsample and colsample_bytree		
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 20, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	sub sample: [0.8,0.9 ,1], col sample_ bytree: [0.8, 0.9,1]	0.9 0.9	Accuracy: 0.963246 Time: 10.4 hrs
4.1	Re-tune subsample and colsample_bytree		
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 20, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	sub sample: [0.9, 0.95], col sample_ bytree: [0.9, 0.95]	0.95 0.95	Accuracy: 0.963683 Time: 7.5 hhrs
5	Tune reg_alpha and reg_lambda		

Table A.5 continued from previous page

Hyperparameter tuning of XGBClassifier() with Forest Cover Type data set.			
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 20, min_child weight =0 , gamma=0, subsample = 0.95, colsample_bytree = 0.95, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	reg_ alpha: [0.01, 0.1, 0], reg_ lambda: [0.01, 0.1]	0 0.1	Accuracy: 0.963794 Time: 17.5 hrs
6	Tune learning_rate		
XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 20, min_child weight =0 , gamma=0, subsample = 0.95, colsample_bytree = 0.95, reg_alpha=0, reg_lambda=0.1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'multi:softprob')	learning _rate: [0.01, 0.1, 0.2]	0.1	Accuracy: 0.963794 Time: 8.5 hrs
Create the final model with the value of the optimized parameters.			
model = XGBClassifier(learning_ rate=0.1,n_estimators = 5000, max_depth = 20, min_child weight =0 , gamma=0, subsample = 0.95, colsample_bytree = 0.95, reg_alpha=0, reg_lambda=0.1, missing=np.NAN, tree_method= 'gpu_hist', objective = ' multi:softprob ')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.963794 Testing Accuracy: 0.968155
The efficiency of the model with testing data was found to be: 0.968155			

Table A.5 continued from previous page

Hyperparameter tuning of XGBClassifier() with Forest Cover Type data set.

Table A.5: Hyperparameter tuning pipeline for the XGBClassifier() with the Forest Cover Type data set.

A.2.6 XGBoost with the MiniBooNE data set.

Hyperparameter tuning of XGBClassifier() with MiniBooNE data set				
Steps	XGBClassifier()	Parameters Grid	Optimized Value	CV Training
1	Tune learning_rate and n_estimators with other default parameters			
XGBClassifier(learning_ rate=0.3,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'learning _rate': [0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	learning _rate: 0.2 n_ estim ators: 500	Accuracy: 0.944714 Time: 0.60 hrs
1.1	Re-tune n_estimators			
XGBClassifier(learning_ rate=0.2,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		n_ estim ators: range(450, 900, 50)	n_ estim ators: 850	Accuracy: 0.94532 Time: 1.18 hrs
1.2	Re-tune n_estimators			
XGBClassifier(learning_ rate=0.2,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'n_esti mators': range(800, 1550, 50)	'n_esti mators': 1300	Accuracy: 0.94571 Time: 0.97 hrs

Table A.6 continued from previous page

Hyperparameter tuning of XGBClassifier() with MiniBooNE data set			
2	Tune max_depth and min_child_weight		
XGBClassifier(learning_ rate=0.2,n_estimators = 1300, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'max_ depth': range (3,10), 'min_ child_ weight': range(0, 8)	'max_ depth': 7, 'min_ child_ weight': 0	Accuracy: 0.946596 Time: 4.38 hrs
3	Tune gamma		
XGBClassifier(learning_ rate=0.2,n_estimators = 1300, max_depth = 7, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'gamma' : [0, 1, 2, 3, 4, 5, 6]	'gamma': 0	Accuracy: 0.946596 Time: 0.30 hrs
4	Tune subsample and colsample_bytree		

Table A.6 continued from previous page

Hyperparameter tuning of XGBClassifier() with MiniBooNE data set			
XGBClassifier(learning_ rate=0.2,n_estimators = 1300, max_depth = 7, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'sub sample': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1] 'col sample_ bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9, 1]	'sub sample': 1 'col sample_ bytree': 1,	Accuracy: 0.946596 Time: 14.72
5	Tune reg_alpha and reg_lambda		
XGBClassifier(learning_ rate=0.2,n_estimators = 1300, max_depth = 7, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	{'reg_ alpha': [0.0001, 0.001, 0.01, 0.1, 0, 1, 2, 3], 'reg_ lambda': [1,2,3, 4, 5]}	'reg_ alpha': 0, 'reg_ lambda': 1	Accuracy: 0.946596 Time: 5.59 hrs
6	Tune learning_rate and n_estimators		

Table A.6 continued from previous page

Hyperparameter tuning of XGBClassifier() with MiniBooNE data set			
XGBClassifier(learning_ rate=0.2,n_estimators = 1300, max_depth = 7, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'learning _rate': [0.001, 0.01, 0.1, 0.2,0.3], n_esti mators: range(1200, 1400, 50)	'learning _rate': 0.2 n_esti mators: 1300	Accuracy: 0.94659 Time: 7 hrs
Create the final model with the value of the optimized parameters.			
model = XGBClassifier(learning_ rate=0.2,n_estimators = 1300, max_depth = 7, min_child weight = 0, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		Training Accuracy: 0.946596 Testing Accuracy: 0.946856
The efficiency of the model with testing data was found to be: 0.946856			

Table A.6: Hyperparameter tuning pipeline for the XGBClassifier() with the MiniBooNE data set.

A.2.7 XGBoost with the Porto Seguro data set.

Hyperparameter tuning of XGBClassifier() with Porto Seguro data set				
Steps	XGBClassifier()	Parameter Grid	Optimized Value	CV Training:
1	Tune learning_rate and n_estimators with other default parameters			
XGBClassifier(learning_ rate=0.3,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'learning _rate': [0.1, 0.2, 0.3], 'n_ estim ators': range(100, 550, 50)	0.01 350	Accuracy: 0.588497 Time: 0.5 hrs
1.1	Re-tune n_estimators and learning_rate			
XGBClassifier(learning_ rate=0.3,n_estimators = 100, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		n_esti mator: range(300, 400,10) learning _rate: [0.001, 0.01]	350 0.01	Accuracy: 0.588497 Time: 0.47 hrs
2	Tune max_depth and min_child_weight			
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 6, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')		'max_ depth': range (3,10), 'min_ child_ weight': range(0, 3)	4 1	Accuracy: 0.590499 Time: 0.6 hrs

Table A.7 continued from previous page

Hyperparameter tuning of XGBClassifier() with Porto Seguro data set			
3	Tune gamma		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'gamma': [0, 1, 2, 3, 4, 5, 6]	0	Accuracy: 0.590499 Time: 0.03 hrs
3.1	Re-tune gamma		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	gamma: [0.001, 0.1 ,0.02, 0.2]	0.2	Accuracy: 0.590591 Time: 0.04 hrs
3.2	Re-tune gamma		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	gamma: [0.2, 0.25,0.3, 0.4]	0.3	Accuracy: 0.590596 Time: 0.03 hrs
3.3	Re-tune gamma		

Table A.7 continued from previous page

Hyperparameter tuning of XGBClassifier() with Porto Seguro data set			
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	gamma: [0.3,0.35, 0.375]	0.3	Accuracy: 0.590596 Time: 0.03 hrs
4	Re-tune n_estimators		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	n_esti mator: range(300, 400,10)	350	Accuracy: 0.590596 Time: 0.09 hrs
4.1	Re-tune n_estimators		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	n_esti mator: range(340, 360,2)	350	Accuracy: 0.590596 Time: 0.09 hrs
5	Tune subsample and colsample_bytree		

Table A.7 continued from previous page

Hyperparameter tuning of XGBClassifier() with Porto Seguro data set			
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'sub sample': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1] 'col sample_ bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9, 1]	0.5 0.4	Accuracy: 0.593188 Time: 0.8 hrs
5.1	Re-tune subsample and colsample_bytree		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 1, colsample_bytree = 1, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	sub sample: [0.45, 0.5, 0.55], col sample_ bytree: [0.35, 0.4, 0.45]	0.5 0.4	Accuracy: 0.593188 Time: 0.07 hrs
6	Tune reg_alpha and reg_lambda		

Table A.7 continued from previous page

Hyperparameter tuning of XGBClassifier() with Porto Seguro data set			
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 0.5, colsample_bytree = 0.4, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'reg_ alpha': [0,0.1, 0.2, 0.3, 0.0001, 0.0002, 0.0003, 0.0004, 1], 'reg_ lambda': [0.01, 0.1,1,2, 1.5]	0.2 1.5	Accuracy: 0.593485 Time: 0.42 hrs
6.1	Re-tune reg_alpha and reg_lambda		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 0.5, colsample_bytree = 0.4, reg_alpha=0, reg_lambda=1, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	'reg_ alpha': [0.2,0.23, 0.25, 0.27], 'reg_ lambda': [1.2,1.3, 1.5, 1.7,1.9]	0.2 1.5	Accuracy: 0.593485 Time: 0.37 hrs
7	Tune n_estimators and learning_rate		
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 0.5, colsample_bytree = 0.4, reg_alpha=0.2, reg_lambda=1.5, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	n_esti mator: range(300, 450,25) learning _rate: [0.001, 0.01, 0.1,0.2]	425 0.01	Accuracy: 0.593772 Time: 0.19 hrs
7.1	Re-tune n_estimators		

Table A.7 continued from previous page

Hyperparameter tuning of XGBClassifier() with Porto Seguro data set			
XGBClassifier(learning_ rate=0.01,n_estimators = 350, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 0.5, colsample_bytree = 0.4, reg_alpha=0.2, reg_lambda=1.5, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	n_esti mator: range(425, 500,5)	450	Accuracy: 0.594033
Create the final model with the value of the optimized parameters.			
XGBClassifier(learning_ rate=0.01,n_estimators = 450, max_depth = 4, min_child weight = 1, gamma=0.3, subsample = 0.5, colsample_bytree = 0.4, reg_alpha=0.2, reg_lambda=1.5, missing=np.NAN, tree_method= 'gpu_hist', objective = 'binary:logistic')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))	Training Accuracy: 0.594033 Testing Accuracy: 0.595298	
The accuracy of the model on the test data set was found to be: 0.595298			

Table A.7: Hyperparameter tuning pipeline for the XGBClassifier() with the Porto Seguro data set.

A.3 Hyperparameter tuning pipeline for LightGBM

A.3.1 LightGBM with the Adult data set

Hyperparameter tuning of LGBMClassifier() with Adult data set				
Steps	LGBMClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1, 0.2, 0.3], 'num_itera tion': range(100, 550, 50)	0.1 100	Accuracy: 0.869656 Time: 0.6 hrs
1.1	Re-tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.01, 0.1], 'num_itera tion': range(50, 150, 10)	0.1 80	Accuracy: 0.870101 Time: 0.17 hrs
1.2	Re_tune num_teration			

Table A.8 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Adult data set			
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_iteration': range(75, 85)	75	Accuracy: 0.870147 Time: 0.07 hrs
2	Tune max_depth and min_gain_to_split		
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_depth': [10, 20, 30], 'min_gain_to_split': [0, 1, 2],	20 1	Accuracy: 0.870505 Time: 0.04 hrs
2.1			
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_depth': range(15,25), 'min_gain_to_split': [0.8,0.9, 1,1.1,1.2],	17 1	Accuracy: 0.870556 Time: 0.43 hrs
3	Tune min_data_in_leaf and num_leaves		

Table A.8 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Adult data set			
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = 17, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_ data_ in_leaf': [10,20, 30,40] 'num_ leaves': [10,20, 30,40]	20 30	Accuracy: 0.870428 Time: 0.09 hrs
3.1	Re-tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = 17, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_ data_ in_leaf': range(15,25), 'num_ leaves': (25,35)	17 32	Accuracy: 0.870715 Time: 0.67 hrs
4	Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = 17, num_leaves = 32, min_data_in_leaf = 17, min_gain_to_split = 1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], feature_ fraction: [0.1, 0.2, 0.3, 04, 0.5, 0.6, 0.7, 0.8, 0.9, 1]	0.1 0.4	Accuracy: 0.871334 Time: 0.7 hrs

Table A.8 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Adult data set			
4.1	Re-Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = 17, num_leaves = 32, min_data_in_leaf = 17, min_gain_to_split = 1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'bagging_ fraction': [0.05, 0.1, 0.15] , 'feature_ fraction': [0.35, 0.4, 0.45]	0.05 0.4	Accuracy: 0.871334 Time: 0.05 hrs
5	Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = 17, num_leaves = 32, min_data_in_leaf = 17, min_gain_to_split = 1, bagging_fraction =0.05, feature_fraction = 0.4, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0,1,2,3,4], lambda_l2: [0,1,2,3,4]	0 0	Accuracy: 0.871334 Time: 0.09 hrs
5.1	Re-Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = 17, num_leaves = 32, min_data_in_leaf = 17, min_gain_to_split = 1, bagging_fraction =0.05, feature_fraction = 0.4, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0.01,0.1, 0], lambda_l2: [0.01,0.1, 0]	0 0	Accuracy: 0.871334 Time: 0.05 hrs
Create the final model with the value of the optimized parameters.			

