IJCCS (Indonesian Journal of Computing and Cybernetics Systems)
Vol.18, No.1, January 2024, pp. 95~106
ISSN (print): 1978-1520, ISSN (online): 2460-7258
DOI: https://doi.org/10.22146/ijccs.90437

Effect of Hyperparameter Tuning Using Random Search on Tree-Based Classification Algorithm for Software Defect Prediction

Muhammad Hevny Rizky[1], Mohammad Reza Faisal*[2], Irwan Budiman[3], Dwi Kartini[4], Friska Abadi[5]
[1,2,3,4,5]Department of Computer Science, Faculty of Mathematics and Natural Science, Lambung Mangkurat University, Banjarbaru, Indonesia
e-mail: [1]hevnyrizky12@gmail.com, ***[2]reza.faisal@ulm.ac.id**, [3]irwan.budiman@ulm.ac.id, [4]dwikartini@ulm.ac.id, [5]friska.abadi@ulm.ac.id

Abstrak

Bidang teknologi informasi sekarang membutuhkan perangkat lunak. Namun, perangkat lunak yang bermasalah masih menimbulkan masalah besar. Untuk meningkatkan kualitas dan keandalan perangkat lunak, prediksi kerusakan menjadi penting. Dalam bidang ini, algoritma berbasis pohon seperti Random Forest, Deep Forest, dan Decision Tree menunjukkan banyak prospek. Konfigurasi hyperparameter yang tepat sangat penting untuk mencapai hasil yang optimal. Dalam penelitian ini, Teknik Pengaturan Hyperparameter Random Search ditunjukkan sebagai metode inovatif untuk memprediksi cacat perangkat lunak. Metodologi ini memungkinkan penyelidikan parameter algoritma secara efektif, yang pada akhirnya akan meningkatkan keakuratan perkiraan kerusakan perangkat lunak. Kami melakukan penelitian ini dengan menggunakan kumpulan data ReLink yaitu, Apache, Safe, dan Zxing. Ini memungkinkan kami untuk menemukan parameter algoritma berbasis pohon yang paling efektif untuk prediksi kerusakan perangkat lunak. Hasilnya menunjukkan bahwa Decision Tree, Random Forest, dan Deep Forest masing-masing memiliki rata-rata AUC 0.73,0.79 dan 0.79 untuk pengaturan hyperparameter menggunakan Pencarian Acak. Hasilnya menunjukkan bahwa metode yang menggunakan penyetelan hyperparameter dengan Pencarian Acak mengungguli algoritma berbasis pohon lainnya. Khususnya, dalam kasus Random Forest, kontribusi utama penelitian ini terletak pada pendekatan metode inovatif yang menggunakan teknik penyetelan hyperparameter Pencarian Acak, yang mencakup pencarian parameter ekstensif. Penemuan kami secara signifikan menekankan keunggulan Pencarian Acak dibandingkan dengan algoritma berbasis pohon lainnya.

Kata kunci*—Prediksi Cacat Software, Hyperparameter Tuning, Decision Tree, Random Forest, Deep Forest.*

Abstract

The field of information technology requires software, which has significant issues. Quality and reliability improvement needs damage prediction. Tree-based algorithms like Random Forest, Deep Forest, and Decision Tree offer potential in this domain. However, proper hyperparameter configuration is crucial for optimal outcomes. This study demonstrates using the Random Search Hyperparameter Setting Technique to predict software defects, improving damage estimation accuracy. Using ReLink datasets, we found effective algorithm parameters for predicting software damage. Decision Tree, Random Forest, and Deep Forest achieved an average AUC of 0.73, 0.79, and 0.79 with Random Search. Random Search outperformed other tree-based algorithms. The main contribution is the innovative Random Search hyperparameter tuning, mainly for Random Forest. Random Search has distinct advantages over other tree-based algorithms.

Keywords—*Software Defect Prediction, Hyperparameter Tuning, Decision Tree, Random Forest, Deep Forest.*

Received November 6th,2023; Revised December 13th, 2023; Accepted December 22th, 2023

1. INTRODUCTION

The expanding influence and complexity of software are being observed across diverse domains of our lives. Software becomes more intricate, the process of rectifying failures or defects becomes increasingly challenging [1]. It is possible to anticipate software failures by forecasting software defects in the preliminary stages of software development, as rectifying them in the later stages would be more costly and challenging to identify. Software defects denote flaws, inaccuracies, or malfunctions within a computer system or program, which may lead to unexpected or erroneous outcomes, thereby impeding the intended functionality of the software and causing a decline in its quality. This deterioration in software quality in itself poses a disadvantage. To ensure the attainment of high-quality software, the final product must exhibit minimal defects. Early identification of software defects can curtail development expenses and rework efforts and yield more dependable software [2]. Predicting software defects is of utmost significance to address software issues while enhancing software quality [3]. The prediction of software defects involves scrutinizing software metrics and subsequently constructing models for defect prognostication. Defect defects in software modules are identified through classification, a method commonly employed by numerous studies [4]. Using metrics to predict software damage is pivotal in developing prognostic models to enhance software quality by forecasting the maximum number of software breakdowns.

