AUTOMATIC HYPER-PARAMETER TUNING FOR GRADIENT BOOSTING MACHINE

MR. KANKAWEE KIATKARUN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF ENGINEERING
(COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI
2020

Automatic Hyper-Parameter Tuning for Gradient Boosting Machine

Mr. Kankawee Kiatkarun B.Eng. (Computer Engineering)

A Thesis Submitted in Partial Fulfillment
of the Requirement for
the Degree of Master of Engineering (Computer Engineering)
Faculty of Engineering
King Mongkut's University of Technology Thonburi
2020

Thesis Committee

…… Chairman of Thesis Committee
(Assoc. Prof. Anan Banharnsakun, Ph.D.)

…… Thesis Advisor
(Asst. Prof. Phond Phunchongharn, Ph.D.)

…… Member
(Assoc. Prof. Tiranee Achalakul, Ph.D.)

…… Member
(Asst. Prof. Khajonpong Akkarajitsakul, Ph.D.)

…… Member
(Asst. Prof. Kejkaew Thanasuan, Ph.D.)

Thesis Title	Automatic Hyper-Parameter Tuning for Gradient Boosting Machine
Thesis Credits	12
Candidate	Mr. Kankawee Kiatkarun
Thesis Advisor	Asst. Prof. Dr. Phond Phunchongharn
Program	Master of Engineering
Field of Study	Computer Engineering
Department	Computer Engineering
Faculty	Engineering
Academic Year	2020

Abstract

Machine learning and predictive modeling have become widely used in many fields. Utilizing algorithm without tuning hyper-parameter can lead to inefficient performance of model. Gradient Boosting Machine (GBM) is one of the tree-based models in which the performance can differ greatly depending on its setting. Tuning hyper-parameters of the model requires background knowledge of the algorithm. Moreover, both the performance and cost of the tuning process need to be kept in consideration. In this paper, we proposed an approach based on Genetic algorithm (GA) in process of GBM tuning. The GA is often used in the optimization problem because of the ability to handle more complex problems. We implemented the GA variation called hyper-genetic to tune the hyper-parameter of GBM and explored the best setting possible of the genetic parameters. In addition, we compared our best setting with the results from Grid-search, Bayesian optimization, and random approach over four sets of data. Our proposed algorithm had competitive performance and outperformed the other algorithms in the dataset with a high dimension while requiring smaller computation time at the optimum point on the majority of experimental datasets.

Keywords: Genetic Algorithm/ Gradient Boosting/ Hyper-Parameter Tuning/ Optimization

หัวข้อวิทยานิพนธ์	การปรับแต่งตัวแปรไฮเปอร์แบบอัตโนมัติสำหรับเกรเดียนท์บูสติ้งแมทชีน
หน่วยกิต	12
ผู้เขียน	นายกันต์กวี เกียรติ์กรัณย์
อาจารย์ที่ปรึกษา	ผศ.ดร.พร พันธุ์จงหาญ
หลักสูตร	วิศวกรรมศาตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ภาควิชา	วิศวกรรมคอมพิวเตอร์
คณะ	คณะวิศวกรรมศาสตร์
ปีการศึกษา	2563

บทคัดย่อ

การเรียนรู้ของเครื่องจักรและแบบจำลองการทำนายผลถูกใช้อย่างแพร่หลายในหลากหลายสาขา การใช้งานอัลกอริทึมโดยไม่ปรับแต่งตัวแปรไฮเปอร์สามารถนำไปสู่การทำให้แบบจำลองไม่สามารถทำงานได้เต็มประสิทธิภาพ Gradient Boosting Machine (GBM) เป็นหนึ่งในแบบจำลองที่มีฐานจากต้นไม้ซึ่งประสิทธิภาพสามารถแตกต่างกันมากขึ้นอยู่กับการตั้งค่า การปรับแต่งตัวแปรไฮเปอร์ของแบบจำลองนั้นต้องการความรู้พื้นเพของอัลกอริทึมนั้น มากไปกว่านั้นประสิทธิภาพและราคาต้นทุนในกระบวนการปรับแต่งนั้นจะต้องถูกนำมาประกอบการพิจารณาด้วย ในวิทยานิพนธ์นี้เรานำเสนอแนวทางที่มีพื้นฐานมาจากขั้นตอนวิธีทางพันธุกรรมในกระบวนการประแต่งของ GBM ขั้นตอนวิธีทางพันธุกรรมมักถูกใช้ในปัญหาการเพิ่มประสิทธิภาพเพราะความสามารถในการจัดการกับปัญหาที่ซับซ้อน เราได้พัฒนารูปแบบของขั้นตอนวิธีทางพันธุกรรมที่เรียกว่าไฮเปอร์เจเนติกเพื่อปรับแต่งตัวแปรไฮเปอร์ของ GBM และสำรวจการตั้งค่าที่ดีที่สุดที่เป็นไปได้ของตัวแปรทางพันธุกรรม ในท้ายที่สุดเราได้เปรียบเทียบการตั้งค่าที่ดีที่สุดของเรากับผลลัพธ์จากการทำการค้นหาแบบกริด, การเพิ่มประสิทธิภาพเบย์เซี่ยนและวิธีการสุ่มบนข้อมูลทั้งหมดสี่ชุด อัลกอริทึมที่เรานำเสนอมีประสิทธิภาพในการแข่งขัน และสามารถทำได้ดีกว่าอัลกอริทึมอื่นในชุดข้อมูลที่มีหลายมิติในขณะที่ใช้เวลาในการคำนวณน้อยกว่า ณ จุดที่เหมาะสมที่สุดบนชุดข้อมูลจากการทดลองส่วนใหญ่

คำสำคัญ: การปรับแต่งตัวแปรไฮเปอร์/ การเพิ่มประสิทธิภาพ/ ขั้นตอนวิธีทางพันธุกรรม/ Gradient Boosting Machine (GBM)/

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to my advisor, Asst. Prof. Dr. Phond Phunchongharn, for the insightful advice, remarks, and support throughout the process of this thesis. This research would have been impossible without the aid and opportunities given by Assoc. Prof. Dr. Tiranee Achalakul. Besides, I would like to thank Asst. Prof. Thanis Tangkitjaroenkun, for the commentary and guidance on my writing skills. I would also like to thank the professors and administrative officers who have helped me all these years. Last but not least, I am grateful for all of my companions, and my family, who have provided me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

CONTENTS

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

LIST OF SYMBOLS

SYMBOL		UNIT
$g(z)$	Sigmoid function given linear function z	
p	Probability of class in the dataset	
BS	Brier score loss	
TS	Time Factor	
f_t	Predicted probability	
o_t	Actual target value	
t	Actual time spent	s
T	Total time spent in generation	s
$\overline{TS}$	The average time factor	
MF	Multiplying Factor	
RP	Reproduction pool	
CR	Reproduction crossover ratio	
N_{elite}	The number of elites in generation	
G_i	Generation index number	
G_N	Total number of generations	

LIST OF TECHNICAL VOCABULARY AND ABBREVIATIONS

GBT	=	Gradient Boosting Trees
GA	=	Genetic Algorithm
EA	=	Evolutionary Algorithm
AUC	=	Area-Under-Curve
NLP	=	Natural Language Processing
SVM	=	Support vector machine
CNN	=	Convolutional Neural Networks
GDPR	=	General Data Protection Regulation
GP	=	Gaussian process priors
TPE	=	Tree-structured Parzen Estimator
LightGBM	=	Light gradient boosting machine
HGA	=	Hybrid Genetic Algorithm
ANN	=	Artificial Neural Network
GBE	=	Gradient Boosting Estimator
CMC	=	Contraceptive Method Choice
SETAP	=	Software Engineering Team Assessment and Prediction
GBC	=	Gradient Boosting Classifier

CHAPTER 1 INTRODUCTION

1.1 Problem Statement and Motivation

Machine learning and model prediction have become a well-known topic in recent decades. A lot of research has been conducted in the field to develop and improve various algorithms and methodologies for both academic and industrial applications. Open-source libraries are available everywhere and the pre-build algorithms also help non-expert users to familiarize and apply machine learning models to their problems. Although some pre-build models are performing fair right out-of-the-box, they still depend on the dataset and model configuration. A properly tuned model could have a significant difference performance compared to the random setting, which also depends on the situation, choice of algorithms and data itself.

The sets of parameters that affect the whole learning process of the model are called hyper-parameters. Normally, these sets of parameters are constant values where other model parameters could be adjusted or learned through the process of training. The hyper-parameters differ from model to model. Tuning these parameters can be done in two ways. First, the manual tuning method requires some understanding regarding the model principal. The second option is to use a model framework in the category of auto-tuning or AutoML type. Furthermore, an automated machine learning algorithm is composed of a single learner and multiple learners. In general, the library such as AutoML is created using multiple learner models. In practice, using a single learner model is sufficient in terms of performance and efficient computational cost-wise [1].

The majority of industrial problems are solved by tree-based models. The potential and practical value of these algorithms are recognized in many fields. There are also many variations such as Decision Tree, Random Forest, Gradient Boosting, Extreme Gradient Boosting, etc. In this research, we focus on Gradient Boosting Trees (GBT) which is available on several programming languages and platform-independent. Gradient Boosting Trees (GBT) is similar to Random Forest in the sense that they ensemble the weak learners which are decision trees, using the boosting method for model training in order to minimize the loss function. GBT also inherits the characteristic of tree-based models so it can handle categorical features, not susceptible to outliers because it does not need feature scaling and is capable of capturing feature interaction.

Although GBT can be considered as a strong learner compared to a decision tree, the performance of the model depends on the setting of hyper-parameters. In the worst case, it can lead to considerably low predictive power or overfitting to the training data. Thus, the hyper-parameters of GBT needs to be tuned. The results from the model-based optimization of hyper-parameter can yield results that exceed the counterpart from the human-expert [2]-[4]. Similar research also involves hyper-parameter tuning as part of the automated machine learning framework in [5]. Hence, the objective of improving the predictive performance of GBT is to fine-tune the hyper-parameter using an optimization algorithm.

In this thesis, we propose a hyper-parameter tuning method for Gradient Boosting Classification problems using a Genetic algorithm called hyper-genetic. Genetic algorithm (GA) is an evolutionary algorithm (EA). The proposed GA variation has been applied to the concept of dynamic mutation to control the elements of randomness in the

reproduction process with a touch of migration concept to further expand search space. Our fitness function is composed of multiple model performance metrics and computation time factors to control the cost of the tuning process. In other words, the objective is to maximize the capability of the Gradient Boosting Classifier while consuming less computational time. To evaluate the performance of our proposed method, we compare the performance with Grid-search, Bayesian optimization, and random approach in terms of area-under-curve (AUC) scores and computation time.

1.2 Objectives

1. To study the machine learning algorithm and hyper-parameter optimization.
2. To study the concept of Genetic Algorithm and applicable modification for GA
3. To design and develop the hyper-parameter optimization framework for Gradient Boosting Classification problems using a Genetic algorithm.
4. To evaluate the performance of our proposed framework against other hyper-parameter optimization techniques.

1.3 Research Scopes

1. The scope of model to optimize only focuses on traditional GBT algorithm.
2. The hyper-parameter optimization technique and machine learning libraries are created by Python and open-source library.
3. The dataset used to evaluate the performance comes from UCI Machine Learning Repository. The target variable is only applied to binary classification.

