Journal of Electronics, Electromedical Engineering, and Medical Informatics
Multidisciplinary: Rapid Review: Open Access Journal **Vol. 6, No. 2, April 2024, pp: 137-147; eISSN: 2656-8632**

RESEARCH ARTICLE **OPEN ACCESS**

Manuscript received February 10, 2024; revised February 26, 2024; accepted March 8, 2024; date of publication April 20, 2024
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v6i2.375
Copyright © 2024 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).
How to cite: Mulia Kevin Suryadi, Rudy Herteno, Setyo Wahyu Saputro, Mohammad Reza Faisal, and Radityo Adi Nugroho, A Comparative Study of Various Hyperparameter Tuning on Random Forest Classification with SMOTE and Feature Selection Using Genetic Algorithm in Software Defect Prediction, Journal of Electronics, Electromedical Engineering, and Medical Informatics, vol. 6, no. 2, pp. 137-147, April 2024

A Comparative Study of Various Hyperparameter Tuning on Random Forest Classification with SMOTE and Feature Selection Using Genetic Algorithm in Software Defect Prediction

Mulia Kevin Suryadi, Rudy Herteno, Setyo Wahyu Saputro, Mohammad Reza Faisal, and Radityo Adi Nugroho

Computer Science Department, Lambung Mangkurat University, Banjarbaru, South Kalimantan, Indonesia
Corresponding author: rudy.herteno@ulm.ac.id

ABSTRACT Software defect prediction is necessary for desktop and mobile applications. Random Forest defect prediction performance can be significantly increased with the parameter optimization process compared to the default parameter. However, the parameter tuning step is commonly neglected. Random Forest has numerous parameters that can be tuned, as a result manually adjusting parameters would diminish the efficiency of Random Forest, yield suboptimal results and it will take a lot of time. This research aims to improve the performance of Random Forest classification by using SMOTE to balance the data, Genetic Algorithm as selection feature, and using hyperparameter tuning to optimize the performance. Apart from that, it is also to find out which hyperparameter tuning method produces the best improvement on the Random Forest classification method. The dataset used in this study is NASA MDP which included 13 datasets. The method used contains SMOTE to handle imbalance data, Genetic Algorithm feature selection, Random Forest classification, and hyperparameter tuning methods including Grid Search, Random Search, Optuna, Bayesian (with Hyperopt), Hyperband, TPE and Nevergrad. The results of this research were carried out by evaluating performance using accuracy and AUC values. In terms of accuracy improvement, the three best methods are Nevergrad, TPE, and Hyperband. In terms of AUC improvement, the three best methods are Hyperband, Optuna, and Random Search. Nevergrad on average improves accuracy by about 3.9% and Hyperband on average improves AUC by about 3.51%. This study indicates that the use of hyperparameter tuning improves Random Forest performance and among all the hyperparameter tuning methods used, Hyperband has the best hyperparameter tuning performance with the highest average increase in both accuracy and AUC. The implication of this research is to increase the use of hyperparameter tuning in software defect prediction and improve software defect prediction performance.

INDEX TERMS Genetic Algorithm, Hyperparameter Tuning, Random Forest, Software Defect Prediction

I. INTRODUCTION

Software system continue to develop and have an important role in every aspect of our society [1]. With this important role, the level of software complexity will increase and will also increase the difficulty in providing high quality, low-cost, and maintainable software. This difficulty will also increase the possibility of creating software defects [2]. A defect is an abnormality in software that causes the system to run incorrectly or produce unexpected results [3]. Software defects can cause a failure in the system which will reduce the quality of desktop or mobile applications. These defects may occur due to syntax failures, spelling errors, incorrect program code in lines, requirements, and designs or specifications [4]. Defect prediction is one of the pivotal and crucial tasks in the software development process. Defect prediction can reduce maintenance costs, improve quality, performance, and improve user satisfaction [3]. The impact of these defect predictions needs to be considered with the rise of software development, especially the trend of using mobile applications. There is a lack of a vast overview of the present state defect prediction research due to the large number of published divergent software defect prediction datasets, approaches, and frameworks [5].

Tuning was not discussed in 78% of the research work [6]. If the parameters are adjusted, the learner algorithm efficiency and performance score increase significantly contrasted to the non-tuned values for fixed code attributes, which usually lead to damaging and illusive outcomes. This ensures that the outcome will be pretty much optimized by searching the entire problem search. The parameter optimization process can significantly increase Random Forest defect prediction performance compared to the default parameter [7]. However, the parameter tuning step is commonly neglected. Random Forest has numerous parameters that can be tuned. As a result, manually adjusting parameters would diminish the efficiency of Random Forest, yield suboptimal results and it will take a lot of time. To address this issue, the hyperparameter tuning method was used to find the best parameter values automatically.

In research [8] the research using Artificial Neural Network (ANN) with Artificial Bee Colony (ABC) and generated AUC about 0.77 on CM1, 082 on P1, and 0.71 on JM1. Different research [9] used CS-ILDM, a hybrid of Cost-Sensitive Learning (CSL) and Large Margin Distribution Machine (LDM) generated AUC of about 0.771 on CM1, 0.856 on PC1, and 0747 on JM1. In other research [10] using Random Forest (RF) generated accuracy of about 0.929 on PC1, 0.983 on PC2, 0.892 on PC3, 0.882 on PC4, and using Improved Random Forest (IRF) generated accuracy of about 0.945 on PC1, 0.985 on PC2, 0.896 on PC3, and 0.906 on PC4.

In this study, authors made a comparison of various Hyperparameter Tuning for Software Defect Prediction that combines SMOTE to handle data imbalance problems, Genetic Algorithm (GA) as feature selection method, and the classification process will be applied using Random Forest (RF) algorithm. The Hyperparameter Tuning methods used are Grid Search, Random Search, Optuna, Bayesian Search (With Hyperopt), Hyperband, Tree Parzen Estimator, and Nevergrad. The approaches to each Hyperparameter Tuning method will be compared based on accuracy and AUC values.

This research aims to increase the accuracy and AUC of software defect prediction by combining all of those procedures. The results of this research are expected to provide contributions such as :

a. It provides a better understanding of feature selection on software defect prediction and classification performance with hyperparameter tuning.
b. This provides insight into the most efficient and optimal strategies for hyperparameter tuning.
c. It has the potential to be implemented in software defect prediction in order to get the more specific and optimal result.
d. The outcome of this study further enrich the awareness of the hyperparameter tuning procedure in software defect prediction.