Table A.8 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Adult data set		
LGBMClassifier(learning _rate=0.1, num_iteration=75, max_depth = 17, num_leaves = 32, min_data_in_leaf = 17, min_gain_to_split = 1, bagging_fraction =0.05, feature_fraction = 0.4, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))	CV Training Accuracy: 0.871334 Testing Accuracy: 0.875389
The accuracy of the model with testing data was found to be: 0.875389		

Table A.8: Hyperparameter tuning pipeline for the LGBMClassifier() the Adult data set.

A.3.2 LightGBM with the Bank data set

Hyperparameter tuning of LGBMClassifier() with the Bank data set				
Steps	LGBMClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1, 0.2, 0.3], 'num_itera tion': range(100, 550, 50)	0.1 100	Accuracy: 0.869416 Time: 0.18 hrs
1.1	Re-Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1, 0.2, 0.3, 1] 'n_esti mators': range(50, 250, 25)	0.1 50	Accuracy: 0.916117 Time: 0.4 hrs
1.2	Re-Tune learning_rate and num_iteration			

Table A.9 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Bank data set			
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.001, 0.01, 0.1] 'n_esti mators': range(40, 60, 2)	0.1 40	Accuracy: 0.916369 Time: 0.2 hrs
1.3	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'n_esti mators': range(35, 45)	39	Accuracy: 0.916524 Time: 0.04 hrs
2	Tune max_depth and min_gain_to_split		
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [10, 20, 30], 'min_ gain_ to_ split': [0, 1, 2]	20 2	Accuracy: 0.916709 Time: hrs
2.1	Re-Tune max_depth and min_gain_to_split		

Table A.9 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Bank data set			
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': range(10, 25), 'min_ gain_ to_split': [1.8, 1.9, 2, 2.1,2.2]	11 2.1	Accuracy: 0.916801 Time: 0.02 hrs
3	Tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = 11, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 2.1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data _in_leaf': [10,20, 30, 40], 'num_ leaves': [10,20, 30,40],	30 20	Accuracy: 0.917073 Time: 0.05 hrs
3.1	Re-Tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = 11, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 2.1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data _in_leaf': range(25, 36), 'num_l eaves': range(15, 26)	26 18	Accuracy: 0.917297 Time: 0.39 hrs
4	Tune bagging_fraction and feature_fraction		

Table A.9 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Bank data set			
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = 11, num_leaves = 18, min_data_in_leaf = 26, min_gain_to_split = 2.1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.1, 0.25, 0.5, 0.75, 1], feature_ fraction: [0.1, 0.25, 0.5, 0.5, 0.75, 1]	0.1 1	Accuracy: 0.917297 Time: 0.078 hrs
4.1	Re-Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = 11, num_leaves = 18, min_data_in_leaf = 26, min_gain_to_split = 2.1, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.05, 0.1, 0.15 0.2, 0.25], feature_ fraction: [0.8, 0.85, 0.9, 0.95, 1]	0.05 1	Accuracy: 0.917297 Time: 0.08 hrs
5	Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = 11, num_leaves = 18, min_data_in_leaf = 26, min_gain_to_split = 2.1, bagging_fraction =0.05, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0, 1, 2, 3], lambda_l2: [0, 1, 2, 3]	0 0	Accuracy: 0.917297 Time: 0.04 hrs
5.1	Re-Tune lambda_l1 and lambda_l2		

Table A.9 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Bank data set			
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = 11, num_leaves = 18, min_data_in_leaf = 26, min_gain_to_split = 2.1, bagging_fraction =0.05, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0, 0.1, 0.2, 0.3], lambda_l2: [0, 0.1, 0.2, 0.3]	0 0	Accuracy: 0.917297 Time: 0.04 hrs
Create the final model with the value of the optimized parameters.			
LGBMClassifier(learning _rate=0.1, num_iteration=39, max_depth = 11, num_leaves = 18, min_data_in_leaf = 26, min_gain_to_split = 2.1, bagging_fraction =0.05, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.917297 Testing Accuracy: 0.915893
The accuracy of the model with testing data was found to be: 0.915893			

Table A.9: Hyperparameter tuning pipeline for the LGBMClassifier() with the Bank data set.

A.3.3 LightGBM with the Dota data set.

Hyperparameter tuning of LGBMClassifier() with the Dota data set				
Steps	LGBMClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1, 0.2, 0.3] 'num_itera tion': range(100, 550, 50)	0.1 100	Accuracy: 0.587996 Time: 0.42 hrs
1.1	Re-Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.001, 0.01, 0.1] 'num_itera tion': range(80, 150, 10)	0.1 100	Accuracy: 0.587996 Time: 0.15 hrs
2	Tune max_depth and min_gain_to_split			

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set			
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [10, 20, 30], 'min_gain _to_split': [0, 1, 2]	20 1	Accuracy: 0.590512 Time: 0.05 hrs
2.1	Re-Tune max_depth and min_gain_to_split		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': range(16, 30, 2) 'min_gain _to_split': [0.8, 0.9,1, 1.1, 1.2]	29 1.2	Accuracy: 0.591230 Time: 0.27 hrs
2.2	Re-Tune max_depth and min_gain_to_split		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': range(29, 35) 'min_gain _to_split': [1.2,1.3, 1.4,1.5]	29 1.2	Accuracy: 0.591230 Time: 0.1 hrs
3	Tune min_data_in_leaf and num_leaves		

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set			
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 1.2, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data _in_leaf': [10,20, 30, 40], 'num_ leaves': [10,20, 30,40],	40 40	Accuracy: 0.591541 Time: 0.09 hrs
3.1	Re-Tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 1.2, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data _in_leaf': range(34, 46,2), 'num_ leaves': range(34, 46,2)	44 40	Accuracy: 0.591677 Time: 0.22 hrs
4	Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction: [0.1, 0.2, 0.3,0.4, 0.5, 0.6, 0.7,0.8, 0.9,1], feature_ fraction: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9,1]	0.1 0.3	Accuracy: 0.592566 Time: 0.7 hrs

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set			
4.1	Re-Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction: [0.05,0.1, 0.15], feature_ fraction: [0.25, 0.3, 0.35]	0.05 0.3	Accuracy: 0.592567 Time: 0.06 hrs
5	Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0, 1, 2, 3], lambda_l2: [0, 1, 2, 3]	1 1	Accuracy: 0.593052 Time: 0.11 hrs
5.1	Re-Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0.8,0.9, 1, 0.1, 1.2], lambda_l2: [0.8,0.9,1, 0.1, 1.2],	0.9 0.8	Accuracy: 0.593154 Time:
5.2	Re-Tune lambda_l2		

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set			
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l2: [0.6, 0.7, 0.8]	0.8	Accuracy: 0.593154 Time: 0.03 hrs
Re-start the whole tuning process from step 1, but with the LGBMClassifier() function with optimized values from step 5.2.			
6	Re-Tune learning_rate and num_iteration		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1, 0.2, 0.3] 'num_itera tion': range(100, 550, 50)	0.1 150	Accuracy: 0.593193 Time: 0.19 hrs
6.1	Re-Tune learning_rate and num_iteration		
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.001, 0.0.01, 0.1] 'num_itera tion': range(100, 150, 5)	0.1 115	Accuracy: 0.593300 Time: 0.26 hrs

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set			
7	Re-Tune max_depth, min_gain_to_split, min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 29, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 1.2, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': range(15, 30, 2), 'min_gain_ to_split': [0.8,0.9, 1,1.1, 1.2, 2, 2.5, 3], 'min_data_ in_leaf': [10,20, 30, 40], 'num_ leaves': [10,20, 30,40]	19, 0.9, 30, 20	Accuracy: 0.594256 Time: 7 hrs
7.1	Re-Tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 19, num_leaves = 40, min_data_in_leaf = 44, min_gain_to_split = 0.9, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data _in_leaf': range(24, 36,2) 'num_ leaves': (14, 26,2)	30 20	Accuracy: 0.594256 Time: 0.2 hrs
8	Re-Tune bagging_fraction and feature_fraction		

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set			
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 19, num_leaves = 20, min_data_in_leaf = 30, min_gain_to_split = 0.9, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9,1], feature_ fraction: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,0.8, 0.9,1]	0.1 0.3	Accuracy: 0.594256 Time: 0.6 hrs
8.1	Re-Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 19, num_leaves = 20, min_data_in_leaf = 30, min_gain_to_split = 0.9, bagging_fraction =0.05, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction: [0.05, 0.1, 0.15], feature_ fraction: [0.25, 0.3, 0.35]	0.1 0.3	Accuracy: 0.594256 Time: 0.05 hrs
9	Re-Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 19, num_leaves = 20, min_data_in_leaf = 30, min_gain_to_split = 0.9, bagging_fraction =0.1, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0.8,0.9, 1, 0.1, 1.2, 2, 3], lambda_l2: [0.6, 0.7, 0.8,0.9, 1, 0.1, 1.2, 1.5, 2, 3]	0.9 0.8	Accuracy: 0.594256 Time: 0.25 hrs

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set			
10	Re-Tune learning_rate and num_iteration		
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 19, num_leaves = 20, min_data_in_leaf = 30, min_gain_to_split = 0.9, bagging_fraction =0.1, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0.8, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1, 0.2, 0.3] 'num_itera tion': range(115, 550, 50)	0.1 165	Accuracy: 0.594407 Time: 0.2 hrs
10.1	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 19, num_leaves = 20, min_data_in_leaf = 30, min_gain_to_split = 0.9, bagging_fraction =0.1, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0.8, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_itera tion': range(115, 200, 10)	115	Accuracy: 0.594451
Create the final model with the value of the optimized parameters.			
LGBMClassifier(learning _rate=0.1, num_iteration=115, max_depth = 19, num_leaves = 20, min_data_in_leaf = 30, min_gain_to_split = 0.9, bagging_fraction =0.1, feature_fraction = 0.3, lambda_l1 = 0.9, lambda_l2 = 0.8, scale_pos_weight =1,n_jobs = -1, objective='binary')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))	CV Training Accuracy: 0.594451 Testing Accuracy: 0.593751	
The accuracy of the model with testing data was found to be: 0.593751			

Table A.10 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Dota data set

Table A.10: Hyperparameter tuning pipeline for the LGBMClassifier() with the Dota data set.

A.3.4 LightGBM with the Flavours of Physics data set

Hyperparameter tuning of LGBMClassifier() with Flavours of Physics data set				
Steps	LGBMClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1, 0.2, 0.3] 'num_itera tion': range(100, 550, 50)	0.1 400	Accuracy: 0.890017 Time: 0.05 hrs
1.1	Re-Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.01, 0.1, 0.15] 'num_itera tion': range(380, 420,5)	0.1 390	Accuracy: 0.890239 Time: 0.08 hrs
1.2	Re-Tune learning_rate and num_iteration			

Table A.11 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Flavours of Physics data set			
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.1] 'num_itera tion': range(385, 395)	0.1 392	Accuracy 0.890254 Time: 0.09 hrs
2	Tune max_depth and min_data_in_leaf		
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_depth': range(10, 30, 5), 'min_data _in_leaf': range(20, 40, 5)	15 25	Accuracy: 0.890989 Time: 0.45 hrs
2.1	Re-Tune max_depth and min_data_in_leaf		
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_depth': range(14, 17, 1), 'min_data _in_leaf': range(22, 27, 1)	15 25	Accuracy: 0.890989 Time: 0.03 hrs
3	Tune min_gain_to_split and num_leaves		

Table A.11 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Flavours of Physics data set			
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 31, min_data_in_leaf = 25, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_gain _to_split': [0, 1, 2, 3], 'num_ leaves': [15,20,25, 30]	0 20	Accuracy: 0.891038 Time: 0.02 hrs
3.1	Re-Tune min_gain_to_split and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 31, min_data_in_leaf = 25, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_gain _to_split': [0, 0.1, 0.2, 0.3], 'num_ leaves': [18,19, 20, 21, 22]	0.2 21	Accuracy: 0.891231 Time: 0.03 hrs
4	Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 21, min_data_in_leaf = 25, min_gain_to_split = 0.2, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], feature_ fraction: [0.1, 0.2, 0.3, 04, 0.5, 0.6, 0.7, 0.8, 0.9, 1]	0.1 1	Accuracy: 0.891231 Time: 0.13 hrs
4.1	Re-Tune bagging_fraction and feature_fraction		

Table A.11 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Flavours of Physics data set			
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 21, min_data_in_leaf = 25, min_gain_to_split = 0.2, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.05,0.08, 0.09, 0.10,0.12, 0.15,0.17], feature_ fraction: [0.93,0.95, 0.96,0.97, 0.98,0.99,1]	0.05 0.99	Accuracy: 0.891231 Time: 0.08 hrs
5	Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 21, min_data_in_leaf = 25, min_gain_to_split = 0.2, bagging_fraction =0.05, feature_fraction = 0.99, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0, 1, 2, 3], lambda_l2: [0, 1, 2, 3]	3 1	Accuracy: 0.892435 Time: 0.02 hrs
5.1	Re-Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 21, min_data_in_leaf = 25, min_gain_to_split = 0.2, bagging_fraction =0.05, feature_fraction = 0.99, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [2, 3, 4, 5], lambda_l2: [1.2, 1.5, 1.7,1.9]	2 1.2	Accuracy: 0.892622 Time: 0.02 hrs
5.2	Re-Tune lambda_l1 and lambda_l2		

Table A.11 continued from previous page

Hyperparameter tuning of LGBMClassifier() with Flavours of Physics data set			
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 21, min_data_in_leaf = 25, min_gain_to_split = 0.2, bagging_fraction =0.05, feature_fraction = 0.99, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [1.9, 2, 2.1, 2.3, 2.4,2.5], lambda_l2: [1.1, 1.2, 1.3, 1.4, 1.5]	2 1.2	Accuracy: 0.892622 Time: 0.05 hrs
Create the final model with the value of the optimized parameters.			
LGBMClassifier(learning _rate=0.1, num_iteration=392, max_depth = 15, num_leaves = 21, min_data_in_leaf = 25, min_gain_to_split = 0.2, bagging_fraction =0.05, feature_fraction = 0.99, lambda_l1 = 2, lambda_l2 = 1.2, scale_pos_weight =1,n_jobs = -1, objective='binary')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.892622 Testing Accuracy: 0.890747
The accuracy of the model with testing data was found to be: 0.890747			

Table A.11: Hyperparameter tuning pipeline for the LGBMClassifier() with the Flavours of Physics data set.

