Fine-tuned Random Forest-based Software Defect Prediction via Metaheuristics

Tadashi Dohi, Jingchi Wu, Junjun Zheng and Hiroyuki Okamura

Graduate School of Advanced Science and Engineering, Hiroshima University
Higashi-Hiroshima 739-8527, Japan
email: {dohi, kure, jzheng, okamu}@hiroshima-u.ac.jp

***Abstract*—Software defect prediction is one of the most fundamental issues to identify defect-prone modules in module testing, and enables to reduce effectively the module testing cost. While several machine learning techniques are applied to the software defect prediction, it is well known that ensemble methods such as random forest are quite useful to detect the defect-prone modules with high predictive performance and low computation cost. Since the Random Forest (RF) contains the hyperparameters, which have to be carefully tuned in advance, the prediction accuracy on the defect-proneness strongly depends on the tuning results in the random forest. In this paper, we propose to apply three metaheuristic algorithms: Latin Hypercube Sampling (LHS) algorithm, Artificial Bee Colony (ABC) algorithm, and Parameter free Genetic Algorithm (PfGA), to search the optimal hyperparameters in the RF-based software defect prediction, and investigate the predictive performance of defect-prone modules. In experiments with six actual software development projects, we compare our fine-tuned RF algorithms with the existing machine learning approaches. It is empirically shown that the fine tuning via metaheuristics could provide better predictive performances on F-score in software defect prediction.**

Keywords-software defect prediction; defect-prone module; random forest; hyperparameters; metaheuristics; fine tuning; search-based software engineering.

I. INTRODUCTION

In almost all software development projects, it is necessary to spend a large amount of testing time and cost, because software systems are implicitly expected to be defect-free after releasing to the users or market. On one hand, it is known that the requirement of defect-free software is rather stringent and cannot be achieved in almost all the development projects, since the software testing period is limited but the testing cost is rather expensive. Hence, the practical and ad hoc approach would be to shorten the testing period by improving the efficiency of software test. Many experiences suggest that software defects should be fixed timely and exactly from the defect-prone modules, if they can be identified. If sufficient testing resources, including testing personnel for review and highly advanced test case generation algorithms etc., can be allocated to the software module testing, it may be possible to detect/fix an enough number of defects even in the module testing level.

However, the defect-prone modules in large-scaled software systems with a number of modules tend to be sparse, testing/reviewing all the modules is not cost-effective. *Software defect prediction* is one of the most fundamental issues to identify the defect-prone modules before the review/test, and enables to reduce effectively the module testing cost. It can be defined as a *statistical discriminant problem* and be characterized by *software defect-prone module probability*, which is defined as the probability that a module contains at least one software defect. More specifically, suppose that the binary data that each module was defect-prone (1) or defect-free (0) are available as training data, and that feature data, called *software metrics*, involving the module size, complexity and development effort, are observed for all the modules. Based on the training data and the feature data, one predicts the defect-prone module probabilities for the remaining modules that have not been reviewed yet, and judges whether they are defect-prone or not.

During almost the last four decades, several machine learning techniques were applied to the software defect prediction. In early phase, standard discriminate analysis techniques such as logistic regressions, multi-layer perceptron neural networks, naive Bayes, etc. have been utilized for the software defect prediction. There are classified into two approaches: deep neural network approach [1] and ensemble approach [29]. In the former approach, the deep neural networks [2] [3] [26] are commonly used techniques to relate software metrics data to the binary discriminant problem. The latter approach is based on boosting algorithms, such as XGBoost [8] and AdaBoost [12], and bagging algorithms [5]. Laradji et al. [17] combined feature selection and ensemble learning on the performance of defect classification, where a new two-variant ensemble learning method is proposed with and without feature selection. Tong et al. [23] applied stacked denoising autoencoders for feature learning in software defect prediction, and combined the deep learning with two-stage ensemble. Riaz et al. [22] developed a novel method by combining rough set theory, K-nearest neighbor rule and noise-filter technique to deal with imbalanced classes for software defect prediction.