II. MATERIAL AND METHODS

FIGURE 1 depicts the research flow for this study, which consists of SMOTE, feature selection, hyperparameter tuning process, and classification. In this study, the first step is to collect the NASA MDP dataset, followed by dividing the data onto data training and data testing. The dataset is split into 80% for data train and 20% for data test. Subsequently, feature selection is performed by employing Genetic Algorithm before hyperparameter tuning and classification. Then, the hyperparameter phase is executed using Grid Search, Random Search, Optuna, Bayesian with Hyperopt, Hyperband, Tree Parzen Estimators, and Nevergrad method. The classification phase used the Random Forest method. The study evaluation is based on the Accuracy and AUC value.

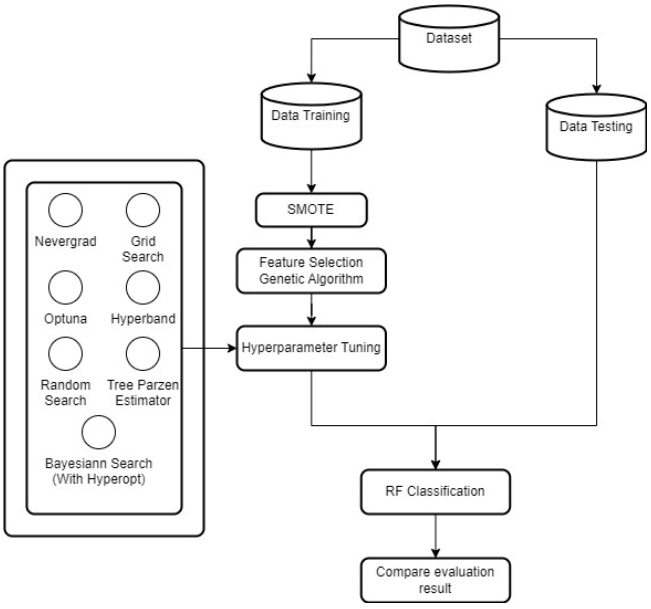


FIGURE 1. Research flow of Random Forest classification model

The search space was taken into consideration while selecting the hyperparameter tuning approach, and some of the parameters were chosen because of their similar process. The search space that was employed for this study was predefined. Other than numbers, search space values can be employed in the hyperparameter tuning approach. For example, bootstrap contains parameter values that can be either True or False.

In our study, hyperparameter tuning optimizes the parameter of Random Forest which is "n_estimators", "max_depth", "min_samples_leaf", "min_samples_split", and "bootstrap". The best parameter options (search space) are set as "[50, 100, 200, 400]" for n_estimators, "[None, 10, 20, 40]" for max_depth, "[2, 5, 10, 20]" for min_samples_split, "[1, 2, 4, 8]" for min_samples_leaf, and "[True, False]" for bootstrap. This parameter option is employed based on a larger search space than the default parameter. The default parameter values for the parameters are 100 for n_estimators, None for max_depth, 1 for min_samples_leaf, 2 for min_samples_leaf and True for bootstrap. Those search spaces are applied to all hyperparameter tuning in this study.

A. DATASET

The dataset used for this study is NASA MDP [11] that have been used in previous research. This dataset has two class labels:

1) Class label "Y" for defective
2) Class label "N" for no defective

Label "Y" is changed to "1" and label "N" changed to "0"

B. SMOTE

SMOTE generates fictional data based on the space characteristic similarities of minority modules. Compared to the standard oversampling method, it successfully avoids the classifier overfitting issue. The fundamental idea involves adding artificially created minority class instances to their closest neighbors, hence increasing the quantity of minority class instances to balance the dataset. [12]. Assume *N* represents the oversampling ratio. First of all, for each minority class instance, select *K* instances at random based on the *P* closest minority class neighbors for every minority class instance. Afterward, generate a synthetic sample for each instance in the minority class and select *K* instances to create *N* additional minority class samples. In the end, integrate the fresh instances with the existing instances set to create a new training instance set (Eq. (1)).

$$x_{new} = x + rand(0,1) \times (y[i] - x) \quad (1)$$

where *i* is 1, 2, … *N*, rand(0,1) are random numbers between 0 and 1. x_{new} is the new instance, *x* is the minority class instance, *y*[i] is the closest to *i* neighbour *x* [13].

C. GENETIC ALGORITHM FEATURE SELECTION

Genetic Algorithm is an optimization procedure that optimizes binary search spaces by manipulating potential solutions. The search area is deputized by a chromosome, comprising a limited series of "0" and "1". The Genetic Algorithm process is based on sample populations. Genetic Algorithm increases the amount of candidates seeking better solutions. Throughout the Genetic Algorithm process, the population encounters genetic operators like as selection, inheritance, and mutation. The GA approach begins by embarking a population hyperparameter sets, which represent possible remedies [14]. To utilize Genetic Algorithm as selection feature to exclude inconsequential or insignificant features, chromosomes are defined as a feature mask. A chromosome is represented as a binary string that is either "0" or "1". A value of "1"indicate the feature is selected, whereas "0" indicates it is not [15].

In studies carried out by [16]. In his research using Genetic Algorithm (GA) compared to other feature selection methods can select the best subset of features better. GA is also compared to Particle Swarm Optimization (PSO). According to the experiment, GA outperforms PSO in terms of performance. So, the feature selection used in this research is Genetic Algorithm.

D. GRID SEARCH HYPERPARAMETER TUNING

A Grid Search involved and constructed by a set of predetermined parameter values that are necessary to give ideal accuracy and AUC [17]. Grid Search integrates all the options that have been established by hyperparameters to get the ideal values for each parameters [18] (Eq. (2)).

$$\text{Parameter} = \arg max_{\theta \in G}\ f(\theta) \quad (2)$$

$\theta \in G$ means there is a consider on every combination of hyperparameter (θ) that exist in the grid set (G). $f(\theta)$ is the evaluation function that measures the performance of models with a particular set of hyperparameter. Grid Search consists of several steps [19], which are:

1) Automatically generates parameter sets depending on a given parameter option. If the specified parameter options are limited to 4 parameter values and applied to 5 parameters, then the grid search will generate $4 \times 4 \times 4 \times 4 \times 4 = 1024$ combination.
2) Analyze and evaluate all potential parameter settings.
3) Identify the best parameter.

E. RANDOM SEARCH HYPERPARAMETER TUNING

Random Search selects hyperparameter values at random from a specified hyperparameter space. This strategy may be more efficient for hyperparameter optimization, particularly when working with high-dimensional search spaces because it does not need to analyze all potential combinations. [20] [21]. The procedure of Random Search can be seen as follows (Eq. (3)).:

$$\text{Parameter} = \arg min_{\theta}\ Loss\ Function(\theta) \quad (3)$$

θ represents the hyperparameter vector to be optimized and $Loss\ Function(\theta)$ is a function that measures model performance based on a certain combination of hyperparameter. Those combinations are selected randomly. Random Search consists of several steps [19], which are:

1) Random Search creates potential parameter settings based on a specified iteration limit. If Random Search is used to seek the optimal parameters of 5 type parameters, the Random Search will generate parameter values according to the number of parameters given. The entire procedure goes on for a set number of iterations.
2) Analyze and evaluate all potential parameter settings.
3) Identify the best parameter.