A.3.5 LightGBM with the Forest Cover Type data set

Hyperparameter tuning of LGBMClassifier() with the Forest Cover Type data set				
Steps	LGBMClassifier()	Parameter Range	Optimized Value	Training
1	Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iterations: range(150, 550,50) learning_ rate: [0.1, 0.2, 0.3]	450 0.1	Accuracy: 0.878218 Time: 1.3 hrs
1.1	Re-tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iterations: range(425, 480,25) learning_ rate: [0.0001, 0.001,0.01, 0.1,0.15]	450 0.1	Accuracy: 0.878218 Time: 0.37 hrs
2	Tune max_depth and min_gain_to_split			

Table A.12 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Forest Cover Type data set			
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [10, 20, 30], 'min_gain_ to_split': [0,0.5,1,1.5]	10 0.5	Accuracy: 0.89279 Time: 0.2 hrs
2.1	Re- Tune max_depth and min_gain_to_split		
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [7,8,9,10], 'min_gain_ to_split': [0.4,0.5,0.6]	8 0.5	Accuracy: 0.902745 Time: 0.36 hrs
3	Tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = 8, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0.5, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data_ in_leaf':[10, 20, 30, 40], 'num_ leaves':[10, 20,31,40]	20 40	Accuracy: 0.905323 Time: 0.3 hrs
3.1	Re-tune min_data_in_leaf and num_leaves		

Table A.12 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Forest Cover Type data set			
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = 8, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0.5, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data_ in_leaf':[16, 18, 20, 22, 24], 'num_ leaves':[36, 38,40, 42, 44]	16 44	Accuracy: 0.908607 Time: 0.69 hrs
3.2	Re-tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = 8, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0.5, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data_ in_leaf': [12,14, 15,16,17], 'num_ leaves': [43, 44, 45, 46,48]	16 44	Accuracy: 0.908607 Time: 0.55 hrs
4	Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = 8, num_leaves = 44, min_data_in_leaf = 16, min_gain_to_split = 0.5, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], feature_ fraction: [0.1, 0.2, 0.3, 04, 0.5, 0.6, 0.7, 0.8, 0.9, 1]	0.1 1	Accuracy: 0.908607 Time: 2.28 hrs

Table A.12 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Forest Cover Type data set			
5	Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = 8, num_leaves = 44, min_data_in_leaf = 16, min_gain_to_split = 0.5, bagging_fraction =0.1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0,0.8,0.9,1, 1.1,1.2], lambda_l2: [0,0.1,0.2, 0.3, 0.4]	0 0	Accuracy: 0.908607 Time: 0.28 hrs
6	Re-tune learning_rate and num_iteration		
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = 8, num_leaves = 44, min_data_in_leaf = 16, min_gain_to_split = 0.5, bagging_fraction =0.1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iterations: range(450, 1050,50) learning _rate: [0.001,0.01, 0.1]	650 0.1	Accuracy: 0.908721 Time: 1.35 hrs
6.1	Re-tune num_iteration		
LGBMClassifier(learning _rate=0.1, num_iteration=450, max_depth = 8, num_leaves = 44, min_data_in_leaf = 16, min_gain_to_split = 0.5, bagging_fraction =0.1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iterations: range(600, 700, 10)	610	Accuracy: 0.908721 Time: 0.16 hrs
Create the final model with the value of the optimized parameters.			

Table A.12 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Forest Cover Type data set		
LGBMClassifier(learning _rate=0.1, num_iteration=610, max_depth = 8, num_leaves = 44, min_data_in_leaf = 16, min_gain_to_split = 0.5, bagging_fraction =0.1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))	CV Training accuracy: 0.908721 Testing accuracy: 0.919009
The accuracy of the model with testing data was found to be: 0.919009		

Table A.12: Hyperparameter tuning pipeline for the LGBMClassifier() with the Forest Cover Type data set.

A.3.6 LightGBM with the MiniBooNE data set

Hyperparameter tuning of LGBMClassifier() with the MiniBooNE data set				
Steps	LGBMClassifier()	Parameter Range	Optimized Value	Training
1	Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [1, 2, 3] 'num_ iteration': range(100, 550, 50)	1 100	Accuracy: 0.916352 Time: 0.12 hrs
1.1	Re-Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.01, 0.1, 0.2, 0.3] 'num_itera tion': range(100, 550,50)	0.2 500	Accuracy: 0.944103 Time: 0.35 hrs
1.2	Re-Tune learning_rate and num_iteration			

Table A.13 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the MiniBooNE data set			
LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'learning _rate': [0.01, 0.1, 0.2, 0.3] num_ iteration: range(100, 850,50)	0.2 800	Accuracy 0.944851 Time: 0.9 hrs
1.3	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.2, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_ iteration': range(800, 1250,50)	1200	Accuracy: 0.945467 Time: 0.9 hrs
1.4	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.2, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_ iteration': range(1200, 1550,50)	1500	Accuracy: 0.945795 Time: 0.28 hrs
1.5	Re-Tune num_iteration		

Table A.13 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the MiniBooNE data set			
LGBMClassifier(learning _rate=0.2, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_ iteration': range(1500, 2050,50)	2000	Accuracy: 0.946071 Time: 0.6 hrs
1.6	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.2, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_ iteration': range(2000, 2550,50)	2450	Accuracy: 0.946297 Time: 0.8 hrs
1.7	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.2, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_ iteration': range(2400, 2510,10)	2430	Accuracy: 0.946343 Time: 0.8 hrs
1.8	Re-Tune num_iteration		

Table A.13 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the MiniBooNE data set			
LGBMClassifier(learning _rate=0.2, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'num_ iteration': range(2420, 2440,2)	2430	Accuracy: 0.946343 Time: 0.08 hrs
2	Tune max_depth and min_gain_to_split		
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [10, 20, 30, 40], 'num_ leaves': [10, 20, 30, 40]	30 40	Accuracy: 0.946676 Time: 1 hrs
2.1	Re-Tune max_depth and min_gain_to_split		
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': range(30, 41) 'num_ leaves': range(40, 51)	34 37	Accuracy: 0.946922 Time: 10 hrs
3	Tune min_data_in_leaf and num_leaves		

Table A.13 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the MiniBooNE data set			
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = 34, num_leaves = 37, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data _in_leaf': range(15, 40, 5) 'min_gain _to_split': [0, 1, 2, 3]	35 0	Accuracy: 0.946953 Time: 0.59 hrs
3.1	Re-Tune min_data_in_leaf and num_leaves		
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = 34, num_leaves = 37, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'min_data _in_leaf': range(32, 38, 1) 'min_gain _to_split': [0, 0.01, 0.1, 0.2]	34 0	Accuracy: 0.946974 Time: 1.5 hrs
4	Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = 34, num_leaves = 37, min_data_in_leaf = 34, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'bagging_ fraction' : [0.1, 0.3, 0.5, 0.7, 0.9, 1], 'feature_ fraction': [0.1, 0.3, 0.5, 0.7, 0.9,1]	0.1 1	Accuracy: 0.946974 Time: 1.44 hrs
4.1	Re-Tune bagging_fraction and feature_fraction		

Table A.13 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the MiniBooNE data set			
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = 34, num_leaves = 37, min_data_in_leaf = 34, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'bagging_ fraction' : [0.05, 0.1, 0.15, 0.2], 'feature_ fraction': [0.93, 0.95, 0.97,1]	0.05 1	Accuracy: 0.946974 Time: 1.4 hrs
5	Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = 34, num_leaves = 37, min_data_in_leaf = 34, min_gain_to_split = 0, bagging_fraction =0.05, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0, 1, 2, 3], lambda_l2: [0, 1, 2, 3]	0 0	Accuracy: 0.946974 Time: 0.8 hrs
5.1	Re-Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = 34, num_leaves = 37, min_data_in_leaf = 34, min_gain_to_split = 0, bagging_fraction =0.05, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_l1: [0, 0.1, 0.2, 0.3], lambda_l2: [0, 0.1, 0.2, 0.3]	0 0	Accuracy: 0.946974 Time: 0.95
Create the final model with the value of the optimized parameters.			

Table A.13 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the MiniBooNE data set		
LGBMClassifier(learning _rate=0.2, num_iteration=2430, max_depth = 34, num_leaves = 37, min_data_in_leaf = 34, min_gain_to_split = 0, bagging_fraction =0.05, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))	Training Accuracy: 0.946974 Testing Accuracy: 0.947624
The accuracy of the model with testing data was found to be: 0.947624		

Table A.13: Hyperparameter tuning pipeline for the LGBMClassifier() with the MiniBooNE data set.

A.3.7 LightGBM with the Porto Seguro data set

Hyperparameter tuning of LGBMClassifier() with the Porto Seguro data set				
Steps	LGBMClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iteration tions: range(150, 850,50) learning _rate: [0.001, 0.01,0.1]	400 0.01	Accuracy: 0.590509 Time: 2.7hrs
1.1	Re-Tune learning_rate and num_iteration			
	LGBMClassifier(learning _rate=0.1, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iteration tions: [380,400, 420] learning _rate: [0.01]	450 0.01	Accuracy: 0.590536 Time: 0.14hrs
1.2	Re-Tune num_iteration			

Table A.14 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Porto Seguro data set			
LGBMClassifier(learning _rate=0.01, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iteration tions: [370,380, 390]	370	Accuracy 0.590592 Time: 0.15 hrs
1.3	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.01, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iteration tions: [360,365, 370,375]	360	Accuracy: 0.590615 Time: 0.2 hrs
1.4	Re-Tune num_iteration		
LGBMClassifier(learning _rate=0.01, num_iteration=100, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	num_ iteration tions: [350,355, 360]	355	Accuracy: 0.590712 Time: 0.15 hrs
2	Tune max_depth, num_leaves, min_data_in_leaf, and min_gain_to_split		

Table A.14 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Porto Seguro data set			
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [10, 20, 30], 'min_ data_ in_ leaf': [20, 30, 40], 'min_ gain_ to_split': [0,0.5,1, 1.5] 'num_ leaves': [20,30,40]	20 40 0 20	Accuracy 0.592265 Time: 4.18 hrs
2.1	Re-Tune max_depth, num_leaves, min_data_in_leaf, and min_gain_to_split		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [15,20,25], 'min_ data_ in_ leaf': [35,40,45], 'min_ gain_ to_split': [0,0.1,0.2] 'num_ leaves': [15,20,25]	20 40 0 20	Accuracy: 0.592265 Time: 3.14 hrs

Table A.14 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Porto Seguro data set			
2.2	Re-Tune max_depth, num_leaves, min_data_in_leaf, and min_gain_to_split		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = -1, num_leaves = 31, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [18,20,22],	18	
	'min_ data_ in_ leaf': [38,40,42],	40	Accuracy: 0.592265
	'min_ gain_ to_split': [0,0.01]	0	Time: 3 hrs
	'num_ leaves': [18,20,22]	20	
2.3	Re-Tune min_data_in_leaf and max_depth		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = -1, num_leaves = 20, min_data_in_leaf = 20, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	'max_ depth': [16, 17, 18],	16	Accuracy: 0.592266
	'min_ data_ in_ leaf': [39, 40, 41]	40	Time: 5.8 hrs
3	Tune bagging_fraction and feature_fraction		

Table A.14 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Porto Seguro data set			
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = 16, num_leaves = 20, min_data_in_leaf = 40, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.1, 0.2, 0.3, 0.4,0.5, 0.6, 0.7, 0.8, 0.9, 1], feature_ fraction: [0.1, 0.2, 0.3, 04,0.5, 0.6, 0.7, 0.8, 0.9, 1]	0.1 0.3	Accuracy: 0.593058 Time: 5.67 hrs
3.1	Re-Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = 16, num_leaves = 20, min_data_in_leaf = 40, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.05, 0.1, 0.15] feature_ fraction: [0.25, 0.3, 0.35]	0.05 0.25	Accuracy: 0.593210 Time: 0.4 hrs
3.2	Re-Tune bagging_fraction and feature_fraction		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = 16, num_leaves = 20, min_data_in_leaf = 40, min_gain_to_split = 0, bagging_fraction =1, feature_fraction = 1, lambda_l1 = 0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	bagging_ fraction : [0.03, 0.04, 0.05, 0.06, 0.07] feature_ fraction: [0.23, 0.24, 0.25, 0.26, 0.27]	0.03 0.24	Accuracy: 0.593413 Time: 1.41 hrs