In this paper, we focus on random forest (RF) -based software defect prediction. RF proposed by Breiman [6] is a representative ensemble method by extending the well-known decision tree algorithms, and can be also used in software defect prediction with high predictive performance and low computation cost. Similar to the other machine learning methods, RF contains hyperparameters, which have to be carefully tuned in advance [21]. Malhotra and Khanna [20] used four strategies of ensemble learning to predict change prone classes by combining seven individual Particle Swarm Optimization (PSO)-based classifiers as constituents of ensembles and aggregating them using weighted voting. Turabieh et al. [25] employed three metaheuristics: Binary Genetic Algorithm

(BGA), binary PSO, and Binary Ant Colony Optimization (BACO), for feature selection problem in software defect prediction. Alsghaier and Akour [4] proposed an approach by integrating GA with support vector machine classifier and PSO for software defect prediction. Zhang et al. [28] considered a novel defect prediction model based on stacked sparse denoising autoencoders in [23], extreme deep learning machine optimized by PSO and another complementary gravitational search algorithm. Khan et al. [14] investigated software defect prediction models with seven machine learning classifiers in conjunction with the artificial immune network by optimizing their hyperparameters. Recently, Thomas and Kaliraj [24] used a fine-tuned RF classifier to optimize hyperparameters in addition to applying an oversampling technique, and enhanced predictive accuracy in software defect prediction.

We consider RF-based software defect prediction, which differs from [14] and [24] in that the hyperparameters of RF are tuned by multiple metaheuristic algorithms. Note that the original RF by Breiman [6] consists of the depth of trees, the number of trees and the number of features, which are regarded as hyperparameters. The commonly used technique to determine the hyperparameters is based on the preliminary experiments to tune them. In fact, several ensemble methods involving RF are available in free tools, where the default values of the hyperparameters are set up. It is worth mentioning that these default values must be tuned carefully or optimized, if possible, for specific problems, but some authors seem to use the default values without doubt in many cases. The most well-known technique to determine the hyperparameters in RF would be grid search [21]. However, the grid search is also a heuristic algorithm, and does not guarantee the globally optimal hyperparameters.

The main challenge of this paper is to apply three metaheuristic algorithms: Latin Hypercube Sampling (LHS) algorithm, Artificial Bee Colony (ABC) algorithm, and Parameter free Genetic Algorithm (PfGA), to search the optimal hyperparameters in the RF-based software defect prediction, LHS algorithm, which was developed in Los Alamos National Laboratory [19], is a classical heuristic method for generating a near-random sample of parameter values from a multi-dimensional distribution. ABC algorithm is an optimization algorithm inspired by the intelligent foraging behavior of honey bee swarm [13]. Kondo and Asanuma [15] applied the ABC algorithm to optimize the hyperparameters in RF. PfGA is an intuitive but most fundamental technique to determine the hyperparameters in ensemble methods. We tune the RF-based software defect prediction models by the above three metaheuristics, and compare them with the grid search approach and the other boosting-based software defect prediction models, such as AdaBoost [12], XGBoost [8] and Light Gradient Boosting Machine (GBM) developed by Microsoft [16].

The remaining part of this paper is organized as follows. In Section II, we formulate the software defect prediction as a statistical discriminant analysis. Section III overviews the RF. Section IV introduces three metaheuristic algorithms: LHS, ABC and PfGA. The experiments with six NASA data sets: CM1, KC1, KC3, MW1, MC1 and MC2, are presented in Section V, where we compare our fine-tuned RF-based software defect prediction models with the common grid search as well as the existing boosting models: AdaBoost, XGBoost, Light GBM, and common Multi-Layer Perceptron (MLP) neural network. It is empirically shown that the fine tuning via metaheuristics could provide better predictive performances in terms of F-score.

II. Software Defect Prediction: Formulation

In software development projects, detecting and fixing software defects in limited testing resources are rather expensive, but play a significant role to assure software reliability. There are two major problems to make the software test effective: software defect localization and software defect prediction. The purpose of software defect localization is to identify the defect position on a software program. The software defect prediction focuses on the identification of defect-prone modules before reviewing/testing software program in the module testing. Identification of software defect-prone modules enables us to carry out software module test effectively.