1.4 Organization of Thesis

The rest of this thesis is organized as follows. Literature review and related work on GA and hyper-parameter optimization are introduced in Chapter 2. Chapter 3 describes our proposed system model and overall design with the validation method. Chapter 4 contains the evaluation and result of model performance. Finally, the conclusion will be in Chapter 5.

CHAPTER 2 THEORETICAL ISSUE/RELATED WORK

In this chapter, we introduce literature review and related studies in three sections. In the first section, we present the introduction to the Gradient-Boosting classification model. The second section covers hyper-parameter optimization and its application. The last section reviews the evolutionary algorithm and genetic algorithm.

2.1 Review on Different Type of Classification Models

The classification model is the machine learning model that attempts to classify or categorize the given input data into sets of specific classes. The trained model will be applied to the new datasets and the class for it will be predicted with certain probability. Gradient Boosting Tree is one of those that belongs to the tree-based algorithm. There are many types of Classification models and each one of them might be suitable for certain types of problem.

2.1.1 Logistic Regression Model

The Logistic Regression belongs to the supervised learning classifier model, which uses the regression model to predict the probability of data set belonging to class "1". The Logistic Regression model uses Maximum likelihood to fit sigmoid function in (2.1) to determine the cut-off threshold for the target prediction.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

$g(z)$ represents sigmoid function given linear function z. Logistic regression has a decent performance in classification problem making it one of the popular models in the field of Natural Language Processing (NLP). However, this model can only perform in the binary classification problem and all features must be independent.

2.1.2 Decision Tree Model

The decision tree is the first and the simplest algorithm in a tree-based classification model. The decision tree is a classification tree where each non-leaf node represents features of the model. Each node will have a split that corresponded to its possible values of specific features that lead to the next decision node or leaf node. Leaf node will contain the probability distribution of each class at that node as shown in Figure 2.1. Decision Tree is a common decision making tool because it can handle the data with outliers and has high explicability. However, it is still considered a weak learner and can be further improved.

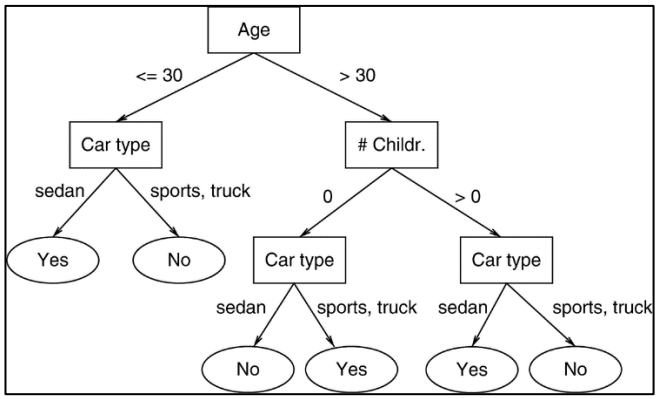


Figure 2.1 Structure of decision tree [9]

2.1.3 Random Forest Model

One of the ensembles of decision tree uses bagging method or bootstrap aggregating. Each decision tree inside the model will be trained on different sub-dataset picked randomly from the original dataset and sampling to the same size. The results from every tree will come from the majority vote of all predicted results (shown in Figure 2.2) to reduce the overfitting issues and to generate the better accuracy.

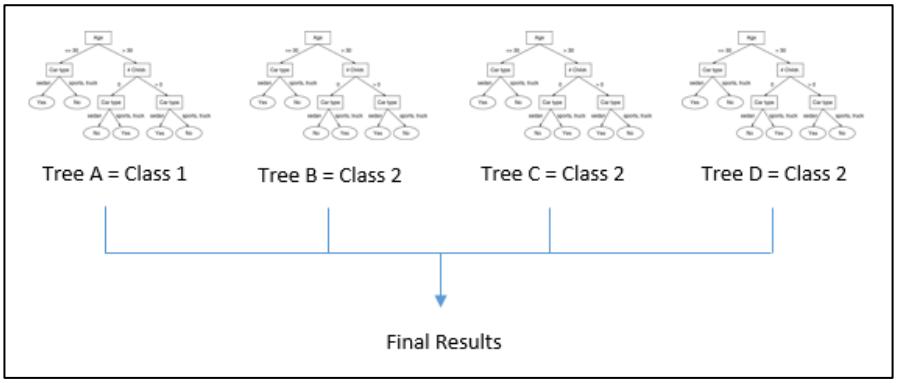


Figure 2.2 Random forest algorithm

The methodology used to create the sequences of rules for the decision tree is entropy and information gained. Entropy is the measurement of homogeneous in the dataset's variable as shown in equation (2.2). Zero entropy represents the dataset that contains only single class, while the dataset composed of different classes will have higher entropy.

$$Entropy(p) = -\sum_{i=1}^{N} p_i \log_2 p_i \tag{2.2}$$

In entropy equation, p represents the probability of class in the dataset, i is the index of class and N is the total number of all the classes.

Information Gain (IG) is the difference of entropy in each decision branch. This will help the decision tree to select the best splitting feature for each node that can clearly separate data from one another. The most impact feature that can split the dataset will be placed on the top of the decision tree. IG will be one in the case of feature that can totally separate

the data or entropy in both leaf node are zero. The process of splitting node will reoccur until the stopping criteria or the max tree depth reached.

2.1.4 Support Vector Machine Model

Support vector machine (SVM) is a supervised machine learning algorithm that is a non-probabilistic binary linear classifier. The SVM model maps the data sample to hyperplane and attempts to separate the target variables into classes. Finding the best hyperplane that can divide the data with the biggest gap or margin can lower the generalization error of the classification. Hyperplane can be in different forms which are determined from the characteristics of dataset to select the suitable kernel function. The prediction will be generated from the placement of data into hyperplane and decided from which side of the gap that sample belongs to as shown in Figure 2.3.

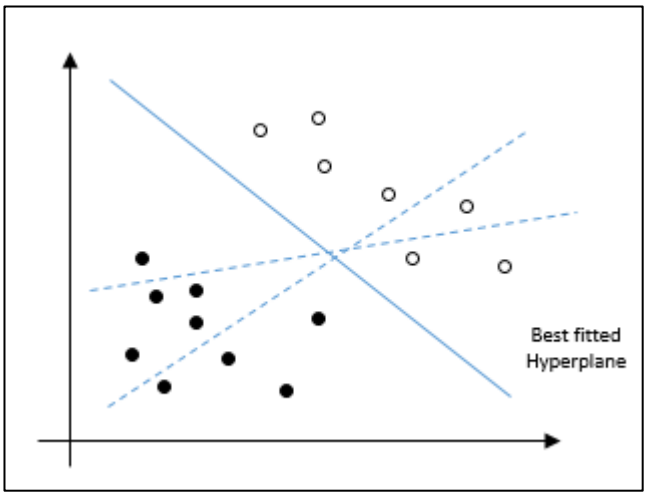


Figure 2.3 SVM hyperplane mapping

2.2 Review of Predictive Modeling

Recently, Deep Learning, Convolutional Neural Networks (CNN), and other Neural based algorithms are often mentioned in many research areas with their high predictive power and capability to capture the intricate pattern of relations. A lot of research has applied the concept of these state-of-the-art algorithms to make a breakthrough in various fields, such as the famous Go player with deep neural networks [6]. In contrast, many real-world or industrial problems are not suitable to bring out the potential of an advanced algorithm. Both the data and computational power in question and the complexity of the algorithm make it less transparent or unexplainable. In many cases, the results from the predictive model are meant to be utilized by a data user and they need to understand what is happening inside to fully bring the potential of provided insight. The importance of transparency has developed to a new height as questioned by [7] after Europe implemented the new General Data Protection Regulation (GDPR) in 2018.

Hence, that explains why GBT becomes our choice of optimization due to the popularity of the algorithm itself and readiness in many built-in data platforms. Many big data analytic engines such as spark environment come with their machine learning tools or library which include GBT. These tree-based algorithms can be found throughout many analytic tools or data workbench. The fact that GBT relies heavily on hyper-parameter setting also highlights the significance of attention required for its optimization processes.

2.2.1 Review on Gradient-Boosting Tree

One of the support tools for decision making that everyone is familiar with is a decision tree. Every decision tree consists of probability for each sequence of condition to happen that will lead to the certain goal. It used to be a common tool in machine learning before it was improved by many ensemble methods. The ensemble method enhances the traditional decision tree by combining the predictive power of each decision tree to generate more accurate and powerful results. The two well-known ensemble methods for decision trees are the bagging and boosting methods. The concept of the bagging method is decreasing the variance of a single decision tree with the quantity. The overall data will be divided into many subsets to be trained by different decision trees and the final result will come from the aggregate result of the outcome of every decision tree either by means or majority votes. On the other hand, the boosting method uses multiple decision trees to train sequentially and adjust the weight on each iteration to increase the chance of correcting the previous error. The final result will be the stronger predictor compared to a single decision tree. There is also a different way to categorize these ensemble methods by their sequence of trials. The bagging method can be represented as a parallel ensemble because each iteration can be trained separately and simultaneously while the boosting method is called the sequential ensemble because of how it exploits the information of each iteration to the advantage of the following one as in Figure 2.4.

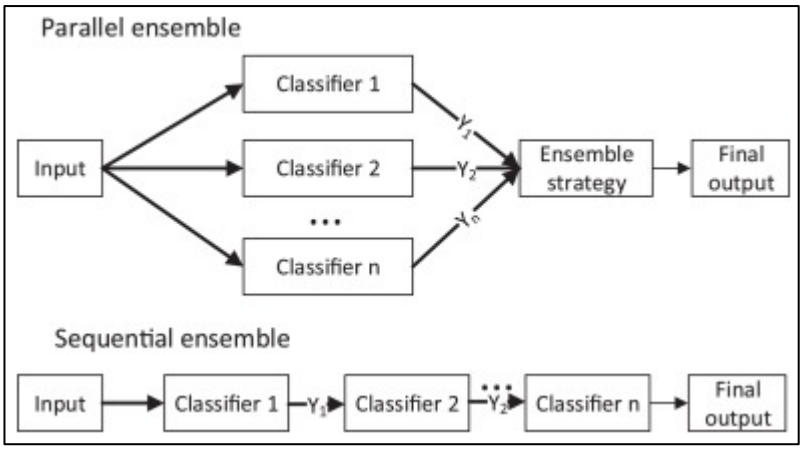


Figure 2.4 Parallel ensemble and sequential ensemble method [4]

One of the many popular variations using the boosting technique is Gradient Boosting Tree. The GBT uses the gradient descent technique to optimize loss function for a better performance in each iteration. Results from one generation will impact the weight of the data in the next generation by giving a lower weight to the corrected prediction result and the incorrect prediction will be assigned with more weight. This process will ensure that the consecutive generation model will likely pick up the error from the previous one. According to Figure 2.5, we can see that starting from the left dataset, all data points are given an equal weight. As the training proceeds, the positive data points that were misclassified will gain a bigger weight compared to the rest of the data points so the model can focus on the error. At the end of the processes, all of the iterations will be rated by their accuracy and contribute to the final prediction.