Table A.14 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Porto Seguro data set			
4	Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = 16, num_leaves = 20, min_data_in_leaf = 40, min_gain_to_split = 0, bagging_fraction =0.03, feature_fraction=0.24, lambda_l1=0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda _l1: [0,1,2,3], lambda _l2: [0,1,2,3]	0 1	Accuracy: 0.593592 Time: 0.85 hrs
5	Re-Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = 16, num_leaves = 20, min_data_in_leaf = 40, min_gain_to_split = 0, bagging_fraction =0.03, feature_fraction=0.24, lambda_l1=0, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda _l1: [0.1,0.2,0.3], lambda _l2: [0.8,0.9,1, 1.1]	0.1 0.8	Accuracy: 0.593906 Time: 1 hrs
5.1	Re-Tune lambda_l1 and lambda_l2		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = 16, num_leaves = 20, min_data_in_leaf = 40, min_gain_to_split = 0, bagging_fraction =0.03, feature_fraction=0.24, lambda_l1=0.1, lambda_l2 = 0, scale_pos_weight =1,n_jobs = -1, objective='binary')	lambda_ l2: [0.6, 0.7, 0.8]	0.8	Accuracy: 0.593906 Time: 0.13hrs

Table A.14 continued from previous page

Hyperparameter tuning of LGBMClassifier() with the Porto Seguro data set		
Create the final model with the value of the optimized parameters.		
LGBMClassifier(learning _rate=0.01, num_iteration=355, max_depth = 16, num_leaves = 20, min_data_in_leaf = 40, min_gain_to_split = 0, bagging_fraction =0.03, feature_fraction=0.24, lambda_l1=0.1, lambda_l2 = 0.8, scale_pos_weight =1,n_jobs = -1, objective='binary')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))	CV Training Accuracy: 0.593906 Testing Accuracy: 0.593455
The accuracy of the model with testing data was found to be: 0.593455		

Table A.14: Hyperparameter tuning pipeline for the LGBMClassifier() with the Porto Seguro data set.

A.4 Hyperparameter tuning pipeline for CatBoost

A.4.1 CatBoost with the Adult data set

Hyperparameter tuning of CatBoostClassifier() with the Adult data set				
Steps	CatBoostClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and iterations with other default parameters			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'learning _rate': [0.1, 0.2, 0.3] 'iterations': range(100, 550, 50)	0.1 200	Accuracy: 0.868980 Time: 3.6 hrs
1.1	Re-Tune learning_rate and iterations			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'learning _rate': [0.001, 0.01, 0.1] 'iterations': range(150, 250, 25)	0.1 200	Accuracy: 0.868980 Time: 0.69 hrs
1.2	Re-Tune iterations			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'iterations': range(180, 225, 10)	200	Accuracy: 0.868980 Time: 0.36 hrs
1.3	Re-Tune iterations			

Table A.15 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Adult data set			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': range(190, 211, 5)	200	Accuracy: 0.868980 Time: 0.36 hrs
2	Tune depth		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [6,7, 8,9,10]	10	Accuracy: 0.865288 Time: 0.08 hrs
2.1	Re-Tune depth		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [7,8, 9,10,11]	7	Accuracy: 0.869874 TIme: 1.3 hrs
2.2	Re-Tune depth		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [4,5,6,7]	5	Accuracy: 0.87055 Time: 0.2 hrs
3	Tune l2_leaf_reg		

Table A.15 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Adult data set			
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'l2_leaf _reg': [0,1,2,3, 4,5]	1	Accuracy: 0.87055 Time: 0.3 hrs
3.1	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'l2_leaf _reg': [0.8,0.9, 1,1.1,1.2]	0.8	Accuracy: 0.870707 Time: 0.27 hrs
3.2	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'l2_leaf _reg': [0.4,0.5,0.6, 0.7,0.8]	0.8	Accuracy: 0.870707 Time: 0.26 hrs
4	Tune random_strength		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0,1,2,3,4]	1	Accuracy: 0.870707 Time: 0.26 hrs
4.1	Re-Tune random_strength		

Table A.15 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Adult data set			
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0.8,0.9,1, 1.1,1.2]	1	Accuracy: 0.870707 Time: 0.27 hrs
5	Tune border_count		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	border_ count: [128,254]	254	Accuracy: 0.870707 Time: 0.02 hrs
5.1	Re-Tune border_count		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	border_ count: [250,251, 252, 253,254]	254	Accuracy: 0.870707 Time: 0.28 hrs
6	Tune bagging_temperature		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	bagging_ tempera ture: [0,1,2,3]	0	Accuracy: 0.870707 Time: 0.27 hrs
6.1	Re-Tune bagging_temperature		

Table A.15 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Adult data set			
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	bagging_ tempera ture: [0,0.5, 0.8, 0.9,1]	0	Accuracy: 0.870707 Time: 0.27 hrs
7	Re-tune learning_rate and iterations		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.1,0.2, 0.3], 'iterations': range(100, 255, 50)	0.1 250	Accuracy: 0.870816 Time: 0.4 hrs
7.1	Re-tune learning_rate and iterations		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.001, 0.01,0.1], 'iterations': range(200, 355, 25)	0.1 225	Accuracy: 0.870877 Time: 1.4 hrs
7.2	Re-tune iterations		
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	iterations: range(200, 255, 10)	240	Accuracy: 0.870973 Time: 0.35 hrs
7.3	Re-tune iterations		

Table A.15 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Adult data set			
CatBoostClassifier(iterations=200, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	iterations: range(235 ,246)	241	Accuracy: 0.871021
Create the final model with the value of the optimized parameters.			
CatBoostClassifier(iterations=241, learning_rate=0.1, depth=5, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.871021 Testing Accuracy: 0.874815
The accuracy of the model on the test data set was found to be: 0.874815			

Table A.15: Hyperparameter tuning pipeline for the CatBoostClassifier() with the Adult data set.

A.4.2 CatBoost with the Bank data set

Hyperparameter tuning of CatBoostClassifier() with the Bank data set				
Steps	CatBoostClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and iterations with other default parameters			
	CatBoostClassifier(iterations=1000, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.1, 0.2, 0.3] 'iterations': range(100, 550, 50)	0.1 100	Accuracy: 0.915829 Time: 0.9 hrs
1.1	Re-Tune learning_rate and iterations			
	CatBoostClassifier(iterations=1000, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.001, 0.01, 0.1] 'iterations': range(80, 120, 10)	0.1 80	Accuracy: 0.916314 Time: 0.47 hrs
1.2	Re-Tune iterations			
	CatBoostClassifier(iterations=1000, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': range(50, 85, 5)	70	Accuracy: 0.916746 Time: 0.2 hrs
2	Tune depth			

Table A.16 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Bank data set			
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [5,6,7,8, 9,10]	7	Accuracy: 0.916902 Time: 0.26 hrs
3	Tune l2_leaf_reg		
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [0,1,2,3, 4]	1	Accuracy: 0.916902 Time: 0.1 hrs
3.1	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [0.7,0.8, 0.9,1,1.1, 1.2]	0.8	Accuracy: 0.916988 Time: 0.12 hrs
4	Tune random_strength		
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0,1,2,3]	1	Accuracy: 0.916988 Time: 0.09 hrs
4.1	Re-Tune random_strength		

Table A.16 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Bank data set			
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0.8,0.9, 1,1.1, 1.2]	1	Accuracy: 0.916988 Time: 0.12 hrs
5	Tune border_count		
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	border_ count: [128,254]	254	Accuracy: 0.916988 Time: 0.02 hrs
5.1	Re-Tune border_count		
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	border_ count: [250,251, 252, 253,254]	254	Accuracy: 0.916988 Time: 0.12 hrs
6	Tune bagging_temperature		
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	bagging_ tempera ture: [0,1,2,3,4]	0	Accuracy: 0.916988 Time: 0.12 hrs
6.1	Re-Tune bagging_temperature		

Table A.16 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Bank data set			
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	bagging_ tempera ture: [0, 0.5,0.8, 0.9,1]	0	Accuracy: 0.916988 Time: 0.12 hrs
7	Re-Tune learning_rate and iterations		
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.1,0.2, 0.3], 'iterations': range (50,155, 20)	0.1 70	Accuracy: 0.916988 Time: 0.6 hrs
Create the final model with the value of the optimized parameters.			
CatBoostClassifier(iterations=70, learning_rate=0.1, depth=7, l2_leaf_reg=0.8,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.916988 Testing Accuracy: 0.917399
The accuracy of the model on the test data set was found to be: 0.917399			

Table A.16: Hyperparameter tuning pipeline for the CatBoostClassifier() with the Bank data set.

A.4.3 CatBoost with the Dota data set.

Hyperparameter tuning of CatBoostClassifier() with the Dota data set				
Steps	CatBoostClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and iterations with other parameters			
CatBoostClassifier(iterations=1000, learning_rate=0.03, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		iterations: range(100, 1000,100) learning_ rate: [0.03,0.01, 0.1]	900 0.03	Accuracy: 0.586077 Time: 3.5 hrs
1.1	Re-Tune iterations			
CatBoostClassifier(iterations=1000, learning_rate=0.03, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		iterations: range(900, 1500,100)	1400	Accuracy: 0.588506 Time: 5.8 hrs
1.2	Re-Tune iterations			
CatBoostClassifier(iterations=1000, learning_rate=0.03, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		iterations: range(1400, 2050,100)	1900	Accuracy: 0.589443 Time: 9.6 hrs
1.3	Re-Tune iterations			

Table A.17 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Dota data set			
CatBoostClassifier(iterations=1000, learning_rate=0.03, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	iterations: range(1850, 2050,50)	1900	Accuracy: 0.589443 Time: 8.15 hrs
2	Tune depth		
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [4,5,6]	6	Accuracy: 0.590667 Time: 4.8 hrs
2.1	Re-Tune depth		
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [6,7,8]	6	Accuracy: 0.590667 Time: 5 hrs
3	Tune l2_leaf_reg		
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=6, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [1,2,3,4]	3	Accuracy: 0.590667 Time: 7 hrs
3.1	Re-Tune l2_leaf_reg		

Table A.17 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Dota data set			
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=6, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [2.8,3, 3.2]	3	Accuracy: 0.590667 Time: 5.7 hrs
4	Tune random_strength		
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [1,2,3]	3	Accuracy: 0.590769 Time: 5.7 hrs
4.1	Re-Tune random_strength		
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [3,4,5]	3	Accuracy: 0.590769 Time: 5.9 hrs
4.2	Re-Tune random_strength		
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [2.8,3,3. 2]	3	Accuracy: 0.590769 Time: 5.9 hrs
5	Re-Tune iterations		

Table A.17 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Dota data set			
CatBoostClassifier(iterations=1900, learning_rate=0.03, depth=6, l2_leaf_reg=3,random_strength=3, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	iterations: [1900, 2500]	2500	Accuracy: 0.590784 Time: 1.8 hrs
6	Re-tune learning rate		
CatBoostClassifier(iterations=2500, learning_rate=0.03, depth=6, l2_leaf_reg=3,random_strength=3, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	learning_ rate: [0.03,0.01, 0.1]	0.03	Accuracy: 0.590784 Time: 1.77 hrs
Create the final model with the value of the optimized parameters.			
CatBoostClassifier(iterations=2500, learning_rate=0.03, depth=6, l2_leaf_reg=3,random_strength=3, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.590784 Testing Accuracy: 0.592838
The accuracy of the model on the test data set was found to be: 0.592838			

Table A.17: Hyperparameter tuning pipeline for the CatBoostClassifier() with the Dota data set.