Suppose that a software system under the module test consists of N modules. Let $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ denote the feature vector of the $i\,(= 1, 2, \ldots, N)$-th software module, where n types of features, called *software metrics*, are available in the coding phase. Define the binary random variable:

$$Y_i = \begin{cases} 1, & \text{if } i-\text{th-module contains software bugs,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Let $P_i(\boldsymbol{x}_i) = \mathrm{P}\{Y_i = 1 \mid \boldsymbol{x}_i\}$ be the defect-prone module probability which denotes a probability that the i-th module contains any software bug, conditioned that the software metrics $\boldsymbol{x}_i$ are given. Since Y_i is a Bernoulli random variable for fixed i, it is evident that $\mathrm{E}[Y_i] = P_i(\boldsymbol{x}_i)$.

For the simplest example, consider the logistic regression model:

$$P_i(\boldsymbol{x}_i) = P_i(\boldsymbol{x}_i \mid \boldsymbol{\beta}, \beta_0) = \frac{\exp(\boldsymbol{z}_i)}{1 + \exp(\boldsymbol{z}_i)}, \tag{2}$$

where $\exp(\boldsymbol{z}_i) = \boldsymbol{\beta}^T \boldsymbol{x}_i + \beta_0$ denotes the random variables representing the tendency of bug presence, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_n)$ is the regression coefficient vector, β_0 is a scalar constant, and T is the transpose. Since the defect-prone module probability $P_i(\boldsymbol{x}_i)$ is given by a function of $\boldsymbol{x}_i$, it turns out that the dependent variable becomes a monotonic function with respect to each module with $\boldsymbol{x}_i$. Define the parameter vector $\boldsymbol{\theta} = (\boldsymbol{\beta}, \beta_0)$. For realizations y_i of the binary random variable Y_i with feature vector $\boldsymbol{x}_i$, the log likelihood function is given by

$$\begin{aligned} \ln L(\boldsymbol{\theta}) \quad = \quad & \sum_{i=1}^{N} \Big\{ y_i \ln P_i(\boldsymbol{x}_i \mid \boldsymbol{\beta}, \beta_0) \\ & + (1 - y_i) \ln[1 - P_i(\boldsymbol{x}_i \mid \boldsymbol{\beta}, \beta_0)] \Big\}. \end{aligned} \tag{3}$$

By maximizing $\ln L(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, we get the maximum likelihood estimate of the parameter $\boldsymbol{\theta} = (\boldsymbol{\beta}, \beta_0)$, and predict the defect-prone module probabilities for the remaining modules that have not been reviewed. Recently, Dohi et al. [10] generalized the classical software defect prediction by introducing the semi-definite logistic regression. From the above example, it can be easily seen that software defect prediction is reduced to a statistical discriminant problem and that a number of machine learning algorithms containing deep neural network and ensemble methods can be applied.

We refer to a static software defect prediction. Suppose that there are $N = k + m$ software modules, where k modules were already reviewed and identified whether they were defect-prone or defect-free. Let $(y_i, \boldsymbol{x}_i)$ $(i = 1, 2, \ldots, k)$ be the training data. Based on the training data, we estimate $P_j(\boldsymbol{x}_j)$ $(j = k+1, k+2, \ldots, k+m)$, where $(y_j, \boldsymbol{x}_j)$ are the validation data. Let ξ $(0 < \xi < 1)$ denote the discriminant threshold, which is a cut off value to identify whether the j-th module is defect-prone or not. If $P_j(\boldsymbol{x}_j)$ is greater than ξ, then we identify that j-th module is defect-prone, otherwise, defect-free. In practice, only defect-prone modules will be reviewed in the module test. To evaluate the predictive performances of the machine learning models, we compare our prediction results for m modules with the validation data y_j $(j = k+1, k+2, \ldots, k+m)$ and obtain the confusion matrix which consists of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP and FP imply the respective numbers of defect-prone modules predicted correctly and incorrectly, TN and FN mean the numbers of defect-free modules predicted correctly and incorrectly. Define the following three prediction metrics:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{4}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{6}$$

Finally, by taking the harmonic mean of Precision and Recall, we obtain the F-score:

$$\text{F} - \text{score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{7}$$

The larger F-score implies the higher predictive performance in software defect prediction. It should be noted that F-score is not a unique prediction measure, because it also has some limitations [9] [18] [27]. Nevertheless, we focus on effects of fine-tuned RF in terms of the resulting F-scores, because it is still useful for general discriminant problems but not for only software bug prediction.