F. OPTUNA HYPERPARAMETER TUNING

Optuna hyperparameter tuning approach involves minimizing or maximizing an objective function that acknowledges a set of hyperparameters as input and returns the validation score. Optuna considers optimization processes as studies, and objective function evaluations as trials [22]. Optuna tests numerous hyperparameter combinations and evaluates their performance on a defined testing dataset. Optuna uses iterative experimentation and evaluation to find the optimal hyperparameter set for a specific performance metric [23]. Optuna can be represented as follows (Eq. (4)) :

$$x_i = Optuna(f, S_{h_1}, S_{h_2}, \dots, S_{h_n}) \quad (4)$$

In Eq. (4) x_i is the number of iterations to be carried out, f is the objective function that must be optimized, and S_{h_n} is the

search space for each hyperparameter h_i. Optuna includes a pruning tool that allows you to prematurely end runs that are not optimum. To do this, the intermediate goal values are tracked and those that do not fulfill established parameters are eliminated. Optuna optimization approach is not confined to a single machine learning library or framework, making it a versatile tool that can be utilized across many domains and with different types of machine learning models [24].

G. BAYESIAN SEARCH (WITH HYPEROPT) HYPERPARAMETER TUNING

Bayesian Optimization when picking the optimal hyperparameter set for the next assessment, consider the previous evaluation to determine the ideal hyperparameter. It involves updating the posterior distribution and maximizing the acquisition function. [25]. Bayes Search works by allocating a precedence likelihood to a particular parameter. Subsequently, multiplying it by the odds dispersion of the grading function to determine the likelihood of discovering better outcomes given a collection of hyperparameters [26]. The Bayesian optimization procedure works as follows:

1) Use Eq. (5) to determine the points for each acquisition function.

$$x_t^i = \arg Max_x u_i\,(x|D_{1:t-1}) \quad (5)$$

2) Choose the nominee using the probability Eq. (6)

$$P_t(j) = e^{ng_{t-1}^j} / \Sigma_{1=1}^k e^{ng_{t-1}^j} \quad (6)$$

3) Obtain a sample of the goal function f using Eq. (7)

$$y_t = f(x_t) + \epsilon_t \quad (7)$$

4) Add The data to Eq. (8) and update the posterior function f.

$$D_{1:t} = \{D_{1:t-1}, (x_t, y_t) \quad (8)$$

5) Earn results with the Eq. (9).

$$r_t^i = \mu_t(x_t^i) \quad (9)$$

6) Update gains with the Eq. (10)

$$g_t^i = g_{t-1}^i + r_t^i \quad (10)$$

where $D_{1:t} = \{x_n, y_n\}_{n=1}^{t-1}$ represents a training dataset consisting of $t-1$ observations of function f. The posterior f is calculated utilizing the Gaussian procedure given by Eq. (11), which assumes that the function mean *m(x)* = 0, the variance function *k* is specified by Eq. (12), and x_i and x_j represent the *i*th and *j*th samples, respectively [25].

$$f(x) \sim GP(m(x), k(x_i, x_j)) \quad (11)$$

$$k(x_i, x_i) = \exp\left(-\frac{1}{2}||x_i - x_j||^2\right) \quad (12)$$

Hyperopt offers an optimization framework that disperses a configuration space, as well as an evaluation function that maps points within this space to actual-valued loss values, thereby enabling optimization techniques for exploring search spaces. These spaces have diverse variable types, sensitivity profiles, and conditional structures [27][28]. Hyperopt employs a Bayesian Optimization-based approach to explore a wide hyperparameter space more efficiently. This implies that Hyperopt tries to forecast the model performance based on prior evaluations and utilizes those predictions to pick the next hyperparameters to test, with the objective of finding the optimum combination with the smallest number of evaluations. This differs from procedures like Grid Search and Random Search, which may be less efficient since they do not leverage knowledge from past assessments [29].

H. HYPERBAND HYPERPARAMETER TUNING

Hyperband is a hyperparameter optimization approach that uses a bandit strategy to distribute resources repeatedly to a series of random hyperparameter configurations [30]. Hyperband creates a collection of *n* trial points and each trial point represents one hyperparameter setting. After that, Hyperband allocates resources data to each test point and assesses its performance. That means each tested hyperparameter configuration receives the same amount of resources to demonstrate its potential performance. By setting and limiting the number of resources used for each experiment, Hyperband tries to reduce the time and resources required to find the optimal hyperparameter configuration. A percentage of trial points with poor performance are regarded as less promising and consequently deleted from the set. This technique is repeated multiple times until there is just one trial point remaining in the set [31].

I. TREE PARZEN ESTIMATOR HYPERPARAMETER TUNING

Tree Parzen Estimator (TPE) is an optimization technique that uses the search area and trial record hyperparameters as input and recommends which values to attempt in the next steps [32]. At every attempt, TPE picks fresh parameter samples and determines which set to use in the following iteration. At first, samples are selected equally over the search region and assessed. The gathered samples are sorted into two categories according to their score. The first category comprises samples that enhance the present score approximation, while the second category comprises the remainder. The goal of TPE is to identify which parameters are most probable to fall into the first category. Based on this process, TPE utilizes the distribution of the most optimal samples rather than the best estimated parameters[33].

J. NEVERGRAD HYPERPARAMETER TUNING

Nevergrad is a derivative-free optimization platform that gathers a vast range of optimization methods and a vast range of test functions to assess them. Nevergrad may simply create and define a search domain, allowing numerous algorithms in Nevergrad to automatically change variables and take consideration of their perhaps logarithmic or discrete nature, as well as any user-defined mutation or recombination operator [34] [35].

K. RANDOM FOREST CLASSIFIER

Random Forest is a classification approach that integrates numerous decision trees. Each decision tree is constructed

with random and independent sampled vector values. These vectors are spread identically across all trees in the model. By combining predictions from these trees, Random Forest can reduce overfitting and improve model performance. Random Forest picks features at random from the whole number of features. After that, the root node is identified using the most efficient split technique. Then, the children of the nodes are going to be extracted using the identical best split method. Those steps will be carried out until a tree is constructed with the root node and the goal is obtained as a leaf node. In the end, all previous steps are repeated in order to generate a random number of trees A significant advantage of RF is that there is no need to trim each tree when there are several trees [36] [37]. This approach is based on two fundamental principles which are randomly selecting a subset of rows from a dataset and fuse the predictions of multiple classifiers. The data are resampled and supplied to the following basic learner algorithms for training [38]. Random Forest has many configurable parameters. Configuration of Random Forest parameters has a big impact on performance, so the hyperparameter tuning process is suitable for Random Forest [39].