A.4.4 CatBoost with the Flavours of Physics data set

Hyperparameter tuning of CatBoostClassifier() with the Flavours of Physics data set				
Steps	CatBoostClassifier()	Parameter Range	Optimized Value	CV Training
1	Tune learning_rate and iterations with other default parameters			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'learning _rate': [0.03,0.01, 0.1, 0.2, 0.3] 'iterations': range(100, 550, 50)	0.1 500	Accuracy: 0.890836 Time: 5.6 hrs
1.1	Re-Tune learning_rate and iterations			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'learning _rate': [0.001, 0.01, 0.1] 'iterations': range(500, 850, 50)	0.1 800	Accuracy: 0.891645 Time: 9.5 hrs
1.1	Re-Tune iterations			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'iterations': range(800, 1550, 50)	1250	Accuracy: 0.892494 Time: 10 hrs
1.2	Re-Tune iterations			

Table A.18 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Flavours of Physics data set			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': range(1200, 1300, 25)	1225	Accuracy: 0.892499 Time: 3.3 hrs
1.3	Re-Tune iterations		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': range(1215, 1240, 5)	1225	Accuracy: 0.892499 Time: 4 hrs
2	Tune depth		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [5,6,7,8,9,10]	6	Accuracy: 0.893999 Time: 4 hrs
3	Tune l2_leaf_reg		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf_ reg:[0,1,2, 3,4]	3	Accuracy: 0.894187 Time: 2.25 hrs
3.1	Re-Tune l2_leaf_reg		

Table A.18 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Flavours of Physics data set			
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf_ reg:[2.8,2.9,3, 3.1,3.2]	3	Accuracy: 0.894187 Time: 2.23
4	Tune random_strength		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0,1,2,3]	1	Accuracy: 0.894187 Time: 1.8 hrs
4.1	Re-Tune random_strength		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0.8,0.9,1, 1.1,1.2]	1	Accuracy: 0.894187 Time: 2.29 hrs
5	Tune border_count		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	border_ count: [128, 200, 254]	254	Accuracy: 0.894187 Time: 1.2 hrs
6	Tune bagging_temperature		

Table A.18 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Flavours of Physics data set			
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	bagging_ temperature: [0,1,2,3]	0	Accuracy: 0.894187 Time: 1.8 hrs
6.1	Re-Tune bagging_temperature		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	bagging_ temperature: [0,0.1,0.2,0.3]	0	Accuracy: 0.894187 Time: 1.8 hrs
7	Re-Tune learning_rate and iterations		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning_rate': [0.01,0.1,0.2] 'iterations': range(1200, 1300, 25)	0.1 1200	Accuracy: 0.894270 Time: 5.5 hrs
7.1	Re-Tune iterations		
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': range(1180, 1235, 10)	1180	Accuracy: 0.894409 Time: 2.2 hrs
7.2	Re-Tune iterations		

Table A.18 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Flavours of Physics data set			
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': range(1150, 1185, 10)	1180	Accuracy: 0.894409 Time: 1.7 hrs
7.3			
CatBoostClassifier(iterations=1225, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': [1175, 1180]	1175	Accuracy: 0.894414 Time: 0.3 hrs
Create the final model with the value of the optimized parameters.			
CatBoostClassifier(iterations=1175, learning_rate=0.1, depth=6, l2_leaf_reg=3,random_strength=1, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.894414 Testing Accuracy: 0.892790
The accuracy of the model on the test data set was found to be: 0.892790			

Table A.18: Hyperparameter tuning pipeline for the CatBoostClassifier() with the Flavours Of Physics data set.

A.4.5 CatBoost with the Forest Cover Type data set.

Hyperparameter tuning of CatBoostClassifier() with the Forest Type Cover data set				
Steps	CatBoostClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and iterations with other default parameters			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')		'learning _rate': [0.1, 0.2, 0.3] 'iterations': range(100, 550, 50)	0.3 500	Accuracy: 0.923984 Time: 1.2 hrs
1.1	Re-Tune learning_rate and iterations			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')		'learning _rate': [0.28, 0.3, 0.35] 'iterations': range(500, 1050, 50)	0.35 1000	Accuracy: 0.941419 Time: 3.6
1.2	Re-Tune learning_rate and iterations			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')		'learning _rate': [0.35, 0.38, 0.4] 'iterations': range(1000, 1550, 50)	0.4 1500	Accuracy: 0.947033 Time: 6 hrs
1.3	Re-Tune learning_rate and iterations			

Table A.19 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Forest Type Cover data set			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'learning _rate': [0.4, 0.45] 'iterations': range(1500, 2550, 100)	0.4 2500	Accuracy: 0.950685 Time: 6.6 hrs
1.4	Re-Tune learning_rate and iterations		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'learning _rate': [0.4, 0.42] 'iterations': range(2500, 3050,100)	0.4 3000	Accuracy: 0.951560 Time: 5.04 hrs
1.5	Re-Tune learning_rate and iterations		
CatBoostClassifier(iterations=100, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'iterations': range(3000, 3550,100)	3500	Accuracy: 0.952 Time: 2.7 hrs
1.6	Re-Tune iterations		
CatBoostClassifier(iterations=100, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'iterations': range(3500, 4000,100)	4000	Accuracy: 0.952420 Time: 3.2 hrs
1.7	Re-Tune iterations		

Table A.19 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Forest Type Cover data set			
CatBoostClassifier(iterations=100, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'iterations': range(4000, 5050,100)	5000	Accuracy: 0.953085 Time: 1.2 hrs
1.8	Re-Tune iterations		
CatBoostClassifier(iterations=100, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	iterations: range(5000, 5550, 100)	5000	Accuracy: 0.953085 Time: 1.8 hrs
1.9	Re-Tune iterations		
CatBoostClassifier(iterations=100, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'iterations': range(4800, 5250,50)	5150	Accuracy: 0.953105 Time: 6.6 hrs
1.10	Re-Tune iterations		
CatBoostClassifier(iterations=100, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'iterations': [5125, 5150, 5175]	5125	Accuracy: 0.953181 Time: 7 hrs
2	Tune depth		

Table A.19 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Forest Type Cover data set			
CatBoostClassifier(iterations=5125, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'depth': [4,5,6,7 ,8,9]	9	Accuracy: 0.954055 Time: 3.8 hrs
2	Re-Tune depth		
CatBoostClassifier(iterations=5125, learning_rate=0.4, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'depth':[9,10, 11]	10	Accuracy: 0.954465 Time: 3.4 hrs
3	Tune l2_leaf_reg		
CatBoostClassifier(iterations=5125, learning_rate=0.4, depth=10, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'l2_leaf _reg': [1,2,3,4]	1	Accuracy: 0.954465 Time: 4.2 hrs
3.1	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=5125, learning_rate=0.4, depth=10, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'l2_leaf _reg': [0,0.8,0.9, 1, 1.1, 1.2]	1.2	Accuracy: 0.95452 Time: 5.17 hrs
3.2	Re-Tune l2_leaf_reg		

Table A.19 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Forest Type Cover data set			
CatBoostClassifier(iterations=5125, learning_rate=0.4, depth=10, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'l2_leaf _reg': [1.2,1.3, 1.4]	1.2	Accuracy: 0.95452 Time: 3.25 hrs
4	Re-Tune learning_rate		
CatBoostClassifier(iterations=5125, learning_rate=0.4, depth=10, l2_leaf_reg=1.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'learning _rate:' [0.01,0.03, 0.1 0.4]	0.1	Accuracy: 0.955604 Time: 4.29 hrs
5	Tune random_strength		
CatBoostClassifier(iterations=5125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'random_ strength': [0.5,1,1.5,2, 3]	0.5	Accuracy: 0.955783 Time: 5 hrs
5.1	Re-Tune random_strength		
CatBoostClassifier(iterations=5125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'random_ strength': [0.4,0.5,0.6]	0.6	Accuracy: 0.955794 Time: 3.5 hrs
5.2	Re-Tune random_strength		

Table A.19 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Forest Type Cover data set			
CatBoostClassifier(iterations=5125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'random_ strength': [0.6, 0.7, 0.8]	0.8	Accuracy: 0.955949 Time: 3.5 hrs
5.3	Re-Tune random_strength		
CatBoostClassifier(iterations=5125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'random_ strength': [0.8,0.9,1]	0.8	Accuracy: 0.955949 Time: 3.5 hrs
6	Re-Tune iterations		
CatBoostClassifier(iterations=5125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=0.8, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'iterations': [5125, 5625]	5625	Accuracy: 0.956217 Time: 2.4 hrs
6.1	Re-Tune iterations		
CatBoostClassifier(iterations=5125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=0.8, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'iterations': [5625, 6125]	6125	Accuracy: 0.956348 Time: 1.61 hrs
7	Re-Tune learning_rate		

Table A.19 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Forest Type Cover data set			
CatBoostClassifier(iterations=6125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=0.8, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'learning _rate': [0.01,0.03, 0.1,0.4]	0.4	Accuracy: 0.956348 Time: 6.2 hrs
7.1	Re-Tune learning_rate		
CatBoostClassifier(iterations=6125, learning_rate=0.1, depth=10, l2_leaf_reg=1.2,random_strength=0.8, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	'learning _rate': [0.4, 0.5, 0.6]	0.4	Accuracy: 0.956348 Time: 6.2 hrs
Create the final model with the value of the optimized parameters.			
CatBoostClassifier(iterations=6125, learning_rate=0.4, depth=10, l2_leaf_reg=1.2,random_strength=0.8, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='MultiClass', eval_metric='Accuracy')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.956348 Testing Accuracy: 0.960062
The accuracy of the model on the test data set was found to be: 0.960062			

Table A.19: Hyperparameter tuning pipeline for the CatBoostClassifier() with the Forest Cover Type data set.

A.4.6 CatBoos with the MiniBooNE data set.

Hyperparameter tuning of CatBoostClassifier() with the MiniBooNE data set				
Steps	CatBoostClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and iterations with other default parameters			
	CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.1, 0.2, 0.3]	0.1	Accuracy: 0.944575
		'iterations': range(100, 550, 50)	500	Time: 1.3 hrs
1.1	Re-Tune learning_rate and iterations			
	CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.001,0.01, 0.1]	0.1	Accuracy: 0.945554
		'iterations': range(500, 1050, 50)	1000	Time: 4.7 hrs
1.2	Re-Tune iterations			
	CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	iterations: range(1000, 1550, 50)	1450	Accuracy: 0.945743
				Time: 2.25 hrs
1.3	Re-Tune iterations			

Table A.20 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the MiniBooNE data set			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	iterations: range(1410, 1495, 20)	1470	Accuracy: 0.945769 Time: 1.3 hrs
1.4	Re-Tune iterations		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'iterations': [1465,1470, 175]	1470	Accuracy: 0.945769 Time: 0.9 hrs
2	Tune depth		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [7,8,9]	7	Accuracy: 0.946287 Time: 0.93 hrs
2.1	Re-Tune depth		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	depth: [5,6,7]	7	Accuracy: 0.946287 Time: 0.48 hrs
3	Tune l2_leaf_reg		

Table A.20 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the MiniBooNE data set			
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [0,1,2,3]	3	Accuracy: 0.946333 Time: 0.7 hrs
3.1	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [3,4,5]	3	Accuracy: 0.946333 Time: 0.6 hrs
3.2	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [2.8,2.9,3, 3.1,3.2]	3.2	Accuracy: 0.946358 Time: 0.8 hrs
3.3	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	l2_leaf _reg: [3.2,3.3, 3.4,3.5]	3.2	Accuracy: 0.946358 Time: 0.6 hrs
4	Tune random_strength		

Table A.20 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the MiniBooNE data set			
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=3.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0,1,2,3]	1	Accuracy: 0.946358 Time: 0.6 hrs
4.1	Re-Tune random_strength		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=3.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0.8,0.9,1, 1.1,1.2]	0.8	Accuracy: 0.946487 Time: 0.9 hrs
4.2	Re-Tune random_strength		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=3.2,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	random_ strength: [0.7, 0.8]	0.8	Accuracy: 0.946487 Time: 0.38 hrs
5	Tune border_count		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=3.2,random_strength=0.8, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	border_count: [128,200,254]	254	Accuracy: 0.946487 Time: 0.4 hrs
6	Tune bagging_temperature		

Table A.20 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the MiniBooNE data set			
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=3.2,random_strength=0.8, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	bagging_ temperature: [0,1,2,3]	0	Accuracy: 0.946487 Time: 1 hrs
7	Re-Tune learning_rate and iterations		
CatBoostClassifier(iterations=1470, learning_rate=0.1, depth=7, l2_leaf_reg=3.2,random_strength=0.8, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'learning _rate': [0.01, 0.1, 0.2] 'iterations': range(1450, 1505, 10)	0.1 1490	Accuracy: 0.946605 Time: 2.83 hrs
Create the final model with the value of the optimized parameters.			
CatBoostClassifier(iterations=1490, learning_rate=0.1, depth=7, l2_leaf_reg=3.2,random_strength=0.8, border_count=254, bagging_temperature= 0, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.946605 Testing Accuracy: 0.947056
The accuracy of the model on the test data set was found to be: 0.947056			

Table A.20: Hyperparameter tuning pipeline for the CatBoostClassifier() with the MiniBooNE data set.

A.4.7 CatBoost with the Porto Seguro data set.