III. Random Forests

RF [6] is a representative of the state-of-the-art ensemble methods and a lightweight machine learning technique in terms of computation cost. It is a substantial extension of bagging (bootstrap aggregating) [5], where the major difference from bagging is to incorporate the randomized feature selection. For better understanding, we quickly overview the RF. Consider a decision tree, which is an algorithm that searches for the most matching leaves by tracing conditional branches from a root. During the construction of a component decision tree, at each step of split selection, RF randomly selects a subset of features, and then carries out the conventional split selection procedure within a selected feature subset. The combination of features and threshold that could reduce the impurity within each node is selected using the entropy. For the current node s, define the entropy:

$$H(s) = -\sum_{c \in \{0,1\}} p(c|s) \log p(c|s), \tag{8}$$

where

$$\Delta I(s) = H(s_a) - \sum_{i \in \{L,R\}} \frac{n_i}{k} H_i(s_b) \tag{9}$$

is the information gain, $p(c|s) = n_c/k$, c is the class of the objective variable, k is the number of training data, n_i is the number of data belonging to the class, s_a is the node before the branch, s_b is the node after the branch, $i = L$ and $i = R$ are the left and right nodes after the branch, respectively. The above equations are used to split the data and to find the optimal splitting condition at each node. We decide whether the end condition is satisfied or not, so if the end condition was satisfied, then we integrate the results. This procedure is repeated recursively until a specified number of times to create multiple decision trees is achieved. For software defect prediction, the results of each decision tree are integrated by means of a majority voting.

In our classification problem, the final prediction value becomes the output, so the class of the objective variable is set to 1 if it contains at least one software defect, otherwise, set to 0. The posterior probability $p(c|s)$ in (8) can be expressed as the total number of software modules containing defects in the leaf node at that time. The combination of thresholds that minimize the posterior probabilities is selected based on the entropy in (8) and the information gain in (9).

Hence, it is obvious to see that we encounter problems to determine the number of trees and the depth of each tree, and a problem which feature should be applied for the discriminant analysis. Note that the parameter of features controls to incorporate the randomness. If all the numbers of features equal, then RF is reduced to a deterministic decision tree. If the parameter of features is given by 1, then one feature is selected randomly. It is known that the depth of the decision tree used in a weak learner is a significant parameter in terms of prediction accuracy. To this end, the predictive performance in RF varies significantly depending on the depth of trees. Also, since RF creates a specified number of trees, the final output by a majority voting with all the trees, and the resulting prediction accuracy depends on the predictive performance. Unfortunately, almost all RF tools request to tune the number of trees, the depth of each tree and the number

of features in advance. In the following section, we introduce three metaheuristics to determine these hyperparameters.

IV. Metaheuristic Algorithms

In this section, we summarize three metaheuristic algorithms employed in the paper.

A. Latin Hypercube Sampling Algorithm

LHS is a method for sampling the search space uniformly within a specified number of iterations [19]. First, we specify the number of iterations α, and divide the search space into α intervals for each feature. We draw one sample randomly from each interval, and take place an experiment by combining it with other features. For RF, we use the above LHS algorithm to find a combination of tree depths, number of trees, and number of features. It is assumed in our study that each depth ranges in 2∼50, the number of trees in 10∼1000, and the number of features in 1∼38. We repeat these random selections 1000 times, and prepare 1000 sets of hyperparameters. Let ϕ be a uniformly distributed pseudo random number. For the hyperparameters $\boldsymbol{h} = (h_1, \ldots, h_{1000})$, we find the maximum value, $\max(\boldsymbol{h})$ and the minimum value, $\min(\boldsymbol{h})$. Then the LHS-based hyperparameters are given by $\phi\{\max(\boldsymbol{h}) - \min(\boldsymbol{h})\} + \min(\boldsymbol{h})$.