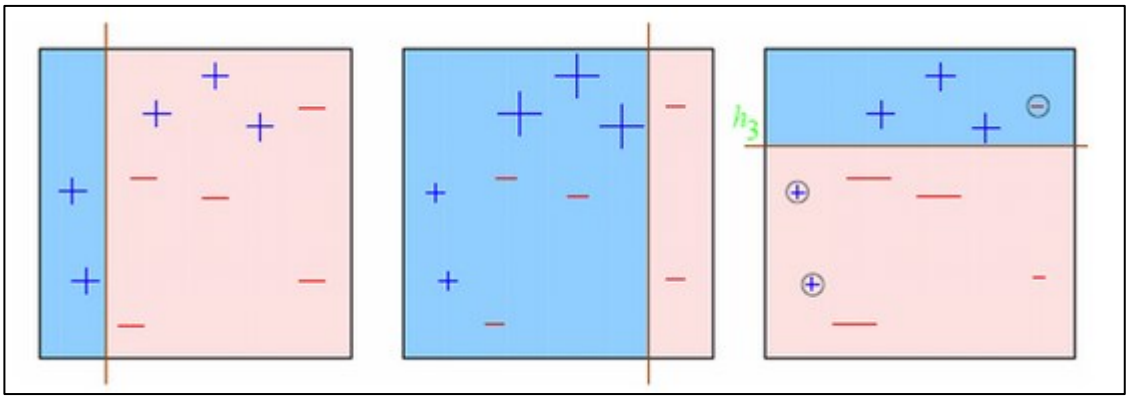


Figure 2.5 Weight adjustment of GBT [8]

2.3 Hyper-parameter Optimization

The processes of hyper-parameter tuning can be accomplished in many possible options. The most primitive way is by manual configuration, which takes a lot of time and effort to manually tune and test each set of hyper-parameter to find the best setting. This might not be possible to achieve in an environment that has a lot of parameters with very long training time and could also lead to not finding the best parameter setting at all. Therefore, a more practical methodology is needed to ensure the likelihood of discovering the best parameter setting within a feasible time slot. The most basic parameter tuning process is searching the parameter within search space. This set of combined parameters can be tested following the search algorithm such as Grid search and Random search.

2.3.1 Review on hyper-parameter search algorithm

Grid search is a traditional hyper-parameter optimization algorithm that implements the exhaustive search in a given search space of the specific model's hyper-parameters. It will sweep through the combination of the parameter to evaluate the performance with a predetermined function. Even though it can reach the optimum point, Bergstra [10] has mentioned that the Grid search wasted trial number is exponential to the number of dimensions. This signifies that the Grid search method is not feasible in the high dimensional search space. It can take too long to reach the ideal spot or it might not reach there at all due to the burden of computational resources. Figure 2.6 depicts that the parameter grid in the example of two-dimension search space might seem evenly covered but it might not evenly be distributed in the subspace of parameters as the result from Random search can produce.

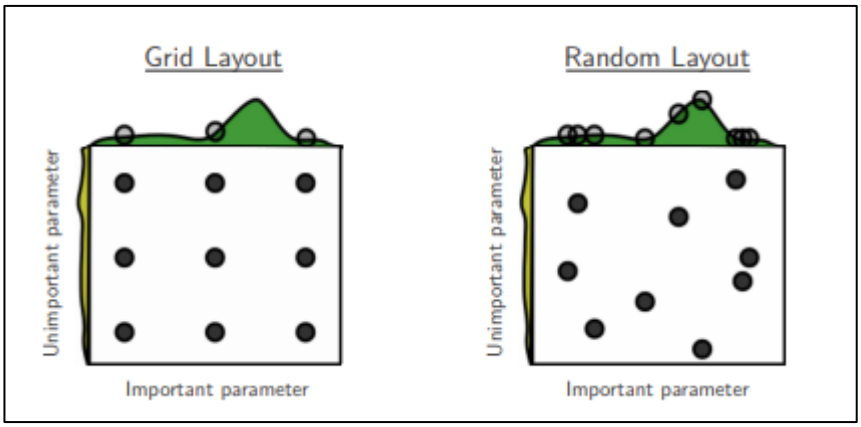


Figure 2.6 Grid search and Random search comparison using nine trials of optimization [10]

2.3.2 Review on Bayesian optimization

Thomas [5] developed the automatic machine learning framework using Bayesian optimization on Gradient Boosting Tree to adapt with any given data input. Bayesian optimization is a searching algorithm for global optimization that applies the probabilistic approach to enhance its learning processes. The result from one iteration is used to help the next searching point to reach closer to the optimum hyper-parameter setting thus resulting in less computation as seen in Figure 2.7. Grid search is an exhaustive search method and Random search cannot guarantee the best possible outcome while Bayesian optimization uses the prior result to its advantage when selecting the next search point.

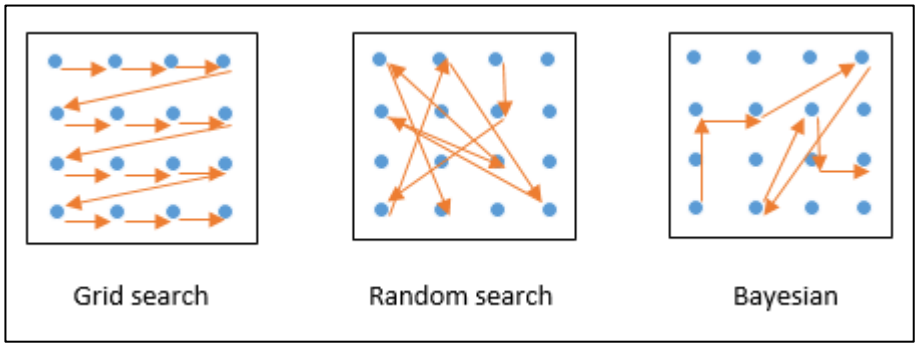


Figure 2.7 Grid search vs Random search vs Bayesian optimization

Snoek, et al. [2] implemented the Bayesian optimization with the Gaussian process priors (GP) on some of the generally applied machine learning. Their model has the capabilities of parallelization and is also able to handle the various time regime. The results of his experiment are better than the human expert tuning method in most cases and successfully beat the state of the art at that time by 3 percent. Yufei [4] applied another variation of gradient boosting machine called Extreme gradient boosting (XGBoost) to the credit scoring problem. This model has a data preparation process and a model-based feature selection before the hyper-parameter tuning using Bayesian optimization as shown in Figure 2.8. Instead of GP, they used Tree-structured Parzen Estimator (TPE) approach that optimizes parameter space in the tree structure. The final result can outperform other baseline models and is even comparable to the state-of-the-art parallel learning model.

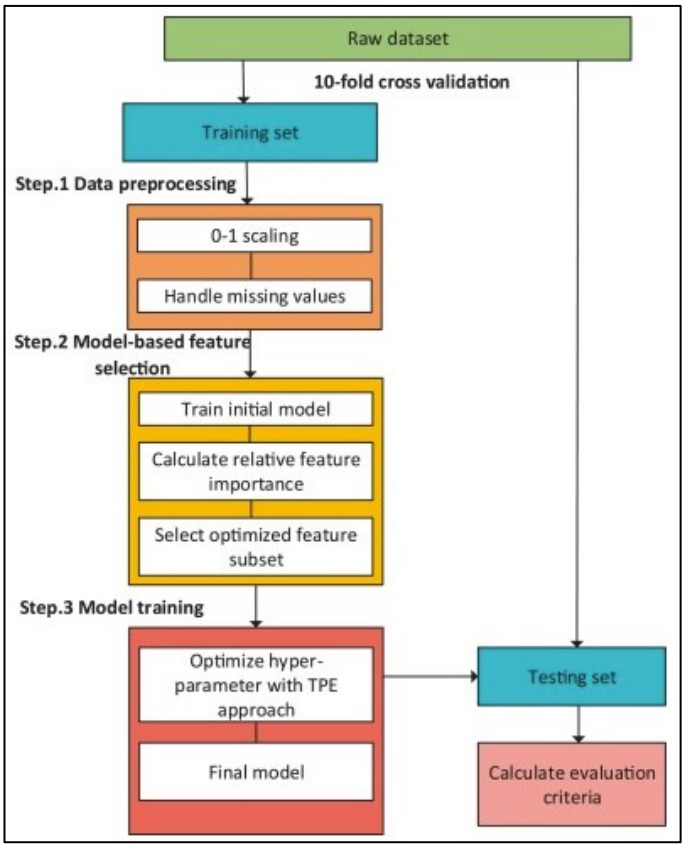


Figure 2.8 Credit scoring model with TPE workflow [4]

2.3.3 Review on other tree-based model optimization

We have mentioned Yufei's works in the previous section about the hyper-parameter optimization for XGBoost earlier. This section will focus on hyper-parameter tuning of other tree based algorithm.

Kumar [11], used Features Weighting and hyper-parameter tuning to create a promising predictive model for the breast cancer prognosis and diagnosis. They used the Kernel Neutrosophic C-Means Clustering for Feature Weighting, which would assign more weight to an applicable features while the less applicable feature would be assigned with less weight. The overall workflow of the breast cancer classifier is shown in Figure 2.9. The Random Decision Forest classification model tuned by the Bayesian optimization can produce the satisfying result in model performance. It has a lower error rate and higher precision in all of the compared methodologies which is a significant improvement.

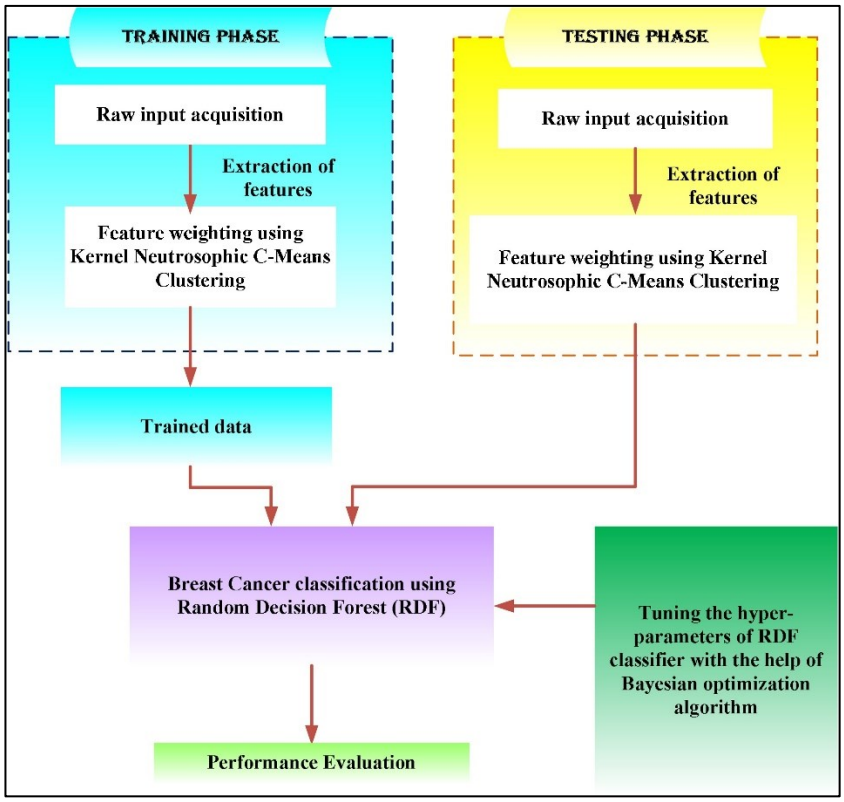


Figure 2.9 Simplified workflow of breast cancer classifier [11]

Nayak [12] introduced the light gradient boosting machine (LightGBM) for hand gesture recognition that is efficient in terms of cost and performance. Their LightGBM model used a modified version of memetic firefly algorithm to optimize the hyper-parameter. They proposed to integrate the perturbation operator into the memetic firefly algorithm to avoid the local optimal issues from the traditional firefly model. The proposed model was evaluated by various measurement metrics in the comparative analysis with different ML algorithm and yielded a dominant result compared to the rest.