III. RESULTS

This section shows the performance of each hyperparameter tuning method utilizing Random Forest as the classification method. The performance of Random Forest is assessed based on the Accuracy and AUC values obtained.

A. SMOTE PROCESS

The SMOTE process is executed on the data train. SMOTE increases the *x* and *y* train on all datasets, from 261 to 460 on CM1, 6225 to 9762 on JM1, 159 to 240 on KC1, 155 to 256 on KC3, no change on KC4, 1590 to 3108 on MC1, 100 to 120 on MC2, 202 to 370 on MW1, 564 to 1030 on PC1, 596 to 1168 on PC2, 861 to 1524 on PC3, 1029 to 1768 on PC4 and 1368 to 2004 on PC5. Those processes use SMOTE with random state = 42.

B. GENETIC ALGORITHM PROCESS

The Genetic Algorithm procedure is carried out with default parameter settings, as was also done in the study [40]. Our Genetic Algorithm employs the "parameter_population_size" = 5, "num_generations" = 50 and "mutation_rate" = 0.05. TABLE 1 displays the outcome of GA feature selection.

TABLE 1
Feature selection with genetic algorithm

Dataset	Features	Feature Selection GA
CM1	37	17
JM1	21	14
KC1	21	11
KC3	39	18
KC4	41	22
MC1	38	20
MC2	39	20
MW1	37	18
PC1	37	22
PC2	36	19
PC3	37	16
PC4	37	23
PC5	38	17

C. HYPERPARAMETER TUNING PROCESS

TABLE 2 shows the optimal parameter obtained after going through the hyperparameter tuning process. TABLE 2 is a pivotal element of this study, detailing the optimal hyperparameters obtained through various tuning methods for different datasets. Each dataset, named from CM1 to PC5, represents a unique case within the NASA MDP dataset collection. The table compares the results of six tuning methods: Grid Search, Random Search, Optuna, Bayesian (with Hyperopt), Hyperband, TPE, and Nevergrad34. These methods are employed to find the best combination of hyperparameters that yield the highest accuracy and AUC values for defect prediction. This granular level of detail enables the researchers to draw meaningful conclusions about the efficacy of each hyperparameter tuning approach, ultimately guiding the selection of the most effective method for enhancing the Random Forest classifier's predictive capabilities. The study's findings, as encapsulated in Table 2, serve as a testament to the importance of hyperparameter tuning in machine learning tasks, particularly in the context of software defect prediction where precision is paramount.

D. PERFORMANCE OF RANDOM FOREST CLASSIFIER

The Random Forest model process is carried out using optimal parameters obtained from the previous process. TABLE 3 displays the Accuracy and AUC values when the Random Forest classification method uses optimal parameters as in TABLE 2. Based on TABLE 3, It can be seen that there is an improvement in Accuracy and AUC in Random Forest which uses optimal parameters compared to Random Forest without using optimal parameter.

In this study, the result of the software defect prediction assessment of NASA MDP datasets on the accuracy and AUC values obtained are presented in TABLE 3. Based on TABLE III there are several Accuracy and AUC values that are similar and even have the same value for several tuning hyperparameters. FIGURE 2 shows a comparison of each performance for each hyperparameter tuning. However, it can be seen in TABLE 3 that there are several performances on certain datasets that experience a decrease in Accuracy and AUC. TABLE 4 shows that of all datasets, Nevergrad has the biggest average gain in accuracy of roughly 3.9%, while Hyperband enhances AUC by approximately 3.5%. However, when utilizing accuracy and AUC as standards for this approach, Hyperband has the highest overall improvement rate.