In Andini et al.'s research using tree-based classification with Grid Search hyperparameter tuning, the average AUC value obtained was 0.69. Random Forest generated an average AUC value of 0.76, whereas Deep Forest produced an average AUC value of 0.79 [5]. In the second research, Afrizal et al. adopted the technique of hyperparameter tuning using Random Search to increase the performance of software defect prediction as the selection of hyperparameters. According to the study's results, hyperparameter tuning by Random Search was useful for the tuning parameter search issue. As a result, without hyperparameter change, the XGBoost classification obtained an accuracy of 95.34%, a recall of 93.78%, and a precision of 95.63%. With hyperparameter tuning, XGBoost classification achieved an accuracy of 95.34%, recall of 95.63%, and precision of 98.44%. Using Random Search in XGBoost for hyperparameter tuning resulted in an estimated 2.35% improvement in accuracy, 2.55% rise in recall, and 2.81% increase in precision [6]. In another study, Zhou et al. offered numerous approaches for software defect prediction, including Random Forest (RF), Naive Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR), and Deep Belief Networks (DBN). NASA, PROMISE, AEEEM, and ReLink databases were all used. Based on the comparative findings, it was discovered that DPDF performed the best for the NASA dataset, with the AUC increasing and reaching the maximum value of 92%. DPDF also outperformed the others in the PROMISE and AEEEM datasets, delivering score gains of 89% and 86%, respectively. However, in several ReLink datasets, DPDF did not outperform RF and DBN, with a maximum score of 75% [7].

Based on previous researchers' exposure to improved performance, this study will employ the utilization of hyperparameters in order to make predictions regarding software defects. This research will be achieved by implementing Random Search techniques for tree-based classifications, specifically decision trees, random forests, and deep forests.

2. METHODS

This research method provides a detailed account of the datasets that were utilized in this study. It also explains the preprocessing techniques employed to prepare the data for analysis. It also dives into the categorization algorithms used, Decision Tree, Random Forest, and Deep Forest are a few examples. Additionally, it elucidates the validation test conducted using cross-validation, which ensured the results' robustness. Moreover, it highlights the hyperparameter search that was conducted using Random Search, which aimed to optimize the performance of the classification algorithms. Lastly, it discusses the performance measurement employed, namely the AUC evaluation method. Moving on to the research procedures that will be carried out, Figure 1 depicts the flow of this study and serves as a reference for the remaining stages that will be conducted.