2.4 Review of Genetic Algorithm

Genetic Algorithm (GA) is one of the Evolutionary Algorithms that is a nature-inspired population-based metaheuristic optimization. The GA applied the concept similar to Charles Darwin's theory of evolution and natural selection to solve the optimization or search problems using biological operators. The first step of GA is to define a genetic representation or substitute of chromosome in biology as a set of configurations that has to be optimized. These genetic representations will be evaluated and given the score from the predefined fitness function. The selected individual with the most preferred score will go on to the reproduction phase to make a new generation with more potential while the worst performed one will get eliminated in natural selection as shown in Figure 2.10. This process will be reoccurred until the terminal condition is met. The best-fitted individual in the last generation will be represented as the best candidate for initial problems. There are many variants of GA with different implementations in the reproduction processes.

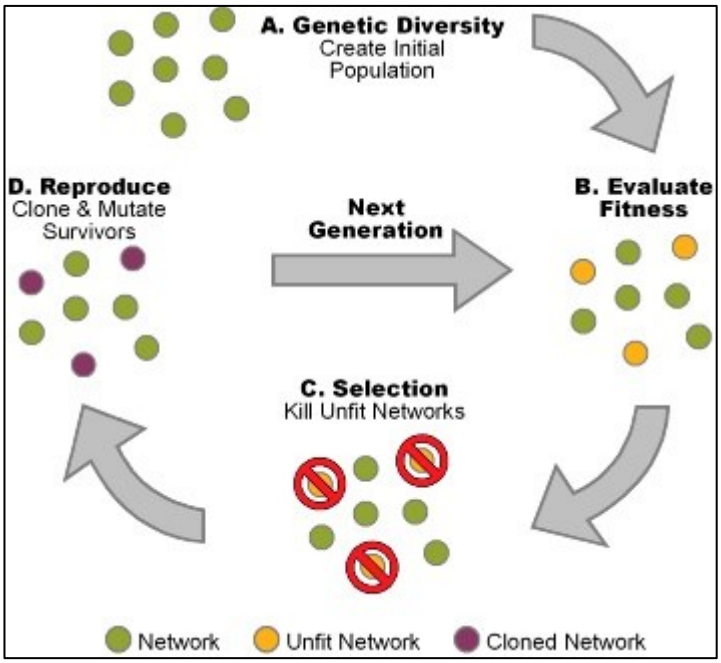


Figure 2.10 Genetic algorithm natural selection cycle [13]

Guo, et al. [14] created the optimization design using an enhanced version of GA called the Hybrid Genetic Algorithm (HGA). They also applied the GA and HGA to the interval optimization to shorten the slope calculation time in interval analysis. Miranda [15] implemented an adaptive genetic algorithm to optimize the discrete event simulation. They experimented on the parameter of GA and found that population sizes and the number of generations had the most impact on computation time and precision of the model. Hence, they proposed the strategy to adaptively adjust these parameters of GA while training to improve the time to converge the final result. Fridrich [16] used GA on the field of hyper-parameter optimization of Artificial Neural Network (ANN) in churn problems. The final result of implementing GA on ANN has highlighted its promising performance in enhancing ANN churn model predictive ability.

2.5 Literature Review Conclusion

This section is the summary of the mentioned research and theories related to this work. There is a lot of state-of-the-art advance machine learning built to solve different problems. However, these models are often tailored for specific problems and the more advanced the model, the more complex it will become. The complex model despite having a stunning predictive performance, is like a black-box model which is hard to explain.

The proposed work focuses on GBT because it has decent predictive performance while retaining the explicability of the tree-based model. Even though the GBT has better performance compared to the traditional Decision Tree, it requires the decent setting of model hyper-parameter. There are many studies on the topic of hyper-parameter optimization or parameter tuning but there are not many papers that specifically focus on GBT. The common parameter optimization methods for the tree-based model are Bayesian Optimization, Random search, etc. These popular optimization algorithm might be able to optimize the model but the cost and time needed to achieve the best performance are questionable. This thesis proposed to implement the famous nature-inspired Genetic Algorithm to the GBT classifier. GA has advantage of modular

implementation, concept easy to parallelize and a large search space. Another reason that encourage the usage of GA is that our GBT parameters have a combination of float and integer variables. Our proposed algorithm has a total of six GA parameters setting. The parameter n_estimator and max_dept are integer while learning_rate, min_samples_split, min_sample_leaf, and subsample are all float type parameters. Our framework using GA with some modification can generate a better performing model with desirable computational time.

CHAPTER 3 PROPOSED WORK

The concept of Gradient Boosting and hyper-parameter optimization including Grid search, Random Search, and Bayesian optimization algorithm was introduced in the previous chapter. The advanced Genetic Algorithm was also mentioned to accentuate the potential to further enhance the capabilities of the selected model. We proposed the design of a hyper-parameter tuning framework for GBT, using the modified version of the Genetic Algorithm to acquire the best performance with the minimum computational time.

3.1 Overall Hyper-Parameter Optimization Framework

The overall design of our proposed framework is shown in Figure 3.1. It contains data preparation at the beginning followed by the Genetic algorithm section. Data preprocessing and prepare cross-validation are included to clean the data and assign an index for each cross-validation while the genetic section will continue until the stopping criteria are met.

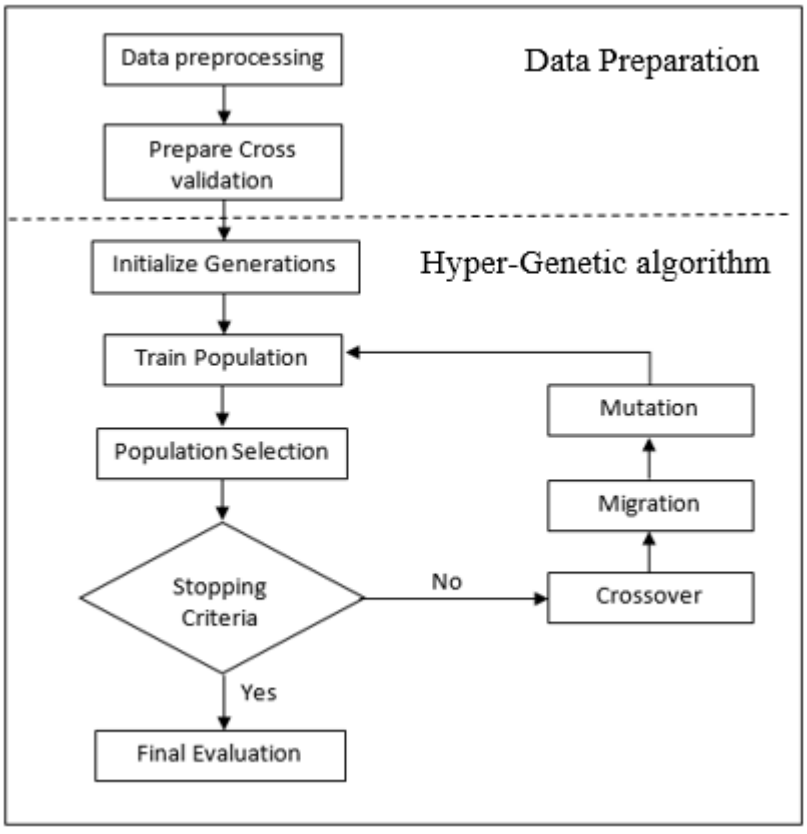


Figure 3.1 Proposed model framework

The data preparation section only contained two processes: Data preprocessing and Prepare Cross-validation. First, Data preprocessing process would get the input datasets from the provided data path and then check for missing values in every column. The process to replace NA would take place to either fill the missing values with mean or most occurred values. In the case of categorical features, it would be indexed by the transformer first before putting it into the model iteration. Then, we prepared the data by

splitting one set of 25% for test model performance while the rest would be partitioned to the number of cross-validation for the model training. The overall steps of our proposed model framework from Figure 3.1 are described below.

1. Data Preprocessing: In this step the input data would be handled according to the data type. Numerical data would be impute missing with the average and median while categorical data will be replaced with mode. All the features included in model would be index transformed.
2. Pre Cross-Validation: The transformed data set would be split into 25 percent for validation and the rest split with cross-validation libraries for model training.
3. Initialized Generations: This is the initialization step for GA to define the population before going through the training step.
4. Train Population: Fit the GBT model to the training data set and run scoring performance.
5. Population Selection: Ranking all individual in the population according to fitness function and select the eligible population for reproduction processes.
6. Crossover: The generation that did not meet the stopping criteria would select pairs of individual to create new population in a reproduction pool.
7. Migration: An additional step to introduce the element of randomness into the reproduction pool.
8. Mutation: Another core step of reproduction process to explore a new combination of configuration.
9. Final Evaluation: At the end of reproduction process the best individual would be selected to represent the result of optimization process.

The detailed description of our proposed design framework will be described in the next section.

3.2 Proposed Hyper-Genetic Algorithm

In the proposed model, we implemented a variation of the Genetic Algorithm to optimize the hyper-parameters for the sklearn Gradient Boosting Estimator (GBE). Our approach is similar to [5] as we applied machine learning over Gradient Boosting. However, our proposed model used GA over the Gradient Boosting classifier for binary classification problems only.

3.2.1 Parameters of Gradient Boosting Trees

In this proposed framework, our Genetic algorithm had each genetic representation of the set of solutions called individuals. Every individual contained six hyper-parameters for GBE:

- learning_rate: determines the contribution of each decision tree with a trade-off relationship between n_estimator
- n_estimators: is the number of trees to perform boosting. Even though the GBE is robust, it can still lead to over-fitting if the parameter is too high.
- max_depth: is the maximum depth of each tree. It can be used to counter over-fitting.
- min_samples_split: is the minimum number of samples to consider splitting the internal node.
- min_samples_leaf: is the minimum number of samples to be in a leaf node.

- subsample: is the fraction of the sample to be performed in each tree. Reducing the subsample rate can reduce variance but also increase bias.

At the initializing phase, all individuals in the first generation would be random and then trained on the model. For this training set, we set the number of cross-validation to 5. Then, each individual was given a fitness score according to the fitness function. The genetic selection processes would continue until the stopping criteria were satisfactory. The stopping criteria were three iterations when the best fitness score did not change.