Hyperparameter tuning of CatBoostClassifier() with the Porto Seguro data set				
Steps	CatBoostClassifier()	Parameter Grid	Optimized Value	CV Training
1	Tune learning_rate and iterations with other parameters			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'learning _rate': [0.1, 0.2, 0.3] 'iterations': range(100, 550, 50)	0.1 100	Accuracy: 0.583559 Time: 3.5 hrs
2	Tune depth			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'depth': [7,8,9,10, 11]	7	Accuracy: 0.591098 Time: 0.3 hrs
2.1	Re-Tune depth			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')		'depth': [5,6,7]	5	Accuracy: 0.592368 Time: 0.06 hrs
2.2	Re-Tune depth			

Table A.21 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Porto Seguro data set			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'depth': [3,4,5]	3	Accuracy: 0.593495 Time: 0.05 hrs
2.3	Re-Tune depth		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=8, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'depth': [1,2,3]	3	Accuracy: 0.593495 Time: 0.03
3	Tune l2_leaf_reg		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=3, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'l2_leaf _reg': [1,2,3]	1	Accuracy: 0.593495 Time: 0.1 hrs
3.1	Re-Tune l2_leaf_reg		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=3, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'l2_leaf _reg': [0,1]	1	Accuracy: 0.593495 Time: 0.01 hrs
4	Tune random_strength		

Table A.21 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Porto Seguro data set			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=3, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'random_ strength': [0,1,2]	1	Accuracy: 0.593495 Time: 0.06 hrs
5	Tune border_count		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=3, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'border_ count': [128, 200, 254]	254	Accuracy: 0.593495 Time: 0.06 hrs
6	Tune bagging_temperature		
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=3, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	'bagging_ temperature': [0,1,2]	0	Accuracy: 0.593495 Time: 0.06 hrs
Create the final model with the value of the optimized parameters.			
CatBoostClassifier(iterations=100, learning_rate=0.1, depth=3, l2_leaf_reg=1,random_strength=1, border_count=254, bagging_temperature= 1, cat_features = nominal, loss_function='Logloss', eval_metric='Accuracy')	model.fit(X_train, y_train) y_predict = model. predict (X_test) print('Testing accuracy score:', accuracy_score (y_test, y_predict))		CV Training Accuracy: 0.593495 Testing Accuracy: 0.592164

Table A.21 continued from previous page

Hyperparameter tuning of CatBoostClassifier() with the Porto Seguro data set
The accuracy of the model on the test data set was found to be: 0.592164

Table A.21: Hyperparameter tuning pipeline for the CatBoostClassifier() with the Porto Seguro data set.

Appendix B

Statistical significance test.

For proving statistical significance of the results obtained from algorithms, we will follow a method discussed by Tom M. Mitchel in his book 'Machine Learning' at chapter 5, section 5.5.

Here, we will compare the results of Classifium from all 7 different data sets, against the results from all other algorithms with the same 7 different data sets.

For proving different accuracy obtained from two different algorithm with same data sets are not just the results by chance, but that they are statistically significant, we will conduct a statistical significance test as given below.

Let us assume that our alternate hypothesis (H1) states that, based on the different results obtained from the two different algorithms tested with the same sample data set, algorithms' classification accuracy on the whole data set are different and algorithms are comparable. And let our null hypothesis (H0) states that, based on the different results obtained from the two different algorithms tested with same sample data set, algorithms' classification accuracy on whole data set are not different, so, algorithms are not comparable. In order to find the correct meaning from the results, we can perform the hypothesis testing.

Let us suppose two different accuracies from two algorithms be acc(h1) and acc(h2).

Let d represents the difference between the accuracy obtained from two hypothesis on same data set D. i.e, the difference between $acc_D(h1)$ and $acc_D(h2)$ can be represented as:

$$\mathrm{d} = acc_D(h1) - acc_D(h2)$$

But during our experiment we took only the part of our data for training and testing. So, the difference in accuracy obtained from sample data set can be represented by:

$\hat{d} = acc(h1) - acc(h2)$

For a random variable $\hat{d}$ obeying a Normal distribution with mean d and variance σ^2, the N% confidence interval estimate for d is $\hat{d} \pm Z_N \sigma$

Concluding that if d is > 0, and $\hat{d}$ falls between the interval of d, then we can say that the difference in accuracy is true or significant for whole data set by at least 95% of probability.

Since we will be testing for the statistical significance between the accuracy obtained from the two different algorithms trained and tested with the same data set, n will be the same for both distribution. So, standard deviation of accuracy will be

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}$

$\sigma = \sqrt{\frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}}$

Now the confidence interval will be defined as,

$\mathrm{d} = \hat{d} \pm (Z_N) * \sigma$

Since our sample are larger in number, so, from the table given by Tom M. Mitchel, we will use 1.96 as value for Z_N Now, if our $\hat{d}$ lies between this interval, and d>0, then we can say that our observations are statistically significant by 95% of confidence.

B.1 Statistical significance test between the classification accuracy obtained from the XGBoost and the Classifium

In this section we will calculate the statistical significance between the results obtained from the two different algorithms, XGBoost and Classifium, with the same data sets. The calculation for each case is demonstrated in the sub-sections below:

B.1.1 Porto Seguro

Let the classification accuracy obtained from XGBoost and Classifium with the Porto Seguro data set be represented by acc(h1) and acc(h2) respectively.

From our experimentation result, we found that,

acc(h1) = 0.595298 and acc(h2) = 0.598544

Let their difference be represented by $\hat{d}$

$\hat{d} = acc(h1) - acc(h2)$

$\hat{d} = 0.595298 - 0.598544$

$\hat{d} = -0.003246$

Now, lets find the approximate variance of two observation over their distribution:

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}$

Since the size of sample in which both algorithms were tested was 21695. So we will replace 'n' in the given equation with the value of sample size, that is 21695.

$\sigma^2 = \frac{0.595298(1-0.595298)}{21695} + \frac{0.598544(1-0.598544)}{21695}$

$\sigma = \sqrt{0.000022181}$

$\sigma = 0.004709671$

Now, lets find the confidence interval.

$d = \hat{d} \pm (Z_N) * \sigma$

$d = -0.003246 \pm (1.96) * 0.004709671$

$d = -0.003246 \pm 0.009230955$

d = -0.012476955

$\leq \hat{d} \leq (0.005984955)$

The difference $\hat{d}$, lies between the confidence interval but d < 0, so we can not say that our result is 95% statistically significant. And, we have to accept the null hypothesis (H0) and reject the alternative hypothesis (H1).

B.1.2 Flavours of Physics

Let the classification accuracy obtained from XGBoost and Classifium with the Flavours of Physics data set be represented by acc(h1) and acc(h2) respectively.

From our experimentation result, we found that,

acc(h1) = 0.892701 and acc(h2) = 0.89158

Let their difference be given by $\hat{d}, suchthat$,

$\hat{d} = acc(h1) - acc(h2)$

$\hat{d} = 0.892701 - 0.89158$

$\hat{d} = 0.001121$

Now, lets find the approximate variance of the two observations over their distribution:

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}$

Since the size of sample in which both algorithms were tested was 33775. So we will replace 'n' in the given equation with the value of sample size, that is 33775.

$\sigma^2 = \frac{0.892701(1-0.892701)}{33775} + \frac{0.89158(1-0.89158)}{33775}$

$\sigma = \sqrt{0.000005698}$

$\sigma = 0.002387048$

Now, lets find the confidence interval.

$d = \hat{d} \pm (Z_N) * \sigma$

$d = 0.001121 \pm (1.96) * 0.002387048$

$d = 0.001121 \pm 0.004678615$

$d =$ (-0.003557615) $\leq \hat{d} \leq$ (0.005799615)

From the above confidence interval we can say that, since our d is not absolutely > 0, we reject alternative hypothesis (H1) and accept null hypothesis (H0). And conclude that there is no any statistical significance between the obtained results and both algorithms on the same whole data set is similar.

B.1.3 MiniBooNE

Let the classification accuracy obtained from XGBoost and Classifium with the MiniBooNE data set be denoted by acc(h1) and and acc(h2).

N = 65031

From our experimentation result, we found that,

acc(h1) = 0.946856 and acc(h2) = 0.947057

So, their difference will be

$\hat{d} = acc(h1) - acc(h2)$

$\hat{d} = 0.946856 - 0.947057$

$\hat{d} =$ (-0.000201)

Now, lets find the approximate variance of two observation over their distribution:

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n} + \frac{acc(h2)(1-acc(h2))}{n}$

Since the size of the sample in which both algorithms were tested was 65031. So we will replace 'n' in the given equation with the value of sample size, that is 65031.

$\sigma^2 = \frac{0.946856(1-0.946856)}{65031} + \frac{0.947057(1-0.947057)}{65031}$

$\sigma = \sqrt{0.000001545}$

$\sigma = 0.00124298$

Now, lets find the confidence interval.

$d = \hat{d} \pm (Z_N) * \sigma$

$d = (-0.000201) \pm (1.96) * 0.00124298$

$d = (-0.000201) \pm 0.002436241$

$d =$ (-0.002637241) $\leq \hat{d} \leq$ (0.002235241)

From the above confidence interval we can say that, since our d is not absolutely > 0, we reject alternative hypothesis (H1) and accept null hypothesis (H0). And conclude that there is no any statistical significance between the obtained results.

B.1.4 Adult

Let the classification accuracy obtained from XGBoost and Classifium with the Adult data set be represented by acc(h1) and acc(h2) respectively.
From our experimentation result, we found that,

acc(h1) = 0.875225 and acc(h2) = 0.87523

Let their difference be represented by $\hat{d}$.

So,

$\hat{d} = acc(h1) - acc(h2)$

$\hat{d} = 0.875225 - 0.87523$

$\hat{d} =$ (-0.000005)

Now, lets find the approximate variance of two observation over their distribution:

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}$

Here the sample size is 24420, so n = 24420.

$\sigma^2 = \frac{0.875225(1-0.875225)}{24420} + \frac{0.87523(1-0.87523)}{24420}$

$\sigma = \sqrt{0.000008944}$

$\sigma = 0.002990652$

Now, lets find the confidence interval.

$d = \hat{d} \pm (Z_N) * \sigma$

$d = (-0.000005) \pm (1.96) * 0.002990652$

$d = (-0.000005) \pm 0.005861678$

$d = (\text{-}0.005866678) \leq \hat{d} \leq (0.005856678)$

From the above confidence interval we can say that, since our d is not absolutely > 0, we reject the alternative hypothesis (H1) and accept the null hypothesis (H0). And conclude that there is no any statistical significance between the obtained results.

B.1.5 Dota

Let the accuracy obtained from XGBoost and Classifium with the Dota data set be represented by acc(h1) and acc(h2) respectively.
From our experimentation result, we found that,

acc(h1) = 0.594859 and acc(h2) = 0.597529

Let the difference between acc(h1) and acc(h2) be reppresented by $\hat{d}$.
So,

$\hat{d} = acc(h1) - acc(h2)$

$\hat{d} = 0.594859 - 0.597529$

$\hat{d} = (\text{-}0.00267)$

Now, lets find the approximate variance of two observation over their distribution:

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}$

Here sample size is 51471, so, n = 51471

$\sigma^2 = \frac{0.594859(1-0.594859)}{51471} + \frac{0.597529(1-0.597529)}{51471}$

$\sigma = \sqrt{0.000009355}$

$\sigma = 0.003058594$

Now, lets find the confidence interval.

$d = \hat{d} \pm (Z_N) * \sigma$

$d = (-0.00267) \pm (1.96) * 0.003058594$

$d = (-0.00267) \pm 0.005994845$

$d =$ (-0.008664845) $\leq \hat{d} \leq$ (0.003324845)

From the above confidence interval we can say that, since our d is not absolutely > 0, we reject alternative hypothesis (H1) and accept null hypothesis (H0). And conclude that there is no any statistical significance between the obtained results.

B.1.6 Bank

Let the classification accuracy obtained from XGBoost and Classifium with the Bank data set be represented by acc(h1) and acc(h2) respectively.

From our experimentation result, we found that,
acc(h1) = 0.916525 and acc(h2) = 0.915995
So, their difference will be

$\hat{d} = acc(h1) - acc(h2)$

$\hat{d} = 0.916525 - 0.915995$

$\hat{d} = (0.00053)$

Now, lets find the approximate variance of two observation over their distribution:

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}$

Here the sample size is 51471, so n = 51471

$\sigma^2 = \frac{0.916525(1-0.916525)}{20593} + \frac{0.915995(1-0.915995)}{20593}$

$\sigma = \sqrt{0.000007452}$

$\sigma = 0.002729835$

Now, lets find the confidence interval.

$d = \hat{d} \pm (Z_N) * \sigma$

$d = (0.00053) \pm (1.96) * 0.002729835$

$d = (0.00053) \pm 0.005350477$

$d = (-\ 0.004820477) \leq \hat{d} \leq (0.005880477)$

From the above confidence interval we can say that, since our d is not absolutely > 0, we reject alternative hypothesis (H1) and accept null hypothesis (H0). And conclude that there is no any statistical significance between the obtained results. So, the both algorithm performed same.

B.1.7 Forest Cover Type

Let the the classification accuracy obtained from XGBoost and Classifium with the Forest Cover Type data set be denoted as acc(h1) and acc(h2) respectively.

N = 2090505

From our experimentation result, we found that,
acc(h1) = 0.968155 and acc(h2) = 0.971873

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So,

$\hat{d} = acc(h1) - acc(h2)$

$\hat{d} = 0.968155 - 0.971873$

$\hat{d} =$ (- 0.003718)

Now, lets find the approximate variance of two observation over their distribution:

$\sigma^2 = \frac{acc(h1)(1-acc(h1))}{n1} + \frac{acc(h2)(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.968155(1-0.968155)}{2090505} + \frac{0.971873(1-0.971873)}{2090505}$

$\sigma = \sqrt{0.000000028}$

$\sigma = 0.000167332$

Now, lets find the confidence interval.