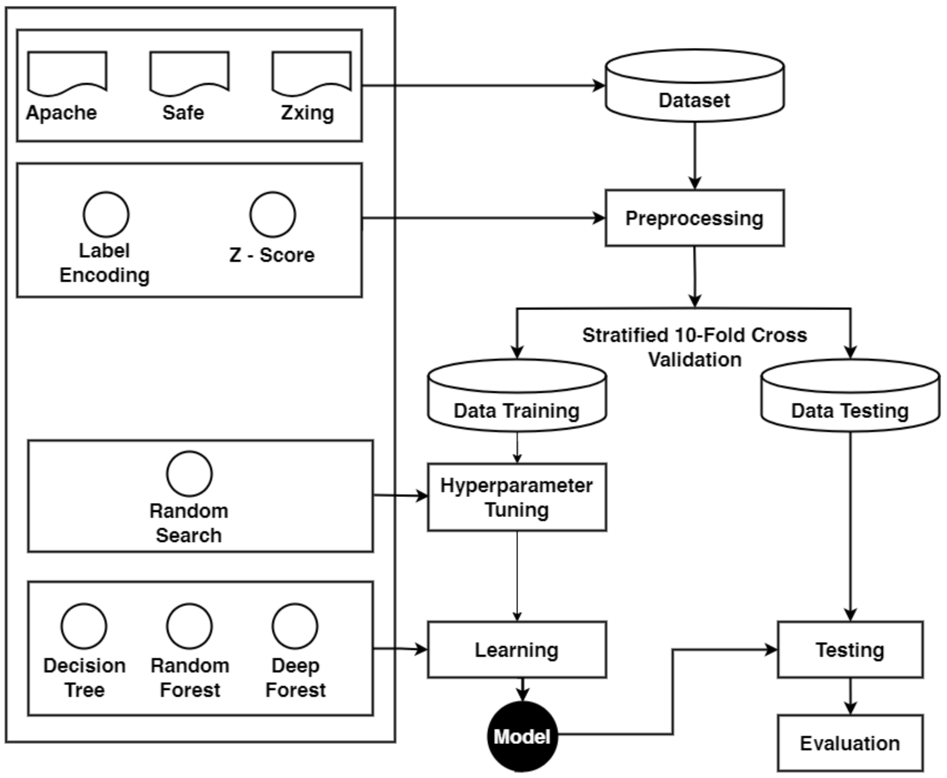

Figure 1 Research flow

The first stage of this research was to assemble the ReLink datasets, which included Apache, Safe, and Zxing. The dataset was then preprocessed using label encoding and z-score normalization. Following that, data exchange was achieved by cross-validation. This research used 10-fold cross-validation as its validation approach. Each ReLink dataset was separated into ten pieces, nine of which were allocated for training and the remaining portions for testing. Following that, an ideal tuning hyperparameter search was performed using Random Search prior to the learning phase. This study's learning phase included three distinct scenarios: classification using a Decision Tree, classification using a Random Forest, and classification using a Deep Forest. The mean AUC ratings were used to evaluate this research.

2.1 Data Collection

The study employed a software metric dataset called ReLink, composed of Apache, Safe, and Zxing data. This dataset is accessible for download at the subsequent hyperlink: https://github.com/bharlow058/AEEEM-and-other-SDP-datasets/tree/master/dataset/Relink.

Table 1 shows the broad range of data volumes within each ReLink dataset, notably Apache with 194 modules, Safe with 56 modules, and Zxing with 399 modules. Furthermore, it provides a complete summary of the software metrics covered by the ReLink data set, divided into two unique categories: Complexity Metrics (CPM) and Counts Metrics (CTM). It is worth noting that it displays an equal amount of software metrics from the ReLink dataset.

Table 1 Number of ReLink Software Metrics

Matrix Category	Attribute	Relink Datasets		
		Apache	Safe	Zxing
CPM	"AvgCyclomatic"	✓	✓	✓
	“AvgCyclomaticModified”	✓	✓	✓
	“AvgCyclomaticStrict”	✓	✓	✓
	“AvgEssential”	✓	✓	✓
	“MaxCyclomatic”	✓	✓	✓
	“MaxCyclomaticModified”	✓	✓	✓
	“MaxCyclomaticStrict”	✓	✓	✓
	“RatioCommentToCode”	✓	✓	✓
	“SumCyclomatic”	✓	✓	✓
	“SumCyclomaticModified”	✓	✓	✓
	“SumCyclomaticStrict”	✓	✓	✓
	“SumEssential”	✓	✓	✓
CTM	“AvgLine”	✓	✓	✓
	“AvgLineBlank”	✓	✓	✓
	“AvgLineCode”	✓	✓	✓
	“AvgLineComment”	✓	✓	✓
	“CountLine”	✓	✓	✓
	“CountLineBlank”	✓	✓	✓
	“CountLineCode”	✓	✓	✓
	“CountLineCodeDecl”	✓	✓	✓
	“CountLineCodeExe”	✓	✓	✓
	“CountLineComment”	✓	✓	✓
	“CountSemicolon”	✓	✓	✓
	“CountStmt”	✓	✓	✓
	“CountStmtDecl”	✓	✓	✓
	“CountStmtExe”	✓	✓	✓
	Module	194	56	399
	Attribute (Metrics)	26	26	26
	Number of bug classes	98	22	118
	Number of clean classes	96	34	281

2.2 Preprocessing

Prior to the execution of data sharing, the data shall be subjected to adjustments in order to cater to the requirements of the algorithm. The process of preprocessing serves the purpose of customizing the data to suit classification algorithms, which can enhance the performance outcomes of classification models [5]. The preprocessing of the data employed in this study encompasses label encoding and normalization. Label Encoding refers to transforming label values into numeric representation [8]. For each model utilized to handle class labels effectively, it is imperative to substitute the label values within the ReLink dataset with their corresponding numeric equivalents. Normalization is a crucial procedure employed during the preprocessing stage whereby numerical attribute data is decomposed, enabling the conversion of values within the data into a specific range [9]. It serves as a technique for effectively mapping data across various scales. Among the diverse array of normalization techniques available, the z-score method stands out. Equation (1) visually represents the z-score normalization technique [5].

$$Z = \frac{X - mean}{std} \tag{1}$$

where X represents the value that has been observed, referred to as the original data. Mean denotes the average value, while Std signifies the standard deviation value. The corresponding z-score will adopt a positive value when the value exceeds the average. Conversely, if the value falls below the average, the z-score will assume a negative value. The ReLink dataset encompasses a range that varies for each feature. Consequently, it becomes imperative to perform a normalization process.