3.2.2 Fitness function definition

In order to set criteria for the best-fitted individual, we formulated the fitness function to be used on each individual. Our proposed fitness function came from the composition of the area under the ROC curve (AUC), brier score loss (BS) and a fraction of the time spent on each training of the individual. The addition of the time factor (TS) was to control the time using in the tuning process.

$$BS = \frac{1}{N}\sum_{t=1}^{N}(f_t - o_t)^2 \tag{3.1}$$

The Brier score is also a type of score function for the accuracy of binary probabilistic prediction as shown in (3.1) where (f_t) is the predicted probability and (o_t) is the actual value. The lower score represents better prediction results. The time factor (TS) can be obtained by (3.2) with calculating actual time spent (t) over all the individual's time spent in the same generation (T).

$$TS = \left(\frac{t - \min(T)}{\max(T) - \min(T)}\right) \tag{3.2}$$

Where $T = t_i \dots t_N$

$$fitness\ score = \ (AUC \times (1 - BS)) + \left(\left(\overline{TS} - TS\right) \times MF\right) \tag{3.3}$$

The final fitness score is shown in (3.3) where the average $\overline{TS}$ is the average time factor and MF is multiplier factor which was configured to 0.3. Then, the fitness score in (3.3) would be used to evaluate individuals in each generation and used for best-fitted selector for the reproduction process.

3.2.3 Reproduction processes

Every generation that did not meet the stopping criteria would proceed to the reproduction phase which had three steps. First, the best performance or best-fitted individual would be preserved to the next generation according to the parameter depicting the number of elite. The number of population in the reproduction pool (RP) would be created from the total population deducting the number of elite multiplied by the crossover ratio (CR) to calculate the number of crossovers that would take place as seen in (3.4).

$$RP = crossover + migrants \tag{3.4}$$

Where

$$crossover = (N - N_{elite}) \times CR \tag{3.5}$$

$$migrants = ((N - N_{elite}) \times (1 - CR)) \tag{3.6}$$

N is the number of population and N_{elite} is the number of elites. CR is the crossover ratio.

The total number of reproduction pools would equal the number of populations and then proceed to the mutation process to create new genes in each generation.

Typically, there are multiple options for crossover including single-point crossover, multi-point crossover, and uniform crossover. The simple crossover might lead to a product of illegal offspring but this can be avoided by using a modified version of uniform crossover which randomly selects each configuration in each individual to generate a new population as shown in Figure 3.2.

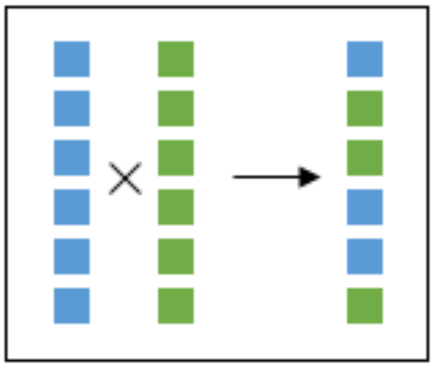


Figure 3.2 Uniform crossover concept

In this proposed model, we used random selection with equal weight to all the population for uniform crossover and after the reproduction pool was filled with new individuals we would fill the rest of the reproduction pool with the migration process. The migration process would create a new individual to increase the diversity of the population and help control the effect of the dynamic mutation.

3.2.4 Dynamic mutation

The mutation is the biological evolution mechanism to increase diversity and introduce a new set of genes into the population. It is an important process to help the genetic algorithm to overcome the local optimum. Increasing the mutation rate can result in more random components and improve the chance to find global optimum. However, too much mutation rate can lead to the divergence problem. To solve this problem, we applied a dynamic mutation rate in [17-18] where the mutation probability could be adjusted over the passing generation. The mutation rate (MR) would start at high value and every generation would recalculate MR by using the current generation number (G_i) over the total number of generations (G_N) as shown in (3.7).

$$MR = 1 - \left(\frac{G_i}{G_N}\right) \tag{3.7}$$

Our mutation rate would be decreasing over time and assist the convergence to the optimum. The mutation process was also applied to increase search space at the initial stage. The population pool that went through the mutation process would merge with the elite population from the current generation to create new populations for the next generation.

3.3 Performance Evaluation

This thesis focuses on improving two aspects of GBT which are computation time and predictive performance. The first aspect is the time required to reach the stopping criteria of the hyper-parameter optimization model. Given the same environment setup, the faster model will spend less cost and wasted trial of parameter tuning which is detrimental to the optimization decision.

The second aspect of evaluation is the performance of the model. The traditional accuracy is the ratio between the accurately predicted observation and the total number of observation. This can create the misinformation when the accuracy is used to evaluate the imbalance dataset. In the highly imbalance dataset with the ratio of 9 to 1, the model can reach 90 percent accuracy with just predicting all the observation as the majority class. Thus our research applied the concept of the area under the receiver operating characteristic (ROC) curve or AUC to use as a measurement for model comparison. The ROC is a graph that displays the performance of classification model by plotting the True Positive Rate (TPR) and False Positive Rate (FPR) as shown in Figure 3.3.

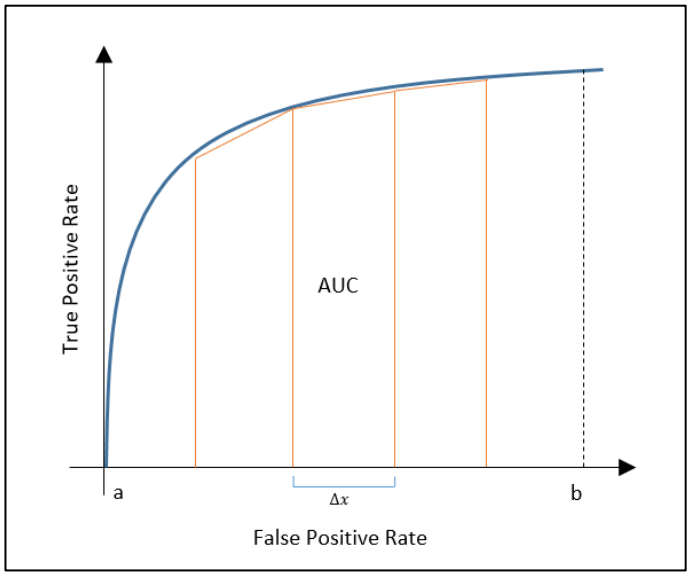


Figure 3.3 Area under the curve of ROC

The formula of TPR and FPR are represented in equation (3.8) and (3.9) respectively. TP is the True Positive rate or number of correct positive observation, which means the capability to accurately predict the class is true when it is actually true. TN is the true negative or number of correct negative observation, suggesting that the model can predict the class is False when it is actually false. False Positive (FP) is the number of incorrect prediction of positive observation, which means the model would predict the class True when it is actually false. False Negative is the number of incorrect negative observation, signifying that the model would predict the false when it in fact belonged to the class True.

$$TPR = \frac{TP}{TP + FN} \tag{3.8}$$

$$FPR = \frac{FP}{FP + TN} \tag{3.9}$$

The AUC would measure the two-dimension area under the ROC curve to capture the aggregate measure of performance of the classification model. We used the sklearn function to calculate the AUC using trapezoidal rules as shown in equation (3.10). It divided the area under the ROC curve into subinterval x_k of range $[a, b]$ as shown in Figure 3.3. Each interval was estimated as multiple trapezoids and used to calculate the area while Δx represented the length of each subinterval in the total of N subintervals.

$$AUC = \int_a^b f(x)dx \approx \sum_{k=1}^{N} \frac{f(x_k - 1) + f(x_k)}{2} \Delta x_k \tag{3.10}$$

The AUC had a possible range from 0.5 to 1.0 where the higher the score referred to the accurate model performance. AUC was robust because its calculation involved all possible classification classes, making it one of the most popular measurement metrics for model evaluation.

3.4 Experimental Design

This section explains about the dataset used to evaluate the performance, experimental setup, design of experiment, and the hardware specification and software specification of this experiment.

3.4.1 Dataset description

To evaluate the performance of our proposed method, we used four different datasets, one of which belonged to the Stanford CS109 with the problem set 5 and the rest were from UCI machine learning repository. The summary of four datasets used to measure the performance is shown in Table 3.1.

Table 3.1 Dataset Description

Dataset	**Number of class**			**Number of attributes**
	False	***True***	***Total***	
TTN_train	416	251	891	1,732
TTN_test	131	90		
CMC_train	482	622	1,473	10
CMC_test	147	222		
SETAP_train	400	194	793	85
SETAP_test	130	69		
DOTA_train	43,868	48,782	102,944	113
DOTA_test	4,792	5,502		

The description of all the dataset used are explained below:

1) TTN: The first dataset was used in CS109, the Titanic dataset from Kaggle competition. It provided information on the passenger boarding on the ship and labelled columns to determine whether passengers had survived the shipwrecking event. We selected the file that contained 891 rows of data. This dataset had a total of 12 columns that described the characteristics of all passengers, which expanded to 1732 columns after indexing.

2) CMC: The second dataset, the Contraceptive Method Choice (CMC) was a subset of the National Indonesia Contraceptive Prevalence Survey in 1987. This data contained demographic and economic information of married women and the choice of their contraceptive method. Originally, the attribute of the contraceptive method had a total of three classes that were converted to contraceptive user and non-user to comply with binary classification metrics.

3) SETAP: It was collected under the Software Engineering Team Assessment and Prediction (SETAP) project. The nature of this dataset was the information of each team performance in a class separated by interval and the score that they gained at the end.

4) DOTA: The final dataset was Dota2 games results. It described the match in a specific region, game type, character selection, and the final results. This dataset was sparse since the combination of characters could be selected 10 from 113 available characters per instance. This resulted in a challenge for feature interaction.

3.4.2 Experimental setup

In order to measure the performance of the proposed algorithm, we compared the result of the hyper-parameter tuning algorithm with other methods. We implemented hyper-parameter optimization with sklearn Grid Search with Cross-Validation, Bayesian Optimization with hyperopt [19], and one set of hyper-parameter from random settings as a baseline model. Grid-search is a traditional exhaustive searching method that sweeps over all the parameters manually provided in the parameter grid while evaluating every parameter combination with cross-validation for the best performer. Bayesian optimization is different from Grid-search as it does not sweep through all possible parameters but using the information from the previous iteration to help determine the direction of optimal value. The details of each methodology are presented in Section II of the document. Each algorithm will be fitted on three fold cross-validation datasets.

3.4.3 Design of experiment

Three experiments were used to evaluate the proposed framework of hyper-parameter tuning for GBT. These experiments were evaluated with three folds cross-validation and the result would come from an average of four experimental runs. In the first experiment, we compared the impact of each model parameter on the framework performance. In this test, the default parameter set in Table 4.1 was configured to be similar to the adjustment to each parameter to observe the effect on performances. To evaluate each parameter, the range of the possible parameter was defined and the model fitting would repeat in the given range. The result were to be displayed in a graph to select the best combination of setting to use for the comparison test.

In the second experiment, the AUC performance of each hyper-parameter tuning algorithm was compared using default parameter set to perceive the difference in performance between the four setups. Each test algorithm would be evaluated on the dataset provided in Section 3.1. The result table would be used to analyze the ranking of performance between algorithms.

Finally, the last experiment compared both the AUC performance and computational time of each algorithm with the hyper-genetic algorithm, using the GA parameter set that had been optimized for each specific dataset. The comparison on computation time included the tuning time of the hyper-genetic algorithm according to the provided guideline.

3.4.4 Environment specification

This experiment was performed on Window 10 operating system. Table 3.2 presents the detail of hardware and software libraries specification in the testing environment. The proposed hyper-genetic frameworks were developed by Python with open-source libraries as described below.