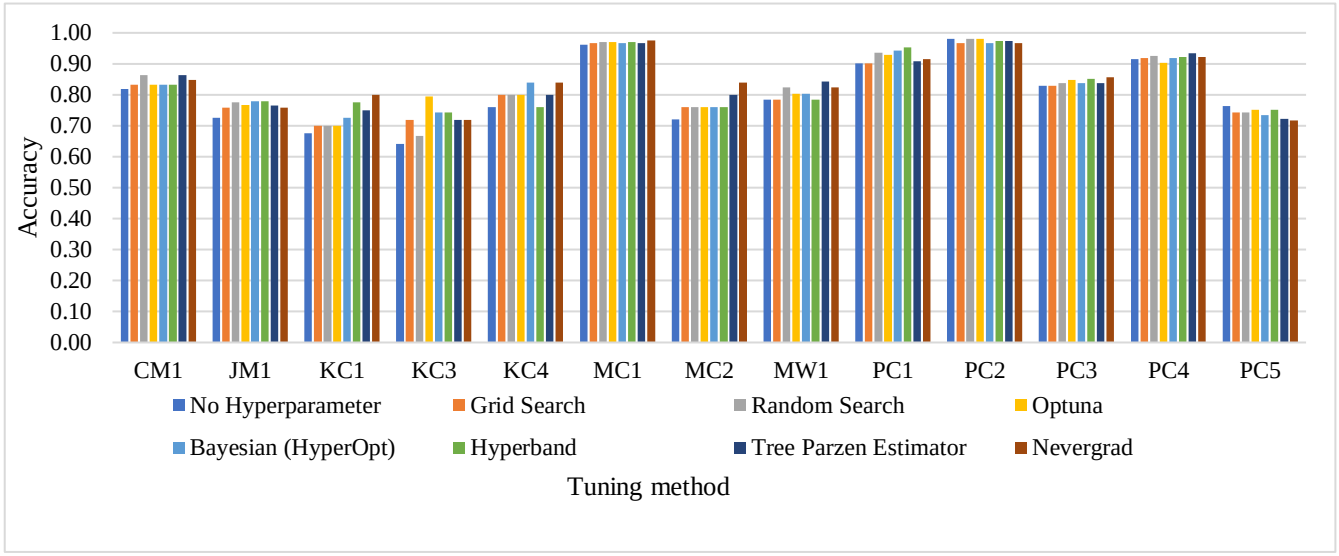
TABLE 2
Optimal parameter

Tuning	Parameter	Dataset												
		CM1	JM1	KC1	KC3	KC4	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Grid Search	n estimators	400	400	400	100	100	400	200	100	50	50	400	50	200
	min samples split	2	10	2	2	20	5	5	5	5	2	2	5	5
	min samples leaf	1	1	1	1	4	1	1	1	1	1	1	1	1
	max depth	40	None	40	10	10	None	10	None	40	40	None	20	None
	bootstrap	False	False	False	True	False	False	True	False	False	False	False	False	False
Random Search	n_estimators	100	200	100	100	50	50	100	50	200	50	200	200	200
	min_samples_split	10	5	2	2	20	5	10	5	2	5	2	5	5
	min_samples_leaf	4	2	1	2	2	1	4	1	2	1	1	2	1
	max_depth	20	40	40	None	10	40	10	10	None	20	None	None	None
	bootstrap	False	False	False	True	False	False	True	True	False	False	False	False	True
Optuna	n_estimators	200	50	400	200	100	50	400	50	400	100	50	200	400
	min_samples_split	20	5	10	5	5	2	2	2	2	10	10	2	2
	min_samples_leaf	1	4	1	1	2	1	1	1	1	1	4	1	2
	max_depth	None	40	10	10	20	40	None	40	None	40	20	40	20
	bootstrap	True	False	False	False	True	True	True	False	True	False	False	True	True
Bayesian (HyperOpt)	n_estimators	200	200	400	400	50	400	50	100	50	100	200	100	100
	min_samples_split	5	2	2	2	2	5	2	2	2	10	2	2	5
	min_samples_leaf	1	2	1	1	20	2	1	1	1	1	1	1	1
	max_depth	40	None	20	20	40	40	40	None	40	40	40	40	20
	bootstrap	False	False	True	True	True	False	True	True	False	False	False	False	False
Hyperband	n_estimators	400	100	200	400	200	200	50	50	50	200	200	400	200
	min_samples_split	2	2	2	2	10	2	5	5	2	2	2	2	5
	min_samples_leaf	1	1	1	4	8	1	2	1	2	2	1	2	1
	max_depth	10	None	40	10	40	40	None	None	20	40	40	20	20
	bootstrap	True	False	False	False	True	False	True	False	True	False	False	False	False
TPE	n_estimators	400	200	400	400	50	400	50	400	50	50	50	400	200
	min_samples_split	10	2	10	10	10	5	5	10	2	20	10	2	10
	min_samples_leaf	1	4	1	1	8	2	2	4	2	2	1	1	1
	max_depth	20	None	40	40	None	20	20	20	40	None	None	40	40
	Bootstrap	False	False	False	False	False	False	False	True	True	True	True	False	False
Nevergrad	n_estimators	400	400	400	200	400	200	200	50	50	50	400	400	400
	min_samples_split	2	20	5	2	10	2	20	20	2	2	5	2	2
	min_samples_leaf	1	2	1	2	1	1	1	4	2	2	2	1	2
	max_depth	10	40	None	10	40	None	20	None	40	20	None	40	20
	bootstrap	False	True	False	False	False	True	False	False	False	False	False	True	True

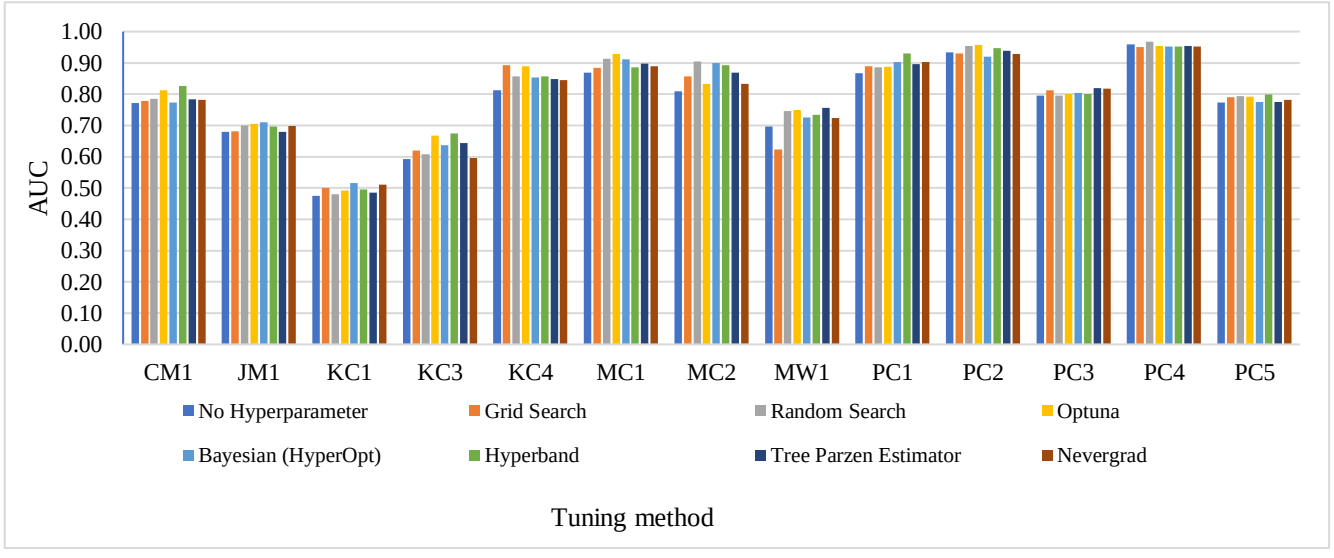
TABLE 3
Random Forest performance with hyperparameter tuning

Data		Tuning							
		No Tuning	Grid Search	Random Search	Optuna	Bayesian Search (Hyperopt)	Hyperband	Tree Parzen Estimator	Nevergrad
CM1	Accuracy	0.8182	0.8333	0.8636	0.8333	0.8333	0.8333	0.8636	0.8485
	AUC	0.7719	0.7785	0.7851	0.8132	0.7736	0.8264	0.7835	0.7818
JM1	Accuracy	0.7264	0.7579	0.7752	0.7662	0.7797	0.7791	0.7649	0.7592
	AUC	0.6789	0.6810	0.7007	0.7048	0.7095	0.6960	0.6791	0.6977
KC1	Accuracy	0.6750	0.7000	0.7000	0.7000	0.7250	0.7750	0.7500	0.8000
	AUC	0.4740	0.5000	0.4805	0.4913	0.5152	0.4957	0.4848	0.5108
KC3	Accuracy	0.6410	0.7179	0.6667	0.7949	0.7436	0.7436	0.7179	0.7179
	AUC	0.5926	0.6204	0.6074	0.6667	0.6370	0.6741	0.6444	0.5963
KC4	Accuracy	0.7600	0.8000	0.8000	0.8000	0.8400	0.7600	0.8000	0.8400
	AUC	0.8117	0.8929	0.8571	0.8896	0.8539	0.8571	0.8474	0.8442
MC1	Accuracy	0.9623	0.9673	0.9698	0.9698	0.9673	0.9698	0.9673	0.9749
	AUC	0.8695	0.8834	0.9135	0.9284	0.9111	0.8849	0.8979	0.8897
MC2	Accuracy	0.7200	0.7600	0.7600	0.7600	0.7600	0.7600	0.8000	0.8400
	AUC	0.8095	0.8571	0.9048	0.8333	0.8988	0.8929	0.8690	0.8333