B. ABC Algorithm

The ABC algorithm is an optimization algorithm that imitates the behavior of honey bee swarm and is divided into three phases [13]. The first phase called the *harvesting bee phase* is to update each candidate solution. In the second phase called the *bee phase*, the updated candidate solution is selected probabilistically in accordance with the degree of adaptation of each candidate solution. In the final *reconnaissance bee phase*, the candidate solutions that have not been updated within a pre-determined number of times are randomly generated, and are replaced by new ones. In updating the solution, the (i, j)-element of the candidate solution, denoted by i-th row and j-th column vector, is given by $\boldsymbol{v}_{i,j} = \boldsymbol{x}_{i,j} + \phi(\boldsymbol{x}_{i,j} - \boldsymbol{x}_{k,j})$. Then we compare with the newly generated candidate solution and replace it by the one with the higher adaptivity $1/\{1 + f(\boldsymbol{x}_i)\}$ with any adaptive function $f(\cdot)$, where ϕ is a uniformly distributed pseudo random number in the interval $[0, 1]$. Following Kondo and Asanuma [15], we use the ABC algorithm to find a combination of tree depth, number of trees, and number of features. Set 2∼50 for the depth, 10∼1000 for the number of trees, and 1∼38 for the number of features as well. Next, we prepare 1000 candidates of hyperparameters. Substituting these hyperparameters into $x_{i,j}$ yields the updated hyperparameters.

C. Parameter Free GA

PfGA is a genetic algorithm that eliminates the need to select parameters with crossover rate, mutation rate and crossover method. In the crossover step, two individuals are randomly selected from the local population to perform a multipoint crossover with random number and random location.

TABLE I. DATA SETS.

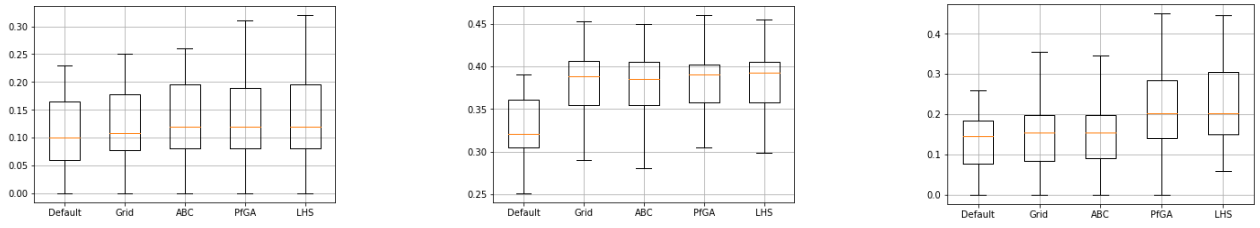
	No. modules	Defect proneness (%)	No. metrics
CM1	327	12.80	38
KC1	2108	9.96	38
KC3	458	9.17	38
MW1	403	7.70	38
MC1	9466	7.18	38
MC2	161	32.20	38

Next, two individuals are randomly selected from the local population and are mutated at random numbers and positions. In the selection, based on the evaluation points of these four individuals of the family, we perform the selection procedure and return the operation to the local population. If both the offspring are better than the two parents, then we form a local population from the two offspring and the better off parent. If the two offspring are worse than the two parents, then we form a local population with the better parent. If only one of the two parents is better than the two children, then the better parent and the better child form a local population. If only one of the two offspring is better than the two parental offspring, then a local population is formed with the better offspring and a randomly selected individual from the entire search space. In our software defect prediction problem, each parameter is used as genetic information to perform the operation. For the hyperparameters in RF, we prepare two random combinations of parameters and put them into the above algorithm.