2.3 Cross Validation

The introduction of cross-validation reduces overfitting in random sampling on data sets. Cross-validation, a popular machine learning approach, divides the original dataset into training and test data. This section aims to ensure successful training and reliable assessment of classification models using training and test data. Data is divided into K subsets, commonly known as validation sets, with K set to 10 by default. K Fold Cross Validation is a cyclic procedure repeated on each validation subset. This procedure ensures that each subset is used only once as the validation set, with the remaining subsets used as training data. The drawback of K Fold Cross Validation is that data sharing is not proportionate, leading to potential data loss, especially when imbalanced data is used [10]. In order to prevent errors from imbalanced data, we utilize stratified K fold (SKF) cross-validation (CV) to distribute the data evenly. Stratified 10 Fold cross-validation ensures that the distribution of data samples is equal across classes and that all instances are tested [11].

2.4 Learning and Hyperparameter Tuning

The Decision Tree (DT) is a widely used and successful approach that finds applications in several disciplines, including machine learning, classification, image processing, and pattern detection. The model's output is determined by sequentially traversing a tree structure consisting of decision nodes [12]. The primary objective of the DT is to construct a training model capable of making predictions regarding the value or variable of a target based on decision rules derived from training data. As its name suggests, the Decision Tree is portrayed as a hierarchical structure consisting of three distinct types of nodes. Embarking from a root node, which symbolizes the initial point of the tree's formation n, the internal nodes serve as pivotal points for branching. In contrast, the leaf node, representing the ultimate node, pertains to the class label. The classification process is executed by segregating the tree branches, whereby each division corresponds to a test conducted on a specific attribute. This branching process persists until the terminal level is attained, wherein the data tuples of each node solely comprise samples belonging to a singular class. The algorithm concludes the partitioning process once the training tuples are exclusively assigned to a single class. Finally, the leaf nodes furnish predictions for the class. The configuration of the Decision Tree model is visually depicted in Figure 2.

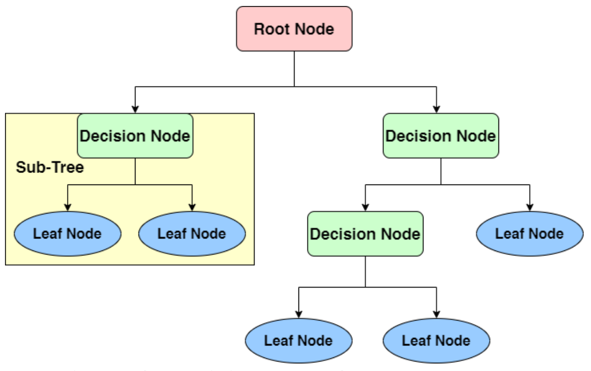

Figure 2 Decision Tree Structure

Random Forest (RF) is an extension of the Decision Tree method. The fundamental concept behind Random Forest is to derive conclusions by using a sequence of determinations represented in the structure of a Decision Tree. This technique is a collection of learning approaches that use the Decision Tree as a created and merged basis classifier. Staying true to its name, Random Forest constructs a forest comprising multiple trees. Within the Random Forest framework, individual DT are responsible for generating predictions for each distinct class. In order to mitigate correlations between DT, RF introduces random feature projection during the growth of each tree. This technique of random feature projection significantly diminishes inter-tree correlations since different trees develop on different sets of features. By adopting a mean prediction approach, RF enhances the Decision Tree algorithm by amalgamating the output of multiple DT to yield a predicted outcome. The model for RF is illustrated in Figure 3.

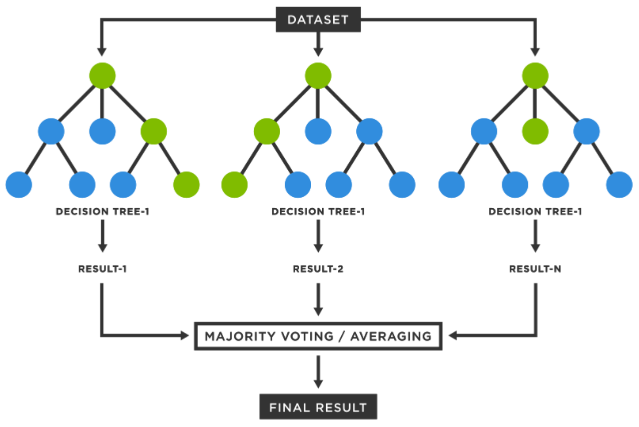

Figure 3 Random Forest Structure

Deep Forest (DF) is commonly known as an alternative Deep Neural Network (DNN). Deep forests consist of layer-by-layer structures known as cascade forests. The structure of each layer in the cascade forest resembles the backpropagation of DNNs, with the distinction that it contains multiple Random Forests instead of neurons [3]. Cascade forest refers to a class distribution each tree generates for every instance. These distributions are calculated by employing the ratios of the various classes within the instances.