$d = \hat{d} \pm (Z_N) * \sigma$

$d = (-0.003718) \pm (1.96) * 0.000167332$

$d = (-0.003718) \pm 0.000167332$

$d =$ (- 0.003885332) $\leq \hat{d} \leq (-0.003550668)$

From the above confidence interval we can say that, since our d does not include 0, and includes $\hat{d}$, so, we accept the alternative hypothesis (H1) and reject the null hypothesis (H0). And conclude that the difference between the classification accuracy from two algorithms are statistically significant with 95% of confidence.

B.2 Statistical significance test between the classification accuracy obtained from the LightGBM and the Classifium

On this section we will calculate for the statistical significance between the results obtained from LightGBM and Classifium with 7 different data sets.

B.2.1 Porto Seguro

Let the accuracy obtained from LightGBM be denoted by acc(h1) and the accuracy obtained from Classifium be denoted by acc(h2).

From our experiment, we found that,
acc(h1) = 0.593455 and acc(h2) = 0.598544
Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So, $\hat{d}$ = acc(h1)-acc(h2)
$\hat{d}$ = 0.593455 - 0.598544
$\hat{d}$ = - 0.005089

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 21695, so, n = 21695

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.593455*(1-0.593455)}{21695} + \frac{0.598544*(1-0.598544)}{21695}$

$\sigma^2 = \sqrt{0.000022197}$

$\sigma = 0.004711369$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval.
So,

d = $-0.005089 \pm 1.96 * 0.004711369$

d= $-0.009800369 \leq \hat{d} \leq -0.000377631$

From the above confidence interval we can say that, since our d does not include 0, and includes $\hat{d}$, so, we accept the alternative hypothesis (H1) and reject the null hypothesis (H0). And conclude that the difference between the classification accuracy from two algorithms are statistically significant with 95% of confidence.

B.2.2 Flavours of Physics

let the classification accuracy obtained by LightGBM and Classifium with the Flavours of Physics data set be denoted by acc(h1) and acc(h2).

From our experiment, we found that,
acc(h1) = 0.890747 and acc(h2) = 0.89158
Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So, $\hat{d}$ = acc(h1)-acc(h2)
$\hat{d}$ = 0.890747 - 0.89158
$\hat{d}$ = - 0.000833

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 33775, so, n = 33775

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.890747*(1-0.890747)}{33775} + \frac{0.89158*(1-0.89158)}{33775}$

$\sigma^2 = \sqrt{0.000005743}$

$\sigma = 0.002396456$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval.
So,

d = $-0.000833 \pm 1.96 * 0.002396456$

d= $-0.005530054 \leq \hat{d} \leq 0.003864054$

Though the given interval, d, contains difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithm performed equally on this data set.

B.2.3 MiniBooNE

let the classification accuracy obtained by LightGBM and Classifium with the MiniBooNE data set be denoted by acc(h1) and acc(h2).

From our experiment, we found that,
acc(h1) = 0.947624 and acc(h2) = 0.947057
Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So, $\hat{d}$ = acc(h1)-acc(h2)
$\hat{d}$ = 0.947624 - 0.947057
$\hat{d}$ = - 0.000833

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 65031, so, n = 65031

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.947624*(1-0.947624)}{65031} + \frac{0.947057*(1-0.947057)}{65031}$

$\sigma^2 = \sqrt{0.000001534}$

$\sigma = 0.001238548$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval.
So,

d = $0.001238548 \pm 1.96 * 0.001238548$

d= $-0.24151686 \leq \hat{d} \leq 0.243993956$

Though the given interval, d, contains difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithm performed equally on this data set.

B.2.4 Adult

let the classification accuracy obtained by LightGBM and Classifium with the Adult data set be denoted by acc(h1) and acc(h2).

From our experiment, we found that,
acc(h1) = 0.875389 and acc(h2) = 0.87523
Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So, $\hat{d}$ = acc(h1)-acc(h2)
$\hat{d}$ = 0.875389 - 0.87523
$\hat{d}$ = 0.000159

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 24420, so, n = 24420

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.875389*(1-0.875389)}{24420} + \frac{0.87523*(1-0.87523)}{24420}$

$\sigma^2 = \sqrt{0.000008939}$

$\sigma = 0.002989816$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval.
So,

d = $0.000159 \pm 1.96 * 0.002989816$

d= $-0.585844936 \leq \hat{d} \leq 0.586162936$

Though the given interval, d, contains difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithm performed equally on this data set.

B.2.5 Dota

let the classification accuracy obtained by LightGBM and Classifium with the Dota data set be denoted by acc(h1) and acc(h2).

From our experiment, we found that,
acc(h1) = 0.593751 and acc(h2) = 0.597529
Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So, $\hat{d}$ = acc(h1)-acc(h2)
$\hat{d}$ = 0.593751 - 0.597529
$\hat{d}$ = - 0.003778

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 51471, so, n = 51471

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.593751*(1-0.593751)}{51471} + \frac{0.597529*(1-0.597529)}{51471}$

$\sigma^2 = \sqrt{0.000009359}$

$\sigma = 0.003059248$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval.
So,

d = $-0.003778 \pm 1.96 * 0.003059248$

d = $-0.003778 \pm 0.599612608$

d= $-0.603390608 \leq \hat{d} \leq 0.595834608$

Though the given interval, d, contains difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithm performed equally on this data set.

B.2.6 Bank

let the classification accuracy obtained by LightGBM and Classifium with the Bank data set be denoted by acc(h1) and acc(h2).

From our experiment, we found that,
acc(h1) = 0.915893 and acc(h2) = 0.915995
Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So, $\hat{d}$ = acc(h1)-acc(h2)
$\hat{d}$ = 0.915893 - 0.915995
$\hat{d}$ = - 0.000102

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 20593, so, n = 20593

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.915893*(1-0.915893)}{20593} + \frac{0.915995*(1-0.915995)}{20593}$

$\sigma^2 = \sqrt{0.000007477}$

$\sigma = 0.00273441$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval.
So,

d $= -0.000102 \pm 1.96 * 0.003059248$

d $= -0.000102 \pm 0.53594436$

d= $-0.53604636 \leq \hat{d} \leq 0.53584236$

Though the given interval, d, contains difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithm performed equally on this data set.

B.2.7 Forest Cover Type

let the classification accuracy obtained from the LightGBM and Classifium with the same data set, Forest Cover Type, be denoted as acc(h1) and acc(h2) respectively. From our experiment, we found that,

acc(h1) = 0.919009 and acc(h2) = 0.971873

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.919009 - 0.971873

$\hat{d}$ = -0.052864

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 290505, so, n = 290505

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.919009*(1-0.919009)}{290505} + \frac{0.971873*(1-0.971873)}{290505}$

$\sigma^2 = \sqrt{0.00000035}$

$\sigma = 0.000591608$

Now, lets find the confidence interval for d

d $= \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

$$\text{d} = -0.052864 \pm 1.96 * 0.000591608$$

$$\text{d} = -0.052864 \pm 0.001159552$$

$$\text{d}= -0.054023552 \leq \hat{d} \leq -0.051704448$$

From the above confidence interval we can say that, since our d does not include 0, and includes $\hat{d}$, so, we accept the alternative hypothesis (H1) and reject the null hypothesis (H0). And conclude that the difference between the classification accuracy from two algorithms are statistically significant with 95% of confidence.

B.3 Statistical significance test between the classification accuracy obtained from the CatBoost and the Classifium

In this section we will perform the statistical significance test between the results obtained by training and testing CatBoost and Classifium with the same data sets. And analyze, whether the result obtained from the sample of data is statistically significant to whole data set or not.

Both the algorithms were trained and tested with the 7 different data sets. In below sub-sections, we will analyse for statistical significance between the classification results obtained from CatBoost and Classifium with the similar data set.

B.3.1 Porto Seguro

let the classification accuracy obtained from CatBoost and Classifium with the same Porto Seguro data set, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.592164 and acc(h2) = 0.598544

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.592164 - 0.598544

$\hat{d}$ = - 0.00638

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 21695, so, n = 21695

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.592164*(1-0.592164)}{21695} + \frac{0.598544*(1-0.598544)}{21695}$

$\sigma^2 = \sqrt{0.000022208}$

σ = 0.004712536

Now, lets find the confidence interval for d.

$\mathrm{d} = \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

$\mathrm{d} = -0.00638 \pm 1.96 * 0.004712536$

$\mathrm{d} = -0.00638 \pm 0.009236571$

$\mathrm{d} = -0.015616571 \leq \hat{d} \leq 0.002856571$

Though the given interval, d, contains difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.3.2 Flavours of Physics

let the classification accuracy obtained from CatBoost and Classifium with the same data set, Flavours of Physics, be denoted as acc(h1) and acc(h2).

From our experiment, we found that,

acc(h1) = 0.892790 and acc(h2) = 0.89158
Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.
So, $\hat{d}$ = acc(h1)-acc(h2)
$\hat{d}$ = 0.892790 - 0.89158
$\hat{d}$ = 0.00121

Now, lets find the approximate variance of two observation over their distribution:

Here the sample size is 33775, so, n = 33775

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.892790*(1-0.892790)}{33775} + \frac{0.89158*(1-0.89158)}{33775}$

$\sigma^2 = \sqrt{0.000005696}$

$\sigma = 0.002386629$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval.

So,

d = $0.00121 \pm 1.96 * 0.002386629$

d = 0.00121 ± 0.004677793

d= $-0.003467793 \leq \hat{d} \leq 0.005887793$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.3.3 MiniBooNE

let the classification accuracy obtained from CatBoost and Classifium with the same data set, MiniBooNE, be denoted as acc(h1) and acc(h2) respectively.

From our experiment, we found that,

acc(h1) = 0.947056 and acc(h2) = 0.947057

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$.

So, $\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.947056 - 0.947057

$\hat{d}$ = -0.000001

Now, lets find the approximate variance of the two observations over their distribution:

Here the sample size is 65031, so, n = 65031

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.947056*(1-0.947056)}{65031} + \frac{0.947057*(1-0.947057)}{65031}$

$\sigma^2 = \sqrt{0.000001542}$

$\sigma = 0.001241773$

Now, lets find the confidence interval

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.000001 \pm 1.96 * 0.001241773$

d = $-0.000001 \pm 0.002433875$

d= $-0.002434875 \leq \hat{d} \leq 0.002432875$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.3.4 Adult

let the classification accuracy obtained from CatBoost and Classifium with the same data set, Adult, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.874815 and acc(h2) = 0.87523

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.874815 - 0.87523

$\hat{d}$ = -0.000415

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 24420, so, n = 24420

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.874815*(1-0.874815)}{24420} + \frac{0.87523*(1-0.87523)}{24420}$

$\sigma^2 = \sqrt{0.000008956}$

$\sigma = 0.002992658$

Now, lets find the confidence interval for d

d $= \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d $= -0.000415 \pm 1.96 * 0.002992658$

d $= -0.000415 \pm 0.00586560968$

d= $-0.00628061 \leq \hat{d} \leq 0.00545061$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.3.5 Dota

let the classification accuracy obtained from CatBoost and Classifium with the same data set, Dota, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.592838 and acc(h2) = 0.597529

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.592838 - 0.597529

$\hat{d}$ = -0.004691

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 51471, so, n = 51471

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.592838*(1-0.592838)}{51471} + \frac{0.597529*(1-0.597529)}{51471}$

$\sigma^2 = \sqrt{0.000009362}$

$\sigma = 0.003059739$

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.004691 \pm 1.96 * 0.003059739$

d = $-0.004691 \pm 0.005997088$

d= $-0.010688088 \leq \hat{d} \leq 0.001306088$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.3.6 Bank

let the classification accuracy obtained from CatBoost and Classifium with the same data set, Bank, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.917399 and acc(h2) = 0.915995

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.917399 - 0.915995

$\hat{d}$ = 0.001404

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 20593, so, n = 20593

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.917399*(1-0.917399)}{20593} + \frac{0.915995*(1-0.915995)}{20593}$

$\sigma^2 = \sqrt{0.000007416}$

σ = 0.002723233

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $0.001404 \pm 1.96 * 0.002723233$

d = 0.001404 ± 0.005337537

d= $-0.003933537 \leq \hat{d} \leq 0.006741537$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.3.7 Forest Cover Type

let the classification accuracy obtained from CatBoost and Classifium with the same data set, Forest Cover Type, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.960062 and acc(h2) = 0.971873

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.960062 - 0.971873

$\hat{d}$ = -0.011811

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 290505, so, n = 290505

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.960062*(1-0.960062)}{290505} + \frac{0.971873*(1-0.971873)}{290505}$

$\sigma^2 = \sqrt{0.000000226}$

σ = 0.000475395

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.011811 \pm 1.96 * 0.000475395$

d = $-0.011811 \pm 0.000931774$

d= $-0.012742774 \leq \hat{d} \leq -0.010879226$

Here, the given interval, d, contains the difference value $\hat{d}$, and also it does not include 0 inside the interval. So, we reject the null hypothesis (H0) and accept the alternative hypothesis (H1). And conclude that Classifium performed better than CatBoost in terms of classification accuracy in this data set.

B.4 Statistical significance test between the classification accuracy obtained from the H2O and the Classifium

In this section we will perform the statistical significance test between the results obtained by training and testing H2O and Classifium with the same datasets. And analyze, whether the result obtained from the same sample of data is statistically significant to whole data set or not.