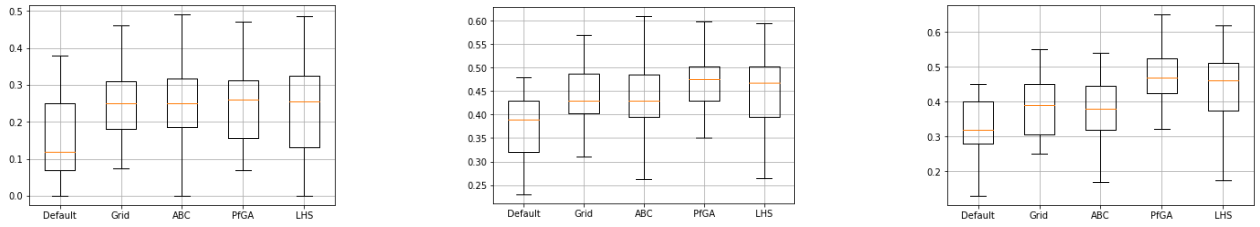
V. Experiments

The well-known benchmark data sets for software defect prediction come from NASA MPD. In Table 1, we present six software programs: CM1, KC1, KC3, MW1, MC1, MC2 with each 37 software metrics. In our experiments, we compare the predictive performances of RFs tuned by grid-search (Grid), LHS, ABC and PfGA with the RF with default hyperparameters (Default). We also compare our refined RFs with four conventional machine learning techniques: MLP with three layers, Adaboost, XGBoost, and LightGBM. Through experiments, we set the discriminant threshold as $\xi = 0.5$. Although the argument to control the discriminant threshold itself exists, we emphasize that the software defect-prone module probability is symmetric to justify our assumption. It is also well-known that the predictive performance in software defect prediction strongly depends on the sampled module, where k modules are sampled as the training data. In order to reduce the prediction bias, we applied the holdout cross-validation and selected randomly 50% of the total number of modules 100 times in each data set.

In Figure 1, we give the box plots of F-scores predicted by RFs and compare the predictive performances of RFs with metaheuristics (Grid, LHS, ABC, PfGA). In all data sets, it is seen that the metaheuristic approaches worked significantly better than the default case. So, we could observe that the tuning of hyperparameters is effective to improve the predictive performance in software defect prediction. Comparing

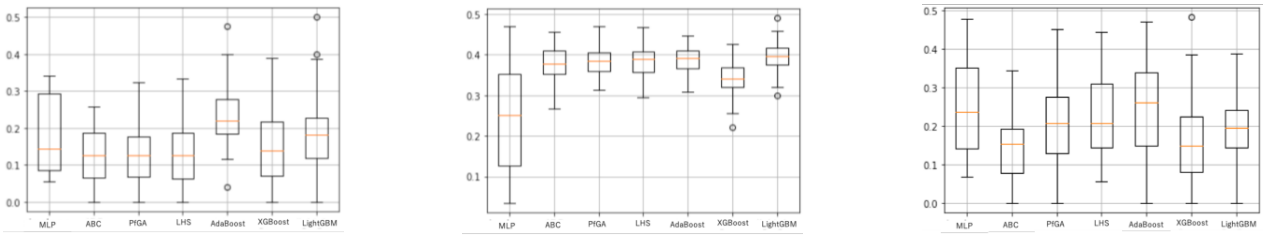


(a) CM1. (b) KC1. (c) KC3.

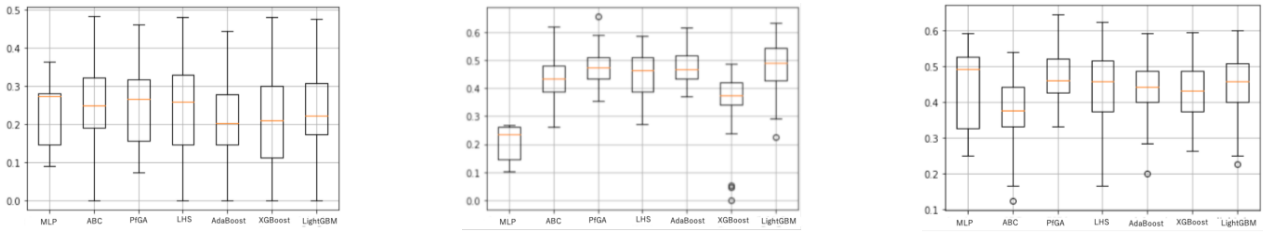


(d) MW1. (e) MC1. (f) MC2.