Data		Tuning							
		No Tuning	Grid Search	Random Search	Optuna	Bayesian Search (Hyperopt)	Hyperband	Tree Parzen Estimator	Nevergrad
MW1	Accuracy	0.7843	0.7843	0.8235	0.8039	0.8039	0.7843	0.8431	0.8235
	AUC	0.6963	0.6232	0.7463	0.7500	0.7256	0.7335	0.7561	0.7244
PC1	Accuracy	0.9007	0.9007	0.9362	0.9291	0.9433	0.9530	0.9078	0.9149
	AUC	0.8672	0.8884	0.8857	0.8866	0.9034	0.9293	0.8957	0.9031
PC2	Accuracy	0.9799	0.9664	0.9799	0.9799	0.9664	0.9732	0.9732	0.9664
	AUC	0.9336	0.9302	0.9539	0.9569	0.9207	0.9474	0.9379	0.9276
PC3	Accuracy	0.8287	0.8287	0.8380	0.8472	0.8380	0.8519	0.8380	0.8565
	AUC	0.7947	0.8125	0.7957	0.8002	0.8042	0.8004	0.8199	0.8170
PC4	Accuracy	0.9147	0.9186	0.9264	0.9031	0.9186	0.9225	0.9341	0.9225
	AUC	0.9589	0.9509	0.9668	0.9544	0.9527	0.9515	0.9544	0.9524
PC5	Accuracy	0.7638	0.7434	0.7434	0.7522	0.7347	0.7522	0.7230	0.7172
	AUC	0.7733	0.7910	0.7945	0.7926	0.7749	0.7993	0.7742	0.7820



(a)



(b)

FIGURE 2. Comparison of each hyperparameter tuning on Random Forest performance (a) Random Forest performance on accuracy, (b) Random Forest performance on AUC

TABLE 4
Average increase value on all dataset

Hyperparameter Tuning	Increase Value	
	Accuracy	AUC
Grid Search	0.0157	0.0136
Random Search	0.0237	0.0277
Optuna	0.0280	0.0335
Bayesian (Hyperopt)	0.0291	0.0268
Hyperband	0.0294	0.0351
TPE	0.0314	0.0240
Nevergrad	0.0390	0.0176

IV. DISCUSSION

FIGURE 3 and FIGURE 4 show a comparative examination of the three best hyperparameter tuning methods to increase Accuracy and AUC on average. This comparison reveals that Nevergrad exceeds the other hyperparameter tuning methods in terms of accuracy, while Hyperband outperforms the remaining hyperparameter tuning method in terms of AUC.

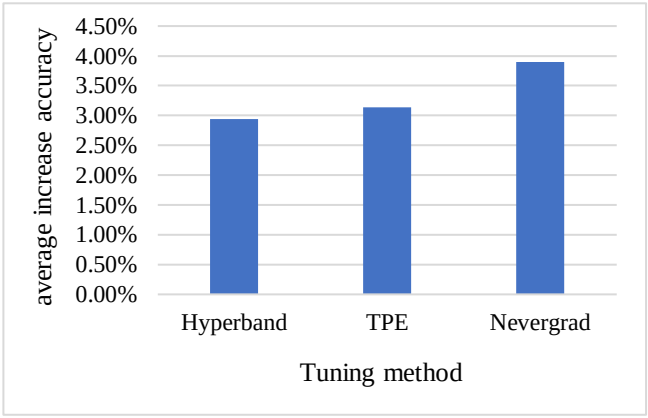


FIGURE 3. Comparison of accuracy performance of each tuning

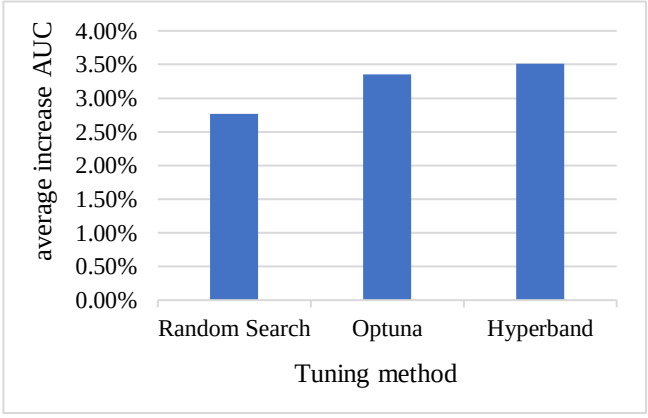


FIGURE 4. Comparison of AUC performance of each tuning

TABLE 5 compares the performance of the models used in this study to the models from the previous study. Previous studies have proposed the IRF approach to increase Random Forest performance. Research has been conducted out and employs PC1, PC2, PC3, PC4 dataset.

TABLE 5
Comparison with previous research

Dataset	Previous Research Method (Accuracy) [10]		Proposed Research with RF (Accuracy)
	RF	IRF	
PC1	92.9%	94.5%	**95.3%**
PC2	98.3%	**98.5%**	98%
PC3	89.2%	**89.6%**	85.65%
PC4	88.2%	90.625%	**92.25%**

The comparison of the two prior studies demonstrates that hyperparameter tuning has the ability to exceed or reach comparable results to previous research. In contrast, when hyperparameter tuning was used on PC1 and PC4, the accuracy value was effectively enhanced by 2.4% and 4.05%, respectively. The accuracy value was also effectively raised in IRF classification around 0.8% on PC1 and 1.625% on PC4.

TABLE 6 compares the performance of the models used in this study to the models from the previous study. Previous studies have proposed the fusion of conventional Artificial Neural Network (ANN) and the inventive Artificial Bee Colony (ABC) machine learning. The other research uses CS-ILDM, ANN-ABC, LDM, and NB. Research has been conducted and employs CM1, PC1, and JM1 datasets.

TABLE 6
Comparison with other research method

Research	Method	Dataset (AUC)		
		CM1	PC1	JM1
[8]	ANN-ABC	0.77	0.82	0.71
	NB	0.75	0.70	0.68
	RF	0.74	0.85	0.75
	C4.5	0.53	0.68	0.61
[9]	CS-ILDM	0.771	0.856	**0.747**
	ANN-ABC	0.773	0.823	0.711
	LDM	0.546	0.603	0.589
	NB	0.716	0.638	0.679
Proposed Research	RF	**0.826**	**0.929**	0.71

Based on the data presented in TABEL 6, a comparison is made between different machine learning classifiers. In contrast, the proposed method using hyperparameter tuning and Genetic Algorithm feature selection produces the best performance on CM1 and PC1 datasets. Random Forest with this research approach outperforms various methodologies and outperforms the Random Forest without hyperparameter tuning. However, there are weaknesses obtained based on TABLE 5 and TABLE 6 where there are several performances that cannot be superior compared to other methods. One of the weaknesses of this method is that the parameters obtained can provide better performance and can also provide worse performance. It also has some limitations, especially in search space. Despite all that, it can be confirmed that the results of this research outperform previous research. The AUC value

and accuracy of Random Forest using Genetic Algorithm and hyperparameter tuning outperform the average AUC value and accuracy value of other methodologies.