Consequently, a class vector is obtained from the average distribution of classes across all the trees and forests. The deep forest algorithm follows a layered and overlaid flow process at each layer level. The first layer receives input from attributes or features in the original dataset, which are then processed alongside Random Forest in the subsequent layer. The layer terminates either when the process generated by Random Forest no longer improves or when the result on the given layer decreases. From each existing layer level, the algorithm evenly distributes the results from layer to layer until the last layer is reached and the maximum value is obtained. Despite taking longer than Random Forest, Deep Forest performs better when dealing with small-scale data [13]. The depiction of the cascade forest's groove can be observed in Figure 4.

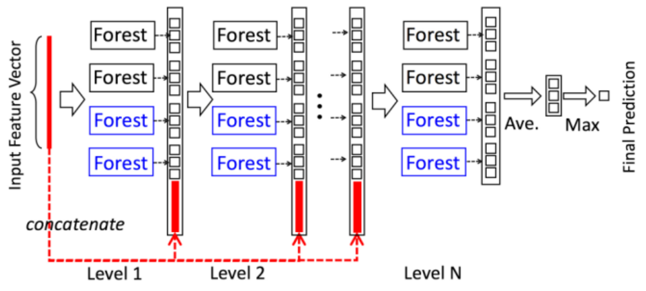

Figure 4 Deep Forest Structure

Hyperparameter tuning refers to searching for the most optimal values for a set of parameters, wherein one must initially specify a list of parameters and their corresponding search ranges [14]. When selecting a tuning strategy, it becomes necessary to identify the list of parameters and their respective search ranges while also considering the option of utilizing default values. In the context of DT, RF, and DF, these models possess sets of hyperparameters that can be configured. Specifically, in the case of the Decision Tree, the parameter min_samples_leaf determines the minimum number of samples required at a leaf node. Another parameter, min_samples_split, controls the minimum number of samples to split internal vertices.

Furthermore, the parameter max_depth determines the depth of the tree, while min_impurity_decrease plays a role in regulating the growth of the tree based on impurity, which is assessed through metrics such as the Gini index and entropy [15]. Like the Decision Tree algorithm, RF also has a parameter called max_depth that regulates the depth of the trees within the forest [16]. Additionally, RF and DF have a parameter called n_estimators, which governs the number of trees in the forest. However, it is crucial to acknowledge a distinction in the role of n_estimators between DF and RF. Within the context of DF, this particular parameter governs the abundance of forest in each layer. In contrast, in the case of RF, it dictates the number of trees within each forest [8].

The process of Random Search commences with the random selection of hyper-parameter pairs, which are subsequently used to train the model. Following this, the training results are recorded, and validation is conducted. These steps are repeated numerous times to generate multiple potential candidates. The validation scores of all the obtained candidates are then compared, and the highest scores are selected. This comparative analysis yields the optimal configuration of parameters. Figure 5 graphically depicts the steps of Random Search.

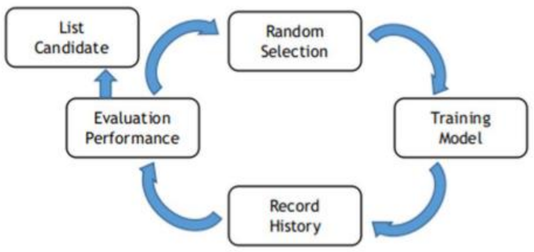

Figure 5 Stages of a Random Search

2.5 Evaluation

The classification performance of DT, RF, and DF models on each ReLink dataset is evaluated using AUC values. The selection of AUC as the evaluation method is based on its appropriateness for assessing the predictive performance of datasets with imbalanced class issues [5]. Table 2 provides comprehensive guidelines for the classification of AUC values.

Table 2 AUC Value Category [5]

Category	Range		
Excellent	0.90	To	1.00
Good	0.80	To	0.90
Pretty good	0.70	To	0.80
Not good	0.60	To	0.70
Failure	Less than <0.60		

3. RESULTS AND DISCUSSION

This section presents the research findings achieved through the stages of encoding labels. The ReLink dataset contains a module consisting of 649 label classes that have not undergone encoding based on the requirements of the model. Therefore, it is necessary to convert the label values in the ReLink dataset into binary numbers, specifically 0 for clean and 1 for defective. Following the label encoding stage, the normalization process is conducted using z-score, as outlined in Equation (1). In this process, if a value exceeds the average, the z-score is positive, while if it falls below the average, the z-score is negative. The data that has undergone normalization can be observed in Table 3, which displays the results of z-score normalization.