1. Pandas is a popular python package that provides data structure and data manipulation tools. The capabilities of this package cover a wide range from loading the data to preprocessing.
2. Numpy is an open-source python package used for working with an array or multi-dimension dataset. It also has a function for linear algebra, etc.
3. Scikit-learn is a python package built on Numpy and SciPi. It provides many advanced and reliable machine learning packages that cover many domains such as classification, regression, clustering, etc.
4. Hyperopt is a distributed asynchronous hyper-parameter tuning python package. It provides Bayesian Optimization for parameter tuning.

Table 3.2 Hardware and software specification

Operating System	Window 10 Education 64 bits
CPU	Intel® Core™ i5-6600 @ 3.30GHz
Memory (RAM)	16 GB
Programming Language	Python 3.7.7
Machine learning package	Pandas version 1.0.4 Numpy version 1.18.1 Scikit-learn version 0.22.1 Hyperopt version 0.2.4

CHAPTER 4 ASSESSMENT AND RESULT

The hyper-genetic framework was developed according to our proposed design to create better performance for GBT automate tuning. In this chapter, we describe our evaluation methods and the final results of our framework to inspect the benefits of this proposed model and possible future improvement.

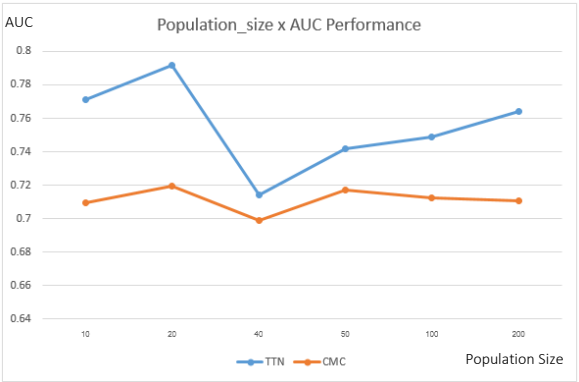
4.1 Assessment Design and Evaluation Method

In this section the hyper-genetic framework was experimented to find the impact of each parameter and to find the best parameter setting. We studied the effects of GA parameters over the performance of the proposed method on two datasets which were TTN and CMC. The performance was evaluated using two metrics which were Area Under Curve (AUC) of the model and computation time. The default setting for GA parameters is shown in Table 4.1.

Table 4.1 Default Parameters Setting

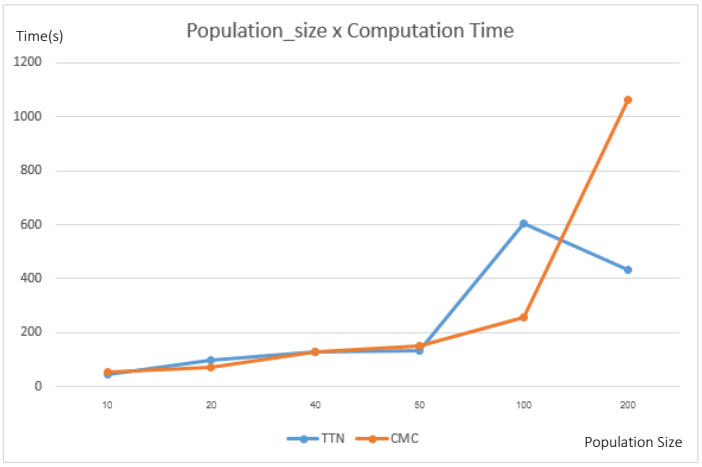
Parameter	Default Values
Number of population	20
Number of generations	40
Number of elites	4
Crossover ratio	0.9
Stopping criteria	3

1) Population size: This experiment measured the effect of the population size. The model performance result and computation time are shown in Figure 4.1a and Figure 4.1b, respectively. We could see in Figure 4.1a that the AUC performance was the highest at the population size of 20. Furthermore, Figure 4.1b also manifested that the population size directly affected the computation time. The more population size meant more computation needed per iteration.



(a) Population size effects on AUC performance

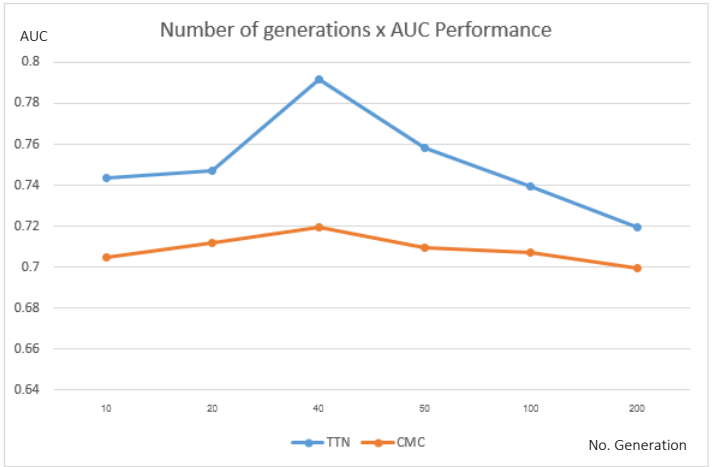
Figure 4.1 Population size effects on CMC and TTN dataset



(b) Population size effects on Computation time

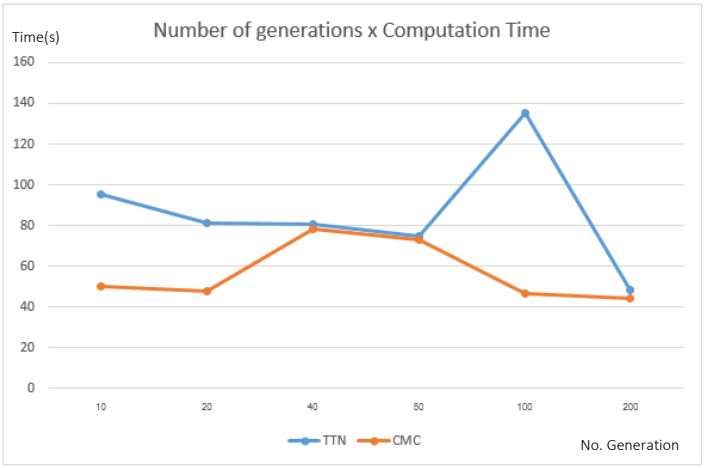
Figure 4.1 Population size effects on CMC and TTN dataset (Cont'd)

2) No of generations: We varied the number of generations to evaluate the performance of the proposed GA. The model performance result and computation time are shown in Figure 4.2a and Figure 4.2b, respectively. From Figure 4.2a, the number of generations affected the AUC performance for a bigger dataset more than a smaller one while it did not relate to computation time. Even though we had stopping criteria to break the iteration before it took a long period, a higher number of generations will lead to a high mutation rate due to our implementation of dynamic mutation. Consequently, the trend of AUC was decreasing after the optimal point. Fig 4.2b also displays that the cutoff point between performance and computation time was clear in the TTN dataset but not for the CMC dataset.



(a) Number of generations effect on AUC performance

Figure 4.2 Number of generation's effect on CMC and TTN dataset

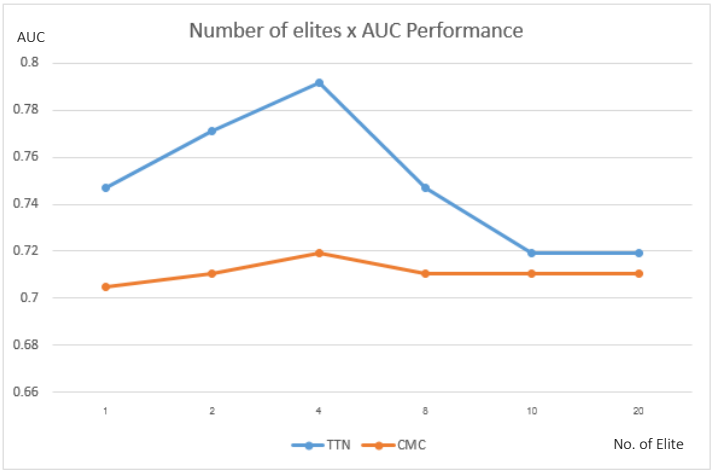


(b) Number of generations effect on computation time

Figure 4.2 Number of generation's effect on CMC and TTN dataset (Cont'd)

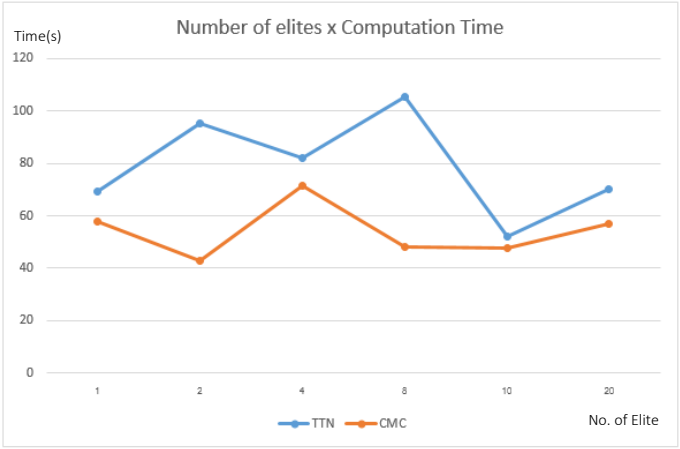
3) No of elites: This measured the effect of the number of elites to the proposed method. The model performance result is shown in Figure 4.3a and computation time is shown in Figure 4.3b. From Figure 4.3a, we could see that the optimal number of elites was 4 given the population size of 20. The more elites were preserved, the less the new population would be generated. Also, the performance also dropped when the number of elites was too low because we would lose the good candidate in every generation.

   The computation time in Figure 4.3b was not stable because the fluctuation in AUC performance came from the contribution of different numbers of generations to evaluate.



(a) Number of elites effects on AUC performance

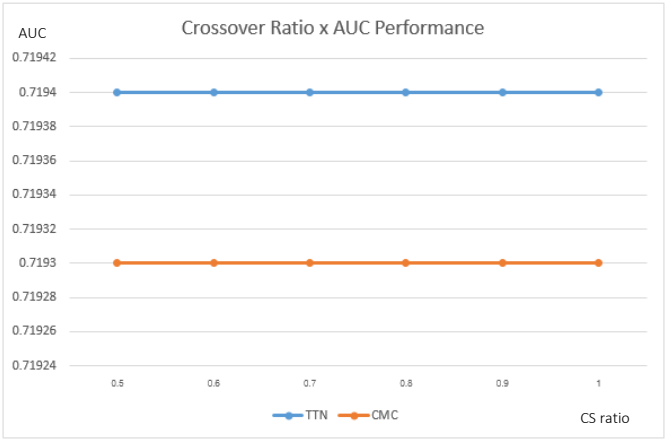
Figure 4.3 Number of elites effects on CMC and TTN dataset



(b) Number of elites effects on Computation time

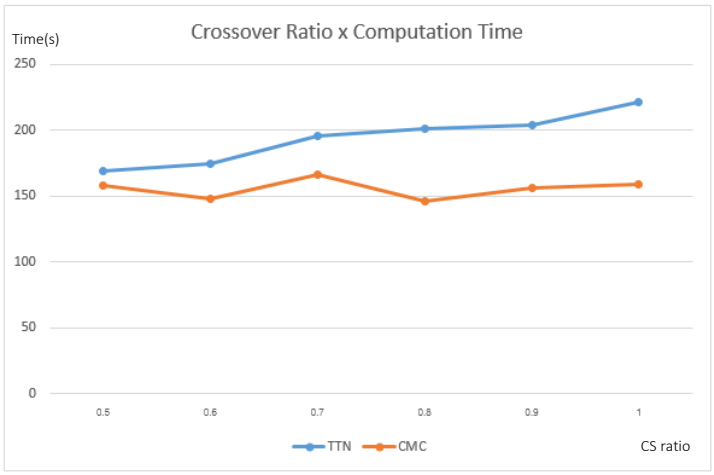
Figure 4.3 Number of elites effects on CMC and TTN dataset (Cont'd)

4) Crossover Ratio: This is the threshold used to select the number of individuals left in the reproduction pool to create a new population. The model performance result and computation time are shown in Figure 4.4a and Figure 4.4b, respectively. According to Figure 4.4a, the crossover ratio had no direct impact on the AUC performance of the final evaluation score. However, the fitness score from each generation was not stable because the crossover ratio controlled the migration rate of our framework. The low crossover ratio could translate to a high migration rate which would import more randomness into the reproduction pool. Fig 4.4b shows the effect of crossover ratio to the computation time in non-consistent in all dataset. In the TTN dataset, the higher crossover ratio took slightly more time while it did not have a direct effect on the computation time.