V. CONCLUSION

Software defect prediction is important for desktop and mobile applications because it helps developers see possible problems before they happen, fix mistakes faster, enhance the quality of the program, and lessen the negative impact on the user experience. This research emphasizes enhancing the performance of Random Forest classification in terms of accuracy and AUC in software defect prediction. When compared to earlier research employing various classifications, it was discovered that Genetic Algorithm feature selection and hyperparameter tuning successfully improved accuracy and AUC results. According to the research findings, Random Forest classification using the Genetic Algorithm feature selection and hyperparameter tuning produces better results than other classification methods used in earlier studies. This is proven by the AUC values for CM1 and PC1 of around 0.826 and 0.929. This is also proven by the accuracy values on PC1 and PC4 of 0.953 and 0.922. According to this research, Hyperband has the best hyperparameter tuning performance with the highest average increase in accuracy and AUC. Hyperband increases the accuracy value on the entire dataset by 2.94% and increases the AUC value by 3.51%.

This research still has several limitations, it can be seen that there is some decrease in accuracy and AUC performance on certain datasets. In future studies, it is recommended to utilize this approach alongside additional classification methodologies to forecasting software defects. The objective is to determine the best classification method to anticipate software defects. It also recommended to use of larger search space in hyperparameter tuning. The aim is to find out a more optimal parameter to use in the classification method to achieve enhanced efficiency or effectiveness.

ACKNOWLEDGEMENT

We extend our gratitude to all the individuals associated with the Computer Science study program, Faculty of Mathematics and Natural Sciences, University of Lambung Mangkurat,for their invaluable support and provision of resources that facilitated the completion of this research. Our heartfelt appreciation also goes out to our fellow project team members for their dedication and collaborative efforts, which greatly contributed to the achievements of this study.

REFERENCES

[1] H. K. Dam *et al.*, "A deep tree-based model for software defect prediction," Feb. 2018, [Online]. Available: http://arxiv.org/abs/1802.00921

[2] Z. Li, X. Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3. Institution of Engineering and Technology, pp. 161–175, Jun. 01, 2018. doi: 10.1049/iet-sen.2017.0148.

[3] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proceedings - International Conference on Software Engineering*, IEEE Computer Society, May 2016, pp. 309–320. doi: 10.1145/2884781.2884839.

[4] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *Proceedings - 2017 International Conference on New Trends in Computing Sciences, ICTCS 2017*, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 252–257. doi: 10.1109/ICTCS.2017.39.

[5] R. S. Wahono, "A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks," *Journal of Software Engineering*, vol. 1, no. 1, 2015, [Online]. Available: http://journal.ilmukomputer.org

[6] G. Rana, E. U. Haq, E. Bhatia, and R. Katarya, "A Study of Hyper-Parameter Tuning in the Field of Software Analytics," in *Proceedings of the 4th International Conference on Electronics, Communication and Aerospace Technology, ICECA 2020*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, pp. 455–459. doi: 10.1109/ICECA49313.2020.9297613.

[7] B. F. F. Huang and P. C. Boutros, "The parameter sensitivity of random forests," *BMC Bioinformatics*, vol. 17, no. 1, Sep. 2016, doi: 10.1186/s12859-016-1228-x.

[8] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Applied Soft Computing Journal*, vol. 33, pp. 263–277, Apr. 2015, doi: 10.1016/j.asoc.2015.04.045.

[9] C. Jin, "Software defect prediction model based on distance metric learning," *Soft comput*, vol. 25, no. 1, pp. 447–461, Jan. 2021, doi: 10.1007/s00500-020-05159-1.

[10] K. R. Magal, S. Gracia Jacob, and A. Professor, "Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques," 2015.

[11] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "NASA MDP Software Defects Data Sets," *IEEE Transactions on Software Engineering 39(9)*, pp. 1208–1215, 2018, doi: https://doi.org/10.6084/m9.figshare.c.4054940.v1.

[12] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200–1219, Nov. 2020, doi: 10.1109/TSE.2018.2876537.

[13] C. Zhang, J. Song, Z. Pei, and J. Jiang, "An Imbalanced Data Classification Algorithm of De-noising Auto-Encoder Neural Network Based on SMOTE," *MATEC Web of Conferences ICCAE 2016*, 2016, doi: 10.1051/conf/2016.

[14] K. A. Putri, W. Fawwaz, and A. Maki, "Enhancing Pneumonia Disease Classification using Genetic Algorithm-Tuned DCGANs and VGG-16 Integration," *Open Access Journal*, vol. 6, no. 1, pp. 11–22, 2024, doi: 10.35882/jeemi.v6i1.349.

[15] S. Aalaei, H. Shahraki, A. Rowhanimanesh, and S. Eslami, "Feature selection using genetic algorithm for breast cancer diagnosis: experiment on three different datasets," 2016.

[16] R. B. Bahaweres, A. Imam Suroso, A. Wahyu Hutomo, I. Permana Solihin, I. Hermadi, and Y. Arkeman, "Tackling Feature Selection Problems with Genetic Algorithms in Software Defect Prediction for Optimization," in *Proceedings - 2nd International Conference on Informatics, Multimedia, Cyber, and Information System, ICIMCIS 2020*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, pp. 64–69. doi: 10.1109/ICIMCIS51567.2020.9354282.

[17] B. H. Shekar and G. Dagnew, *Grid Search-Based Hyperparameter Tuning and Classification of Microarray Cancer Data 2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*. IEEE, 2019.

[18] T. N. Nuklianggraita, A. Adiwijaya, and A. Aditsania, "On the Feature Selection of Microarray Data for Cancer Detection based on Random Forest Classifier," *JURNAL INFOTEL*, vol. 12, no. 3, pp. 89–96, Aug. 2020, doi: 10.20895/infotel.v12i3.485.

[19] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The Impact of Automated Parameter Optimization on Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 683–711, Jul. 2019, doi: 10.1109/TSE.2018.2794977.

[20] P. Probst, M. N. Wright, and A. L. Boulesteix, "Hyperparameters and tuning strategies for random forest," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 3. Wiley-Blackwell, May 01, 2019. doi: 10.1002/widm.1301.

[21] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications," Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.05689

[22] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, Jul. 2019, pp. 2623–2631. doi: 10.1145/3292500.3330701.