Figure 1: Comparison of RFs with four metaheuristics in terms of F-score.



(a) CM1. (b) KC1. (c) KC3.



(d) MW1. (e) MC1. (f) MC2.

Figure 2: Comparison with other machine learning algorithms in terms of F-score.

with the grid search-based RF, it can be seen that ABC could not show the larger median values than Grid in KC1, KC3, MW1, MC1 and MC2. However, we find that LHS and PfGA provided the larger median values than Grid in all data sets. In the comparison between LHS and PfGA, we can see that PfGA gave slightly better prediction results than LHS. From these results, we can conclude that the refinement with metaheuristics in RFs could improve the F-scores in software defect prediction, and could outperform the baseline RF with default hyperparameters. However, it should be noted that the lengths from lower quartile to upper quartile in LHS and ABC are longer than those in Default and Grid in CM1, MW1, MC1 and MC2. This implies that the tuning of hyperparameters with LHS and ABC tends to show the optimistic prediction with higher median and variance.

Next we concern the comparison of our fine-tuned RFs with the other boosting algorithms (Adaboost, XGBoost, LightBGM) and the classical MLP. Figure 2 gives the box plots in terms of the F-scores with seven machine learning techniques. In CM1 and KC3, AdaBoost gave the highest F-scores among others. In KC1 and MC1, LightGBM was the best predictor. In MC2 and MW1, MLP could provide the highest F-scores. In this way, as seen in the existing works, the machine learning technique with the highest predictive performance depends on the kind of data sets. However, it should be noted that our refined RFs could be the second and/or third best predictors

 ISBN: 978-1-68558-398-9

in many cases except CM1 and KC3, and could provide rather stable prediction results. Especially, it can be shown that the variance of PfGA is smaller than that of LHS expect in KC3. From these results, we agree that more recent boosting algorithms, such as XGBoost and LightBGM tended to give better prediction results, but conclude that the refinement of hyperparameters in the classical RF could improve the predictive performances effectively.

VI. Conclusion and Future Work

In this paper, we have improved the RFt-based software defect prediction by applying three metaheuristic algorithms; LHS algorithm, ABC and PfGA, to search the optimal hyperparameters. Through experiments with six NASA MPD data sets, we have investigated the predictive performances of software defect-prone modules, and compared our refined random forests with the default RF approach as well as the existing machine learning approaches. It has been shown that the RF with metaheuristics could provide stable prediction results in many cases.

In the future, we will apply the other metaheuristics, such as PSO, BGA, binary PSO, and BACO, for further potential improvement of RF-based software defect prediction. Also, effects of fine tuning should be investigated in different prediction measures from F-score [9] [18] [27]. Another direction is to improve the predictive performance by tuning the hyperparameters and preprocessing the imbalanced classes in software modules. In general, the number of defect-prone modules is sparse in all the software modules. For such imbalanced data, almost all machine learning models cannot work to guarantee satisfactory F-scores. Then, the so-called oversampling algorithms will be used to generate the training data, where SMOTE [7] and ADASYN [11] are the most well-known algorithms. Actually, Riaz et al. [22] improved the software defect prediction by combining the machine learning algorithm with the oversampling algorithm. We will also study the efficiency by combining the fine-tuning of the hyperparameters with the oversampling algorithm in the forthcoming article.

References

[1] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, Springer, 2023.

[2] Q. Al-Qasem, M. Akour, and M. Alenezi, "The influence of deep learning algorithms factors in software fault prediction", *IEEE Access*, vol. 8. pp. 63945–63960, 2020.

[3] E. N. Akimova et al., "A survey on software defect prediction using deep learning", *MDPI Mathematics*, vol. 9, article no. p. e1180, 2021.