Table 3 Z-score results

AvgCyclomatic	AvgCyclomaticModified	…	SumCyclomaticStrict	SumEssential	Label
1.105313	1.292224	…	0.993067	0.471288	1
-0.964815	-1.118271	…	-0.690226	-0.660408	0
…	…	…	…	…	…
-0.102261	-0.022591		0.876742	0.583060	0
0.07024	0.196545	…	-0.183870	-0.367005	1

The ReLink dataset was subsequently divided using cross-validation techniques, specifically employing data-sharing training and testing with stratified 10-fold cross-validation rules. Following this, a learning (classification) process was carried out. Classification was performed at this stage utilizing the Decision Tree, Random Forest, Deep Forest algorithms, and Random Search with hyperparameter tuning. The classification procedure encompassed the entire ReLink dataset, consisting of the Apache, Safe, and Zxing subsets.

A Random Search was conducted for 30 iterations, producing 30 optimal hyperparameter candidates. The parameter range for the model is provided in Table 4.

Table 4 Hyperparameter Search Ranges

Model	Parameter	Description	Range	
DT	“min_impurity_decrease”	Impurity	0	2
	“min_samples_split”	The minimum number of samples required to create an	2	20
	“max_depth”	Maximum depth of tree	1	20
	“min_samples_leaf”	The minimum number of samples required to be at a leaf	1	10
RF	“n_estimators”	Number of Individual trees	100	2000
	“max_depth”’	Maximum depth of an individual tree	10	100
DF	“n_estimators”	Number of forests in each layer	3	11
	“n_trees”	Number of trees in each forest	50	1000
	“max_depth”	Maximum depth of an individual tree	10	100

The parameter search produces optimal parameters using Random Search, which gives the best prediction performance results showing the AUC value produced from each model. The results of evaluating optimal parameter values and AUC are shown in Table 5.

Table 5 Apache Dataset AUC Result

Model	Parameter Optimal	AUC
DT	"min_sample_split: 18, min_sample_leaf: 10, min_impunity_decrease: 0, max_depth: 1"	0.74
RF	"n_estimators: 500, max_depth: 80"	**0.79**
DF	"max_depth': 30 n_trees: 200, 30 n_estimators: 7"	0.78

The evaluation results on the Safe dataset produce optimal parameters and AUC, as seen in Table 6.

Table 6 Dataset Safe AUC Results

Model	Parameter optimal	AUC
DT	"min_impunity_decrease: 0, min_sample_split: 8, max_depth: 13, min_sample_leaf: 5"	0.83
RF	"n_estimators: 100, max_depth: 40"	**0.86**
DF	"max_depth: 10 n_estimators: 3, n_trees: 250"	**0.86**

The evaluation results on the Zxing dataset produce optimal parameters and AUC, as seen in Table 7.

Table 7 Zxing Dataset AUC Results

Model	Parameter optimal	AUC
DT	"min_sample_split: 8, min_sample_leaf: 5, max_depth: 13, min_impunity_decrease: 0"	0.63
RF	"max_depth: 40, n_estimators: 100"	**0.74**
DF	"max_depth: 10 n_estimators: 3, n_trees: 250"	0.73

Table 8 illustrates the mean AUC outcomes derived from hyperparameter tuning via the Random Search approach across the complete range of ReLink datasets.

Table 8 Performance Results of All Datasets

Dataset	DT	RF	DF
Apache	0.74	0.79	0.78
Safe	0.83	0.86	0.86
Zxing	0.63	0.74	0.73
Average	0.73	**0.79**	**0.79**

The Random Forest model outperforms the Decision Tree model regarding average AUC values on the ReLink dataset. Deep Forest achieved an overall AUC value of 0.79 in the study. The performance on Safe data was particularly impressive, with a result of 0.86. Similarly, Random Forest had an average AUC score of 0.79, with its highest performance on Safe data at 0.86. In contrast, the Decision Tree model performed poorly compared to the other models. It achieved an average AUC value of 0.73 for the generated dataset, with its highest performance on Safe data reaching 0.83. The overall performance of the average AUC values provided by Random Forest and Deep Forest is superior.