[23] K. Cheng and S. Takada, "Software Defect Prediction based on JavaBERT and CNN-BiLSTM," 2023. [Online]. Available: http://ceur-ws.org

[24] S. Shekhar, A. Bansode, and A. Salim, "A Comparative study of Hyper-Parameter Optimization Tools," Jan. 2022, [Online]. Available: http://arxiv.org/abs/2201.06433

[25] F. F. Firdaus, H. A. Nugroho, and I. Soesanti, "Deep Neural Network with Hyperparameter Tuning for Detection of Heart Disease," in *Proceedings - 2021 IEEE Asia Pacific Conference on Wireless and Mobile, APWiMob 2021*, Institute of Electrical and Electronics Engineers Inc., Apr. 2021, pp. 59–65. doi: 10.1109/APWiMob51111.2021.9435250.

[26] H. Erbin and R. Finotello, "Machine learning for complete intersection Calabi-Yau manifolds: A methodological study," *Physical Review D*, vol. 103, no. 12, Jun. 2021, doi: 10.1103/PhysRevD.103.126014.

[27] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn," 2014.

[28] J. Bergstra, B. Komer, D. Yamins, C. Eliasmith, and D. D. Cox, "Computational Science & Discovery Hyperopt: a Python library for model selection and hyperparameter optimization," 2015.

[29] S. Putatunda and K. Rama, "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Nov. 2018, pp. 6–10. doi: 10.1145/3297067.3297080.

[30] S. Falkner, A. Klein, and F. Hutter, "Combining Hyperband and Bayesian Optimization," 2017.

[31] J. Wang, J. Xu, and X. Wang, "Combination of Hyperband and Bayesian Optimization for Hyperparameter Optimization in Deep Learning," Jan. 2018, [Online]. Available: http://arxiv.org/abs/1801.01596

[32] H. Li, Q. Zhang, X. Qin, and S. Yuantao, "Raw vibration signal pattern recognition with automatic hyper-parameter-optimized convolutional neural network for bearing fault diagnosis," *Proc Inst Mech Eng C J Mech Eng Sci*, vol. 234, no. 1, pp. 343–360, Jan. 2020, doi: 10.1177/0954406219875756.

[33] C. Maurice, F. Madrigal, and F. Lerasle, "Hyper-optimization tools comparison for parameter tuning applications," *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) : Aug. 29 2017-Sept. 1 2017.*, 2017.

[34] J. Rapin, M. Gallagher, P. Kerschke, M. Preuss, and O. Teytaud, "Exploring the MLDA benchmark on the Nevergrad platform," in *GECCO 2019 Companion - Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, Inc, Jul. 2019, pp. 1888–1896. doi: 10.1145/3319619.3326830.

[35] J. Rapin, P. Bennet, E. Centeno, D. Haziza, A. Moreau, and O. Teytaud, "Open source evolutionary structured optimization," in *GECCO 2020 Companion - Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, Inc, Jul. 2020, pp. 1599–1607. doi: 10.1145/3377929.3398091.

[36] M. R. Ansyari, M. I. Mazdadi, F. Indriani, D. Kartini, and T. H. Saragih, "Implementation of Random Forest and Extreme Gradient Boosting in the Classification of Heart Disease Using Particle Swarm Optimization Feature Selection," *Open Access Journal*, vol. 5, no. 4, pp. 250–260, 2023, doi: 10.35882/jeemi.v5i4.322.

[37] K. Vijiyakumar, B. Lavanya, I. Nirmala, and S. S. Caroline, *Random Forest Algorithm for the Prediction of Diabetes*. 2019.

[38] V. Maulida, R. Herteno, M. R. Faisal, D. Kartini, and F. Abadi, "Feature Selection Using Firefly Algorithm with Tree-Based Classification in Software Defect Prediction," *Open Access Journal*, vol. 5, no. 4, pp. 223–230, 2023, doi: 10.35882/jeemi.v5i4.315.

[39] Y. N. Soe, P. I. Santosa, and R. Hartanto, *Software Defect Prediction Using Random Forest Algorithm 2018 12th South East Asian Technical University Consortium (SEATUC)*. IEEE, 2018.

[40] I. Syarif, A. Prugel-Bennett, and G. Wills, "SVM Parameter Optimization using Grid Search and Genetic Algorithm to Improve Classification Performance," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 14, no. 4, p. 1502, Dec. 2016, doi: 10.12928/telkomnika.v14i4.3956.

BIBLIOGRAPHY

Mulia Kevin Suryadi originated in Martapura, Banjar, South Kalimantan. Since 2020, he has pursued his academic endeavors as a student of the Computer Science Department at Lambung Mangkurat University. His current area of research lies within the realm of software engineering. He has selected this particular interest due to his affinity towards software engineering. Additionally, his final project involves research centered on predicting software defects. The goal of this research is to predict defects in software.

Rudy Herteno, was born in Banjarmasin, South Kalimantan. After graduating from high school, he pursued his undergraduate studies in the Computer Science Department at Lambung Mangkurat University and graduated in 2011. After completing his undergraduate program, he worked as a software developer to gather experience for several years. He developed a lot of software, especially for local governments. In 2017, He completed his master's degree in Informatics from STMIK Amikom University. Currently, he is a lecturer in the Faculty of Mathematics and Natural Science at Lambung Mangkurat University. His research interests include software engineering, software defect prediction, and deep learning.

Setyo Wahyu Saputro is a lecturer in Computer Science Department, Faculty of Mathematics and Natural Science, Lambung Mangkurat University in Banjarbaru. He received bachelor's degree also in Computer Science from Lambung Mangkurat University, and received his master's degree in Informatics from STMIK Amikom University. His research interests include software engineering and artificial intelligence applications.

Mohammad Reza Faisal was born in Banjarmasin. Following his graduation from high school, he pursued his undergraduate studies in the Informatics department at Pasundan University in 1995, and later majored in Physics at Bandung Institute of Technology in 1997. After completing his bachelor's program, he gained experience as a training trainer in the field of information technology and software development. Since 2008, he has been a lecturer in computer science at Universitas Lambung Mangkurat, while also pursuing his master's program in Informatics at Bandung Institute of Technology in 2010. In 2015, he furthered his education by pursuing a doctoral degree in Bioinformatics at Kanazawa University, Japan. To this day, he continues his work as a lecturer in Computer Science at Universitas Lambung Mangakurat. His research interests encompass Data Science, Software Engineering, and Bioinformatic.

Radityo Adi Nugroho received his bachelor's degree in Informatics from the Islamic University of Indonesia and a master's degree in Computer Science from Gadjah Mada University. Currently, he is an assistant professor in the Department of Computer Science at Lambung Mangkurat University. His research interests include software defect prediction and computer vision. He is also a practitioner in the field of information technology as a project manager and systems analyst to develop software and information systems used by universities.