(a) Crossover ratio effects on AUC performance

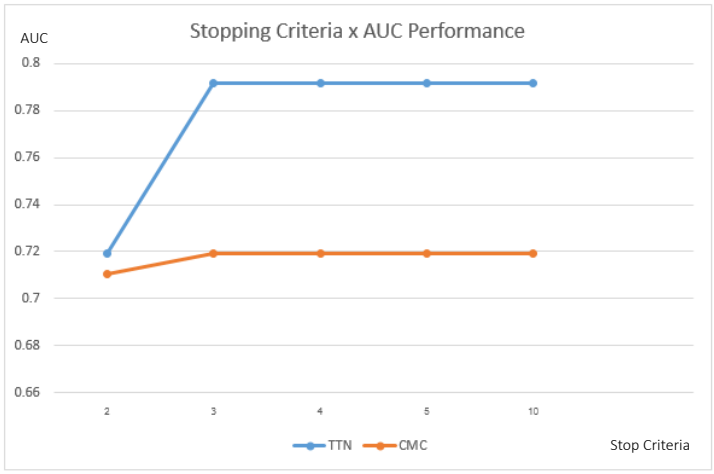
Figure 4.4 Crossover ratio effects on CMC and TTN dataset



(b) Crossover ratio effects on Computation time

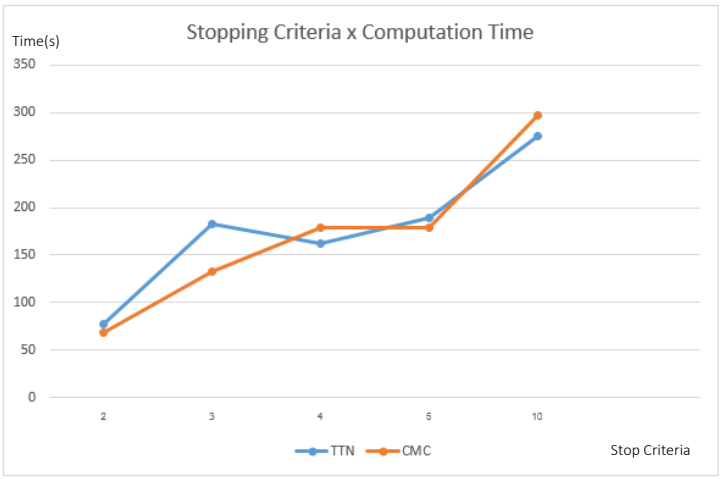
Figure 4.4 Crossover ratio effects on CMC and TTN dataset (Cont'd)

5) Stopping Criteria: This parameter represents the number of consecutive iteration without fitness score improvement before termination. The model performance result and computation time are shown in Figure 4.5a and Figure 4.5b, respectively. From Figure 4.5a, we can see that the AUC performance was still in the increasing trend but the process would stop prematurely if we set the stopping criteria too low. However, the increasing number of stopping criteria higher than three did not apply to the increase of AUC performance. Fig 4.5b also displays that computation time tended to increase along with the number of stopping criteria. A higher bound of stopping criteria would increase the number of wasted trials after the optimal point but setting stopping criteria too low would create the situation that our configuration stopped before it reached the best outcome.



(a) Stopping criteria effects on AUC performance

Figure 4.5 Stopping criteria effects on CMC and TTN dataset



(b) Stopping criteria effects on Computation time

Figure 4.5 Stopping criteria effects on CMC and TTN dataset (Cont'd)

The best practice was to trade between a certain number of the elites and the new population from the reproduction pool. We selected the optimal parameter setting from the point that had the greatest AUC score while maintaining a reasonable computation time. The result from our experiment also displayed the effect of parameter setting. Our recommended default setting could be the combination of peak AUC score over five parameters, which are a population size of 20, 40 generations, 4 elites selected per generation and stopping criteria at 3 generations. The crossover ratio would be recommended at 0.9 as the minimum ratio for migration rate.

The results from TTN and CMC datasets yielded the baseline for the default parameters set for the hyper-genetic algorithm. However, a different set of data might require a more specific tuning to bring the best out of it. Tuning all of the five parameters every time might demand more computation cost than the benefits it can produce so we conducted similar experiments on the SETAP and DOTA in Figure 4.6 and Figure 4.7 to create a guideline for hyper-genetic parameter setting.

Figure 4.6 Population size effects on SETAP and DOTA dataset

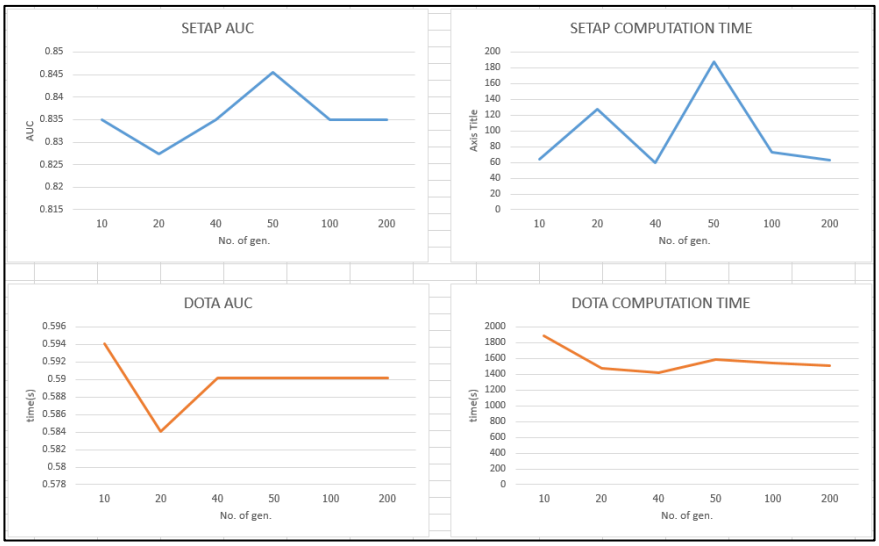


Figure 4.7 Number of generation effects on SETAP and DOTA dataset

The result from the experiment showed that we can narrow down the parameters to configure from 5 parameters to 2 parameters. The two parameters which had a direct impact on the performance were population size and the number of generations. Thus, we recommend using the default parameter set that we provided or tuning the two parameters by comparing the analysis result from each setting with the default value pair of another parameter. This would reduce the complexity of computation from the power of five parameters to two sets of parameters. Using the full five parameters tuning could also lead to the overfitted setting.

The parameter population size directly impacts the computation time. Figure 4.1 and figure 4.6 have shown that the population size would increase the computation time steeply after 50 while the slight increase was sometimes not worth the cost. Therefore, we recommend tuning the population size from 10 to 50 at most while leaving the number of elites, crossover ratio, and stopping criteria at default values. Table 4.2 shows the result from hyper-genetic parameter tuning for each dataset and its total computation time.

Table 4.2 GA Parameter best setting

Datasets	Number of population	Number of generations	AUC	Tuning time	Training time	Total time
TTN	20	40	0.7916	734.419	86.7142	821.1332
CMC	20	40	0.7193	649.7524	69.9702	719.7226
SETAP	50	50	0.8614	510.1336	253.1761	763.3097
DOTA	50	10	0.5921	16006.95	3979.318	19986.27

4.2 Results Comparison and Discussion of Default Parameters

In this section, we measured the performance of the proposed algorithm with the other optimization methods. In order to measure the performance of the proposed algorithm, we compared the result of the hyper-parameter tuning algorithm with other methods. We implemented hyper-parameter optimization with sklearn Grid Search with Cross-

Validation, Bayesian Optimization with hyperopt, and one set of hyper-parameter from random settings as a baseline model. Grid-search is a traditional exhaustive searching method that sweeps over all the parameters manually provided in the parameter grid while evaluating every parameter combination with cross-validation for the best performer. Bayesian optimization is different from Grid-search as it does not sweep through all possible parameters but using the information from the previous iteration to help determine the direction of optimal value.

Our proposed model had the best-practice GA default parameter setting as follows:

- population size = 20
- number of generation = 40
- number of elites = 4
- crossover ratio = 0.9
- stopping criteria = 3

We ran each of the hyper-parameter optimization methods and selected the best candidate GBT from each method to compare in terms of AUC performance and computation time. All optimization methods were performed in the same environment using CPU Intel i5 3.3GHz 4 cores and RAM 16GB.

However, our proposed algorithm had a set of possible GA parameters configurable. Thus, the experiment was conducted under the default parameter sets and the optimized parameter sets with tuning time included in the calculation. The other two algorithms that were used to compare did not have the specific parameter for their libraries to be tuned. We defined their parameter search space to be in the same configuration as our algorithm for Bayesian optimization while Grid-search optimization was executed under the defined interval under the same range as provided below.

A parameter defined range for GBT algorithm:

- learning_rate from 0.01 to 1
- n_estimator from 10 to 1000
- max_dept from 1 to 15
- min_samples_split from 0.01 to 1
- min_sample_leaf from 0.01 to 0.5
- subsample from 0.7 to 1

4.2.1 Comparison of predictive capability

Table 4.3 shows the predictive performance of the four algorithms. From Table 4.3, all of the candidates except Random Setting performed quite well as the AUC scores were very similar to each other. Hyper-genetic performed slightly better than the rest at the TTN dataset which had the most dimension of attributes. The fact that the best GA parameter setting for the TTN and CMC was similar to the default setting we selected in this experiment also contributed to the performance of these two outperforming the other datasets. The other two datasets CMC and DOTA had a very negligible difference in performance, while the top three algorithms performing quite similar with less than one percent different from each other. On the other hand, the random setting could represent the effect of the non-tuned model on the same environment which was outperformed by the other algorithms with high margins.