The comparison of the outcomes obtained from the previous studies indicated that the method of hyperparameter tuning Random Search, which was applied to the entire tree-based algorithm, successfully enhanced the prediction of software defects compared to the previous studies. This improvement was discovered by comparing the average performance of each suggested approach to that of the prior study method. Compared to the decision tree method with grid search tuning, Random Search revealed a 4% improvement. Furthermore, the suggested random forest approach, which combines hyperparameter tuning RS with hyperparameter tuning Grid Search (GS), outperformed grid search hyperparameter tuning by 3%. The deep forest approach used Random Search hyperparameter tweaking and produced the same AUC value of 0.79 as earlier researchers who used previously investigated hyperparameters.

In a previous study, hyperparameters were adjusted using the grid search method. The leaf node parameters were set from 1 to 10, internal nodes from 2 to 10, tree depth from 1 to 10, and impurity from 0 to 3. The decision tree algorithm classification resulted in an average AUC value of 0.69. Furthermore, the classification using the random forest algorithm with grid search tuning had the parameters set from 100 to 500 for the number of trees and from 1 to 5 for the tree depth, resulting in an average AUC value of 0.76. Lastly, the deep forest classification with grid search tuning had the parameters set from 2 to 11 for the number of forests and from 100 to 2000 for the number of trees, yielding an average AUC value of 0.79.

In this study, the hyperparameter tuning was performed using a Random Search with the parameters outlined in Table 4 for the decision tree classification algorithm, resulting in an average AUC value of 0.73. For the random forest algorithm, the average AUC value obtained was 0.79. Similarly, the deep forest algorithm yielded an average AUC value of 0.79. A comparison of the AUC values is presented in Table 9.

Table 9 Comparison of AUC Values

Dataset	Previous research methods [5]			Proposed research method		
	DT	RF	DF	DT	RF	DF
Apache	0.76	0.76	0.81	0.74	0.79	0.78
Safe	0.68	0.87	0.85	0.83	0.86	0.86
Zxing	0.63	0.64	0.71	0.63	0.74	0.73
Average	0.69	0.76	0.79	**0.73**	**0.79**	**0.79**

Table 10 presents a comprehensive analysis of the overall outcomes compared to previous research endeavours that employed various methodologies, including LR, DF, SVM, NB, and RF. It becomes apparent that the findings derived from this investigation surpass those of its precursors. More specifically, the average AUC value achieved through utilising DT, RF, and DF classification techniques, employing the Random Search hyperparameter tuning approach, outperforms the average AUC value obtained by applying alternative methodologies.

Table 10 Comparison of AUC Results with Other Research Methods

Dataset	Previous research methods[7]					Proposed research method		
	NB	LR	SVM	RF	DF	DT	RF	DF
Apache	0.74	0.70	0.76	0.76	0.75	0.74	0.79	0.78
Safe	0.69	0.67	0.69	0.73	0.73	0.83	0.86	0.86
Zxing	0.61	0.57	0.66	0.67	0.70	0.63	0.74	0.73
Average	0.68	0.64	0.70	0.72	0.73	**0.73**	**0.79**	**0.79**

4. CONCLUSIONS

This particular study focuses on predicting software defects in ReLink datasets. The prediction is done using tree-based categorization models, namely DT, RF, and DF. The models are further enhanced by conducting hyperparameter tuning using the Random Search technique. The performance of these models is found to vary, as indicated by the results obtained from various trials and comparisons. Applying hyperparameter tuning using Random Search improves the AUC metric's performance significantly. This improvement is particularly noteworthy compared to previous studies that utilized hyperparameter tuning with Grid Search for tree-based classification.

Interestingly, the Random Search approach outperforms other studies that employed NB, LR, and SVM with default parameter configurations. Regarding RF and DF, the RF parameters mainly focus on the number of trees within a specified range, typically from 100 to 1000. On the other hand, DF introduces an additional parameter, namely the number of forests, with possible values of 3, 4, and 5. In this study, the average AUC value achieved by the RF model was 0.72, while the DF model performed slightly better with an average AUC of 0.73.

The results of this research clearly show that using hyperparameter tweaking with Random Search in combination with the RF classification model produces improved results. This superiority is evident from the obtained average AUC value of 0.79. This result further solidifies the effectiveness of the Random Search technique in optimizing the parameter search process compared to the commonly used Grid Search approach in tree-based classification.

In future investigations, there is a possibility of incorporating a parameter candidate search range into the Random Search parameter search algorithm by expanding the parameter candidate range. The primary objective of this endeavour is to ascertain the degree to which the performance of Random Search can be optimized in terms of parameter search, thereby yielding superior performance values. By expanding the range within which potential parameter candidates are considered, researchers can delve into the realm of possibility and explore the potential benefits and drawbacks of this novel approach.