Both the algorithms were trained and tested with the 7 different data sets. In the below sub-sections, we will analyse for statistical significance between the classification results obtained from H2O and Classifium with the similar data set. Here H2O is trained and tested with ensemble of all possible base learners it supports, Gradient Boosting Machine, Distributed Random Forest, Deep Learning, XGBoost, Navie Bias Classifier, and so on. The classification accuracy from the final stacked ensemble model of H2O, is used here against the accuracy results obtained from Classifium with the same data sets to find the statistical significance between their different results.

B.4.1 Porto Seguro

let the classification accuracy obtained from ensemble H2O and Classifium with the same data set, Porto Seguro, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.596958 and acc(h2) = 0.598544

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.596958 - 0.598544

$\hat{d}$ = -0.001586

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 21695, so, n = 21695

$$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$$

$$\sigma^2 = \frac{0.596958*(1-0.596958)}{21695} + \frac{0.598544*(1-0.598544)}{21695}$$

$$\sigma^2 = \sqrt{0.000022166}$$

$\sigma = 0.004708078$

Now, lets find the confidence interval for d

$\mathrm{d} = \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

$\mathrm{d} = -0.001586 \pm 1.96 * 0.004708078$

$\mathrm{d} = -0.001586 \pm 0.009227833$

$\mathrm{d} = -0.010813833 \leq \hat{d} \leq 0.007641833$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.4.2 Flavours of Physics

let the classification accuracy obtained from ensemble H2O and Classifium with the same data set, Flavours of Physics, be denoted as acc(h1) and acc(h2) respectively. From our experiment, we found that,

acc(h1) = 0.889297 and acc(h2) = 0.89158

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.889297 - 0.89158

$\hat{d}$ = -0.002283

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 33775, so, n = 33775

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.889297*(1-0.889297)}{33775} + \frac{0.89158*(1-0.89158)}{33775}$

$\sigma^2 = \sqrt{0.000005777}$

$\sigma = 0.002403539$

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.002283 \pm 1.96 * 0.002403539$

d = $-0.002283 \pm 0.004710936$

d= $-0.006993936 \leq \hat{d} \leq 0.002427936$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.4.3 MiniBooNE

let the classification accuracy obtained from ensemble H2O and Classifium with the same data set, MiniBooNE, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.945564 and acc(h2) = 0.947057

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.945564 - 0.947057

$\hat{d}$ = -0.001493

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 65031, so, n = 65031

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.945564*(1-0.945564)}{65031} + \frac{0.947057*(1-0.947057)}{65031}$

$\sigma^2 = \sqrt{0.000001563}$

$\sigma = 0.0012502$

Now, lets find the confidence interval for d

$\mathrm{d} = \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

$\mathrm{d} = -0.001493 \pm 1.96 * 0.0012502$

$\mathrm{d} = -0.001493 \pm 0.002450392$

d= $-0.003943392 \leq \hat{d} \leq 0.000957392$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.4.4 Adult

let the classification accuracy obtained from ensemble H2O and Classifium with the same data set, Adult, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.876577 and acc(h2) = 0.87523

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.876577 - 0.87523

$\hat{d} = 0.001347$

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 24420, so, n = 24420

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.876577*(1-0.876577)}{24420} + \frac{0.87523*(1-0.87523)}{24420}$

$\sigma^2 = \sqrt{0.000008902}$

$\sigma = 0.002983622$

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $0.001347 \pm 1.96 * 0.002983622$

d = 0.001347 ± 0.005847899

d= $-0.004500899 \leq \hat{d} \leq 0.007194899$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.4.5 Dota

let the classification accuracy obtained from ensemble H2O and Classifium with the same data set, Dota, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.597268 and acc(h2) = 0.597529

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.597268 - 0.597529

$\hat{d}$ = -0.000261

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 51471, so, n = 51471

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.597268*(1-0.597268)}{51471} + \frac{0.597529*(1-0.597529)}{51471}$

$\sigma^2 = \sqrt{0.000009346}$

$\sigma = 0.003057123$

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.000261 \pm 1.96 * 0.003057123$

d = $-0.000261 \pm 0.005991961$

d= $-0.006252961 \leq \hat{d} \leq 0.005730961$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.4.6 Bank

let the classification accuracy obtained from ensemble H2O and Classifium with the same data set, Dota, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.915894 and acc(h2) = 0.915995

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.915894 - 0.915995

$\hat{d}$ = -0.000101

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 24420, so, n = 24420

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.915894*(1-0.915894)}{24420} + \frac{0.915995*(1-0.915995)}{24420}$

$\sigma^2 = \sqrt{0.000006306}$

σ = 0.002511175

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.000101 \pm 1.96 * 0.002511175$

d = $-0.000101 \pm 0.004921903$

d= $-0.005022903 \leq \hat{d} \leq 0.004820903$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.4.7 Forest Cover Type

Let the classification accuracy obtained from the H2O with the stacked ensemble model and Classifium, with the same data set, Forest Cover Type, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.954307 and acc(h2) = 0.971873

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.954307 - 0.971873

$\hat{d}$ = -0.017566

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 290505, so, n = 290505

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.954307*(1-0.954307)}{290505} + \frac{0.971873*(1-0.971873)}{290505}$

$\sigma^2 = \sqrt{0.000000244}$

σ = 0.000493964

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.017566 \pm 1.96 * 0.000493964$

d = $-0.017566 \pm 0.000968169$

d= $-0.018534169 \leq \hat{d} \leq -0.016597831$

From the above confidence interval we can say that, since our d does not include 0, and includes $\hat{d}$, so, we accept the alternative hypothesis (H1) and reject the null hypothesis (H0). And conclude that the difference between the classification accuracy from two algorithms are statistically significant with 95% of confidence.

B.5 Statistical significance test between the classification accuracy obtained from the H2O with XGBoost as base learner and the Classifium

In this section we will perform the statistical significance test between the results obtained by training and testing H2O with XGBoost only as its base learner, and the Classifium with the same data sets. And analyze, whether the result obtained from the same sample of data is statistically significant to whole data set or not.

B.5.1 Porto Seguro

let the classification accuracy obtained from XGBoost base H2O and Classifium with the same data set, Porto Seguro, be denoted as acc(h1) and acc(h2) respectively. From our experiment, we found that,

acc(h1) = 0.595391 and acc(h2) = 0.598544

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.595391 - 0.598544

$\hat{d}$ = -0.003153

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 21695, so, n = 21695

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.595391*(1-0.595391)}{21695} + \frac{0.598544*(1-0.598544)}{21695}$

$\sigma^2 = \sqrt{0.00002218}$

σ = 0.004709565

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence

interval. So,

$$\text{d} = -0.003153 \pm 1.96 * 0.004709565$$

$$\text{d} = -0.003153 \pm 0.009230747$$

$$\text{d= } -0.012383747 \leq \hat{d} \leq 0.006077747$$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.5.2 Flavours of Physics

let the classification accuracy obtained from the XGBoost based H2O and Classifium with the same data set, Porto Seguro, be denoted as acc(h1) and acc(h2) respectively. From our experiment, we found that,

acc(h1) = 0.887224 and acc(h2) = 0.89158

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.887224 - 0.89158

$\hat{d}$ = -0.004356

Now, lets find the approximate variance of the two observations over their distribution. Here the sample size is 33775, so, n = 33775

$$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$$

$$\sigma^2 = \frac{0.887224*(1-0.887224)}{33775} + \frac{0.89158*(1-0.89158)}{33775}$$

$$\sigma^2 = \sqrt{0.000005825}$$

$$\sigma = 0.002413504$$

Now, lets find the confidence interval for d

$$\text{d} = \hat{d} \pm (Z_N) * \sigma$$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.004356 \pm 1.96 * 0.002413504$

d = $-0.004356 \pm 0.004730468$

d= $-0.009086468 \leq \hat{d} \leq 0.000374468$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.5.3 MiniBooNE

let the classification accuracy obtained from the XGBoost based H2O and Classifium with the same data set, MiniBooNE, be denoted as acc(h1) and acc(h2) respectively. From our experiment, we found that,

acc(h1) = 0.944242 and acc(h2) = 0.947057

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.944242 - 0.947057

$\hat{d}$ = -0.002815

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 65031, so, n = 65031

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.944242*(1-0.944242)}{65031} + \frac{0.947057*(1-0.947057)}{65031}$

$\sigma^2 = \sqrt{0.000001581}$

σ = 0.001257378

Now, lets find the confidence interval for d

$d = \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

$d = -0.002815 \pm 1.96 * 0.001257378$

$d = -0.002815 \pm 0.002464461$

$d = -0.005279461 \leq \hat{d} \leq -0.000350539$

Here, the given interval, d, contains the difference value $\hat{d}$, and also it does not include 0 inside the interval. So, we reject the null hypothesis (H0) and accept the alternative hypothesis (H1). And conclude that Classifium performed better than H2O having only XGBoost as its base learner, in terms of classification accuracy in this data set.

B.5.4 Adult

let the classification accuracy obtained from the XGBoost based H2O and Classifium with the same data set, Adult, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.874611 and acc(h2) = 0.87523

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.874611 - 0.87523

$\hat{d}$ = -0.000619

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 24420, so, n = 24420

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.874611*(1-0.874611)}{24420} + \frac{0.87523*(1-0.87523)}{24420}$

$\sigma^2 = \sqrt{0.000008963}$

$\sigma = 0.002993827$

Now, lets find the confidence interval for d

$\mathrm{d} = \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

$\mathrm{d} = -0.000619 \pm 1.96 * 0.002993827$

$\mathrm{d} = -0.000619 \pm 0.005867901$

$\mathrm{d} = -0.006486901 \leq \hat{d} \leq 0.005248901$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the null hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.5.5 Dota

let the classification accuracy obtained from the XGBoost based H2O and Classifium with the same data set, Dota, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.583474 and acc(h2) = 0.597529

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.583474 - 0.597529

$\hat{d}$ = -0.014055

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 51471, so, n = 51471

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.583474*(1-0.583474)}{51471} + \frac{0.597529*(1-0.597529)}{51471}$

$\sigma^2 = \sqrt{0.000009394}$

$\sigma = 0.003064963$

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.014055 \pm 1.96 * 0.003064963$

d = $-0.014055 \pm 0.006007327$

d= $-0.020062327 \leq \hat{d} \leq -0.008047673$

Here, the given interval, d, contains the difference value $\hat{d}$, and also it does not include 0 inside the interval. So, we reject the null hypothesis (H0) and accept the alternative hypothesis (H1). And conclude that Classifium performed better than H2O having only XGBoost as its base learner, in terms of classification accuracy in this data set.

B.5.6 Bank

let the classification accuracy obtained from the XGBoost based H2O and Classifium with the same data set, Bank, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.915894 and acc(h2) = 0.915995

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.915894 - 0.915995

$\hat{d}$ = -0.000101

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 20593, so, n = 20593

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.915894*(1-0.915894)}{20593} + \frac{0.915995*(1-0.915995)}{20593}$

$\sigma^2 = \sqrt{0.000007477}$

$\sigma = 0.00273441$

Now, lets find the confidence interval for d

$\mathrm{d} = \hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

$\mathrm{d} = -0.000101 \pm 1.96 * 0.00273441$

$\mathrm{d} = -0.000101 \pm 0.005359444$

$\mathrm{d} = -0.005460444 \leq \hat{d} \leq 0.005258444$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the hull hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.

B.5.7 Forest Cover Type

let the classification accuracy obtained from the XGBoost based H2O and Classifium with the same data set, Forest Cover Type, be denoted as acc(h1) and acc(h2) respectively.
From our experiment, we found that,

acc(h1) = 0.955329 and acc(h2) = 0.971873

Let the difference between acc(h1) and acc(h2) be represented by $\hat{d}$. So,

$\hat{d}$ = acc(h1)-acc(h2)

$\hat{d}$ = 0.955329 - 0.971873

$\hat{d}$ = -0.016544

Now, lets find the approximate variance of the two observations over their distribution:
Here the sample size is 290505, so, n = 290505

$\sigma^2 = \frac{acc(h1)*(1-acc(h1))}{n} + \frac{acc(h2)*(1-acc(h2))}{n}$

$\sigma^2 = \frac{0.955329*(1-0.955329)}{290505} + \frac{0.971873*(1-0.971873)}{290505}$

$\sigma^2 = \sqrt{0.000000241}$

σ = 0.000490918

Now, lets find the confidence interval for d

d = $\hat{d} \pm (Z_N) * \sigma$

For the larger sample value, according to Tom M. Mitchel on his book 'Machine Learn-ing' at chapter 5, section 5.2.2, value of Z_N should be 1.96 for 95% confidence interval. So,

d = $-0.016544 \pm 1.96 * 0.000490918$

d = $-0.000101 \pm 0.000962199$

d= $-0.001063199 \leq \hat{d} \leq 0.000861199$

Though the given interval, d, contains the difference value $\hat{d}$, but also it includes 0 inside the interval. So, we accept the hull hypothesis (H0) and rejects the alternative hypothesis (H1). And conclude that the both algorithms performed equally on this data set.