[4] H. Alsghaier and M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier", *Software: Practice and Experience*, vol. 50, no. 4, pp. 407–427, 2020.

[5] L. Breiman, "Bagging predictors", *Machine Learning*, vol. 24, pp. 123–140, 1996.

[6] L. Breiman, "Random forests", *Machine Learning*, vol. 45, pp. 5–32, 2001.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique", *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[8] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2016)*, pp. 785–794, ACM ICPS, 2016.

[9] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation", *BMC Genomics*, vol. 21, no. 6, pp. 1-–13, 2020.

[10] T. Dohi, J. Wu, and H. Okamura, "Software bug prediction based on semi-definite logistic regression model", *Proceedings of The 10th International Conference on Advances and Trends in Software Engineering (SOFTENG 2024)*, pp. 11–16, IARIA Press, 2024.

[11] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning", *Proceedings of 2008 IEEE International Joint Conference on Neural Networks*, pp. 1322–1328, IEEE CPS, 2008.

[12] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[13] D. Karaboga," An idea based on honey bee swarm for numerical optimization", *Technical Report-TR06*, Department of Computer Engineering, Engineering Faculty, Erciyes University, 2005.

[14] F. Khan, S. Kanwal, S. Alamri, and B. Mumtaz, "Hyper-parameter optimization of classifiers, using an artificial immune network and its application to software bug prediction", *IEEE Access*, vol. 8, pp. 20954–20964, 2020.

[15] H. Kondo and Y. Asanuma, "Optimization of hyperparameters and feature selection for random forests and support vector machines by artificial bee colony algorithm (in Japanese)", *Journal of the Japanese Society for Artificial Intelligence*, vol. 34, no. 2, pp. G-136: 1–11, 2019.

[16] L. Kopitar, P. Kocbek, L. Cilar, A. Sheikh, and G. Stiglic, "Early detection of type 2 diabetes mellitus using machine learning-based prediction models", *Scientific Reports*, vol. 10, p. e11981, 2020.

[17] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features", *Information and Software Technology*, vol. 58, pp. 388—402, 2015.

[18] L. Lavazza and S. Morasca, "Comparing ϕ and the F-measure as performance metrics for software-related classifications, *Empirical Software Engineering*, vol. 27, no. 185, 2022.

[19] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code", *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[20] R. Malhotra and M. Khanna, "Particle swarm optimization-based ensemble learning for software change prediction", *Information and Software Technology*, vol. 102, pp. 65–84, 2018.

[21] P. Probst, M. N. Wright, and A.-L. Boulesteix, "Hyperparameters and tuning strategies for random forest", *WIREs Data Mining and Knowledge Discovery*, vol. 9, no. 3, p. e1301 (2019).

[22] S. Riaz, A. Arshad, and L. Jiao, "Rough noise-filtered easy ensemble for software fault prediction", *IEEE Access*, vol. 6, pp. 46886–46899, 2018.

[23] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked demising auto encoders and two-stage ensemble learning", *Information and Software Technology*, vol. 96, pp. 94–111, 2018.

[24] N. S. Thomas and S. Kaliraj, "An improved and optimized random forest based approach to predict the software faults", *SN Computer Science*, vol. 5, p. e530, 2024.

[25] H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction", *Expert Systems with Applications*, vol. 122, pp. 227–42, 2019.

[26] Z. Xu et al., "LDFR: Learning deep feature representation for software defect prediction", *Journal of Systems and Software*, vol. 158, p. e11040, 2019.

[27] J. Yao and M. J. Shepperd, "The impact of using biased performance metrics on software defect prediction research", *Infomation and Software Technology*, vol. 139, p. e106664, 2021.

[28] N. Zhang, S. Ying, K. Zhu, and D. Zhu, "Software defect prediction based on stacked sparse denoising autoencoders and enhanced extreme learning machine", *IET Software*, pp. 1–19, 2021.

[29] Z.-H. Zhou, *Ensemble Methods: Foundation and Algorithm*, CRC Press, 2012.