ACKNOWLEDGEMENTS

This article results from a final assignment in the Department of Computer Science, Faculty of Mathematics and Natural Sciences, Lambung Mangkurat University, Banjarbaru, Indonesia.

REFERENCES

[1] A. Elmishali and M. Kalech, “Issues-Driven features for software fault prediction,” *Information and Software Technology*, vol. 155, 2023, doi: 10.1016/j.infsof.2022.107102.

[2] M. K. Thota, F. H. Shajin, and P. Rajesh, “Survey on software defect prediction techniques,” *International Journal of Applied Science and Engineering*, vol. 17, no. 4, pp. 331–344, 2020, doi: 10.6703/IJASE.202012_17(4).331.

[3] W. Zheng, S. Mo, X. Jin, Y. Qu, Z. Xie, and J. Shuai, “Software defect prediction model based on improved deep forest and AutoEncoder by forest,” *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2019-July, no. 3, pp. 419–424, 2019, doi: 10.18293/SEKE2019-008.

[4] M. A. Mabayoje, A. O. Balogun, H. A. Jibril, J. O. Atoyebi, H. A. Mojeed, and V. E. Adeyemo, “Parameter tuning in KNN for software defect prediction: an empirical analysis,” *Jurnal Teknologi dan Sistem Komputer*, vol. 7, no. 4, pp. 121–126, 2019, doi: 10.14710/jtsiskom.7.4.2019.121-126.

[5] E. Andini, M. Reza Faisal, R. Herteno, R. Adi Nugroho, and F. Abadi, “PENINGKATAN KINERJA PREDIKSI CACAT SOFTWARE DENGAN

HYPERPARAMETER TUNING PADA ALGORITMA KLASIFIKASI DEEP FOREST," *Jurnal MNEMONIC*, vol. 5, no. 2, 2022, [Online]. Available: https://github.com/bharlow058/AEEEM-and-other-

[6] M. Ryan Afrizal, R. Adi Nugroho, D. Kartini, R. Herteno, J. Ahmad Yani Km, and K. Selatan, "XGBOOST DENGAN RANDOM SEARCH HYPER-PARAMETER TUNING UNTUK KLASIFIKASI SITUS PHISING," 2022.

[7] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Information and Software Technology*, vol. 114, no. July 2018, pp. 204–216, 2019, doi: 10.1016/j.infsof.2019.07.003.

[8] A. Javeed, S. Zhou, L. Yongjian, I. Qasim, A. Noor, and R. Nour, "An Intelligent Learning System Based on Random Search Algorithm and Optimized Random Forest Model for Improved Heart Disease Detection," *IEEE Access*, vol. 7, pp. 180235–180243, 2019, doi: 10.1109/ACCESS.2019.2952107.

[9] H. Aji Prihanditya and N. Hestu Aji Prihanditya, "The Implementation of Z-Score Normalization and Boosting Techniques to Increase Accuracy of C4.5 Algorithm in Diagnosing Chronic Kidney Disease," 2020.

[10] B. Kovalerchuk, "Enhancement of Cross Validation Using Hybrid Visual and Analytical Means with Shannon Function," *Studies in Computational Intelligence*, vol. 835, pp. 517–543, 2020, doi: 10.1007/978-3-030-31041-7_29.

[11] H. Aljamaan and A. Alazba, "Software defect prediction using tree-based ensembles," *PROMISE 2020 - Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Co-located with ESEC/FSE 2020*, pp. 1–10, 2020, doi: 10.1145/3416508.3417114.

[12] M. Joye and F. Salehi, *Private yet efficient decision tree evaluation*, vol. 10980 LNCS. Springer International Publishing, 2018. doi: 10.1007/978-3-319-95729-6_16.

[13] L. V. Utkin, "An imprecise deep forest for classification," *Expert Systems with Applications*, vol. 141, p. 112978, 2020, doi: 10.1016/j.eswa.2019.112978.

[14] H. Alibrahim and L. Simone A., "2021 IEEE Congress on Evolutionary Computation," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 5, pp. 740–740, 2020, doi: 10.1109/tetci.2020.3020707.

[15] R. G. Mantovani, T. Horváth, R. Cerri, S. B. Junior, J. Vanschoren, and A. C. P. de L. F. de Carvalho, "An empirical study on hyperparameter tuning of decision trees," no. December, 2018, [Online]. Available: http://arxiv.org/abs/1812.02207

[16] M. Daviran, A. Maghsoudi, R. Ghezelbash, and B. Pradhan, "Computers and Geosciences A new strategy for spatial predictive mapping of mineral prospectivity : Automated hyperparameter tuning of random forest approach," *Computers and Geosciences*, vol. 148, no. January, p. 104688, 2021, doi: 10.1016/j.cageo.2021.104688.