Table 4.3 Prediction Performance Comparison

Algorithm	AUC Performance on dataset			
	TTN	**CMC**	**SETAP**	**DOTA**
Hyper-genetic	**0.7916**	**0.7193**	**0.8349**	**0.5902**
Grid-search	0.7620	0.7091	0.8503	0.5906
Bayesian Optimization	0.7416	0.7136	0.8430	0.5889
Random Setting	0.6727	0.6210	0.6767	0.5

4.2.2 Comparison of computation time efficiency

Table 4.4 Computation Time Comparison on the Four Datasets

Algorithm	Computation time on dataset (sec)			
	TTN	**CMC**	**SETAP**	**DOTA**
Hyper-genetic	**86.7142**	**69.9702**	**56.0711**	**1544.4073**
Grid-search	1097.6906	1127.2980	1403.2798	42931.6769
Bayesian Optimization	74.9826	79.5372	130.2907	2231.2927

We also compared the performance in terms of computation time among three algorithms which are our proposed algorithm, Grid-search, and Bayesian Optimization in Table 4.4. Note that a random setting consumed negligible time. The Grid-search method took the highest computation time as expected. Our proposed model consumed the least computation time in three of the four datasets since our fitness function took the computation time into account. Bayesian optimization could achieve the least computation time only in the dataset TTN. This could result from too many generations spent on training.

4.3 Results Comparison and Discussion of Tuned Parameter

In this section, we evaluated the performance of our proposed algorithm with the GA parameter sets optimized specifically for each dataset according to our recommendation guideline in Section 4.1. The result from our optimized hyper-genetic algorithm included the GA parameter tuning time before comparing it to the other hyper-parameter optimization methods. We compared the result in term of AUC performance and computation cost as we had done in the experiment for default parameters.

4.3.1 Comparison of predictive capability on optimized parameter

In Table 4.5 we compared the AUC performance of hyper-genetic algorithm with the GA parameter optimized for each dataset compared to Grid-search and Bayesian optimization methods.

Table 4.5 Prediction Performance Comparison of tuned setting

Algorithm	AUC Performance on dataset			
	TTN	**CMC**	**SETAP**	**DOTA**
Tuned Hyper-genetic	**0.7916**	**0.7193**	**0.8614**	**0.5921**
Grid-search	0.7620	0.7091	0.8503	0.5906
Bayesian Optimization	0.7416	0.7136	0.8430	0.5889
Random Setting	0.6727	0.6210	0.6767	0.5

The results from Table 4.5 show that under the optimized setup for hyper-genetic algorithm, it could perform slightly better in terms of AUC performance in the CMC and DOTA datasets compared to the previous experiment. This table shows that our proposed algorithm with the GA parameter tuned for every dataset had a better AUC performance with the GA parameter tuned specifically for every dataset.

4.3.2 Comparison of computation time efficiency on tuned parameter

Table 4.6 Computation Time Comparison of Four Candidate with tuned setting

Algorithm	Computation time on dataset (sec)			
	TTN	**CMC**	**SETAP**	**DOTA**
Tuned Hyper-genetic	**821.1332**	**719.7226**	**763.3097**	**19986.2665**
Grid-search	1097.6906	1127.2980	1403.2798	42931.6769
Bayesian Optimization	74.9826	79.5372	130.2907	2231.2927

This experiment displayed that tuned GA parameter setting for hyper-genetic algorithms can take much longer computation time, compared to the previous experiment. This was obviously due to the tuning time as an additional step to the processes. Table 4.6 shows that our proposed algorithm has taken more computation time than Bayesian optimization while performing better than Grid-search bot not a large margin in most cases. Although the tuned hyper-genetic have a better performance, it might not be worth the increase in computation cost in most cases depending on the characteristics of datasets. Consequently, the proposed hyper-genetic algorithm with the GA default parameter setting is recommended for automated hyper parameter tunning in GBT.

These experiments also show that our algorithm has the best performance in the TTN dataset which has a high number of attribute dimensions and a wide range of AUC performance between each hyper-parameter setting. The range of varied performance provides the information for our fitness function to evaluate and learn the improvement. The low range of AUC performance such as the DOTA dataset also generates a greater

local minimum. This will result in the difference between our proposed algorithm and other algorithms becoming nonobvious.

CHAPTER 5 CONCLUSION

In this thesis, we have proposed a Genetic algorithm (GA) on the hyper-parameter optimization (called, Hyper-genetic) for Gradient Boosting Classifier (GBC). Our approach was implemented as a non-supervised hyper-parameter tuning tool, which is robust and efficient. The Hyper-genetic defined the hyper-parameter set of GBC as genetic representation and then reproduced new parameters with more reliable performance as a better generation. We used the genetic property of crossover, dynamic mutation, and elitism with the modification of fitness function and migration processes. The mutation and migration could overcome the local optimum and converge at the best parameters set. In the evaluation, we have compared the performance of our proposed algorithm with Grid-search, Bayesian Optimization, and random setting in four datasets. The results revealed that our proposed algorithm achieves a competitive prediction performance within the least computation time. Moreover, it could solve the problem of background knowledge required to tune hyper-parameters of GBC.

The result from the experiment shown that the parameters that have the strongest impact on our hyper-parameter optimization frameworks are population size and number of generations. The first parameter is directly related to the computation time required in each generation while the second parameter has a relationship with the dynamic mutation rete. The number of elite has a significant effect on one dataset and a non-obvious impact on the other, so it is not the most critical parameter if we keep it in the medium threshold.

Comparison analysis has shown that our proposed framework with default setting has the best AUC performance results in two from the four datasets while keeping the competitive computation time with the others. The proposed frameworks also produce a close to the top of the performance table in the biggest dataset but require much less time compared to the best performer model. We can conclude that our hyper-genetic framework is the best candidate to extract the best performance of the chosen model within a specific amount of time for most datasets compared to the other optimization methods.

The performance of our proposed model increases in most datasets with the tuned GA parameter and is even better than comparing optimization methods. However, the computation cost also significantly increases compared to the hyper-genetic with default parameter sets. Thus, we encourage the usage of default parameter sets unless the predictive performance is the top priority.

Future research should focus on expanding the range of predictive model from classification to regression. There is more room for research on a comparison between a wider range of datasets with different characteristics or implementing on different platforms. In addition, future research can be done on reducing the parameter setting tuning time of proposed algorithm. Scaling up the experiment to big data platform with spark library to evaluate the performance of the trained model of enormous data can also highlight the differences in performance between each optimization method.

REFERENCES

1. Probst, P., Bischl, B, and Boulesteix, A., 2018, Tunability: **Importance of Hyperparameters of Machine Learning Algorithms**, [Online], Available: https://arxiv.org/abs/1802.09596v3 [2020, June 27].

2. Snoek, J. and Larochelle, H. and Ryan, P.A., 2012, "Practical Bayesian Optimization of Machine Learning Algorithms", In **Advances in Neural Information Processing Systems**, Curran Associates, Inc, pp. 2951-2959.

3. Bergstra, J., Pinto, N. and Cox, D., "Machine Learning for Predictive Auto-tuning with Boosted Regression Trees", **2012 Innovative Parallel Computing (InPar)**, 13-14 May 2012, San Jose, CA, pp. 1-9, doi: 10.1109/InPar.2012.6339587.

4. Yufei, X., Chuanzhe, L., YuYing, L. and Nana, L., 2017 , "A Boosted Decision Tree Approach Using Bayesian Hyper-parameter Optimization for Credit Scoring", **Expert Systems with Applications**, Volume 78, pp. 225-241.

5. Thomas, J. and Coors, S. and Bischl, B., **Automatic Gradient Boosting**, [Online] Available : https://arxiv.org/abs/1807.03873 [2020, June 30].

6. Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D., 2016, "Mastering the Game of Go with Deep Neural Networks and Tree Search". **Nature**, Vol. 529, pp. 484–489.

7. Holzinger, A., Kieseberg, P., Weippl, E. and Tjoa, A.M., 2018, "Current Advances, Trends and Challenges of Machine Learning and Knowledge Extraction: From Machine Learning to Explainable AI", **Machine Learning and Knowledge Extraction**. CD-MAKE. Lecture Notes in Computer Science, Vol. 11015. Springer, Cham, pp. 295–303.

8. Jain, A., 2016, **Complete Machine Learning Guide to Parameter Tuning in Gradient Boosting (GBM) in Python**, [Online] Available :https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/ [2020, June 30].

9. Sethneha, 2020, **Entropy – A Key Concept for All Data Science Beginners**, [Online], Available: https://www.analyticsvidhya.com/blog/2020/11/entropy-a-key-concept-for-all-data-science-beginners/ [2021, April, 26]**.**

10. Bergsta, J., Bengio, Y., 2012, "Random Search for Hyper-Parameter Optimization", **Journal of Machine Learning Research 13**, pp. 281-305.

11. Kumar, P., Bai, M.A. and G.Nair, G., 2021, "An Efficient Classification Framework for Breast Cancer Using Hyper Parameter Tuned Random Decision Forest Classifier and Bayesian Optimization", **Biomedical Signal Processing and Control**, Vol. 68, No. 102682, Elsevier Ltd.

12. Nayak, J., Naik, B., Dash, P. B., Souri, A. and Shanmuganathan, V., 2021, "Hyper-parameter Tuned Light Gradient Boosting Machine Using Memetic Firefly Algorithm for Hand Gesture Recognition", **Applied Soft Computing**, Vol.107, No. 107478, Elsevier Ltd.

13. Aparra, 2016, **Applying Genetic Algorithms to Define a Trading System**, [Online], Available: https://quantdare.com/ga-to-define-a-trading-system/, [2021, April 26]**.**

14. Guo, P., Wang, X. and Han, Y., 2010, "The Enhanced Genetic Algorithms for the Optimization Design**", 2010 3rd International Conference on Biomedical Engineering and Informatics**, 16-18 October 2010, China, pp. 2990-2994.

15. Miranda, R., Montevechi, J.A. and De Pinho, A. F., 2015, "Development of an Adaptive Genetic Algorithm for Simulation Optimization", **Acta Scientiarum Technology**, Vol. 37, No. 3, pp. 321-328.

16. Fridrich, M., 2017, "Hyperparameter Optimization of Artificial Neural Network in Customer Churn Prediction Using Genetic Algorithm", **Trends Economics and Management**, Vol. 11, pp. 9.

17. Lin, C., 2009, "An Adaptive Genetic Algorithm Based on Population Diversity Strategy", **Third International Conference on Genetic and Evolutionary Computing,** 14-17 October 2009, China, pp. 93-96.

18. Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A. and Prasath, V.B.S. , 2019, "Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach", **Information**, Vol. 10, No. 12, pp. 390.

19. Kraus, M., 2019, **Using Bayesian Optimization to Reduce the Time Spent on Hyperparameter Tuning**, [Online], Available: https://github.com/MBKraus/Hyperopt [2020, July 2].

CURRICULUM VITAE

NAME	Mr. Kankawee Kiatkarun
DATE OF BIRTH	19 February 1994
EDUCATIONAL RECORD	
HIGH SCHOOL	High School Graduation Suankularb Vittayalai School, 2011
BACHELOR'S DEGREE	Bachelor of Engineering (Computer Engineering) King Mongkut's University of Technology Thonburi, 2015
MASTER'S DEGREE	Master of Engineering (Computer Engineering) King Mongkut's University of Technology Thonburi, 2020
PUBLICATION	Kiatkarun, K. and Phunchongharn, P., 2020, "Automatic Hyper-Parameter Tuning for Gradient Boosting Machine", **2020 1st International Conference on Big Data Analytics and Practices (IBDAP)**, 25-26 September 2020, Bangkok, Thailand, pp. 1-6.