Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)
Vol. 10, No. 2, June 2024, pp. 476-488
ISSN: 2338-3070, DOI: 10.26555/jiteki.v10i2.28973

Random Search-Based Parameter Optimization on Binary Classifiers for Software Defect Prediction

Misbah Ali[1], Muhammad Sohaib Azam[2], Tariq Shahzad[3]
[1]Department of Computer Science and Information Technology, Virtual University, Lahore, Pakistan
[2]Department of Computer Science, COMSATS University Islamabad, Sahiwal Campus, Pakistan
[3]Department of Electrical and Electronic Engineering Science, University of Johannesburg, Johannesburg 2006, South Africa.

ARTICLE INFO

Article history:

Received June 04, 2024
Revised July 07, 2024
Published July 23, 2024

Keywords:

Software Defect Prediction;
Software Metrics;
Machine Learning;
Classification;
Decision Tree;
Support Vector Machine;
Naïve Bayes

ABSTRACT

Machine learning classifiers consist of a set of parameters. The efficiency of these classifiers in the context of software defect prediction is greatly impacted by the parameters chosen to execute the classifiers. These parameters can be optimized to achieve more accurate results. In this research, the efficiency of binary classifiers for software defect prediction is analyzed through parameter optimization using random search technique. Three heterogeneous binary classifiers i.e., Decision tree, Support vector machine, and Naïve Bayes are selected to examine the results of parameter optimization. The experiments were performed on seven publicly available NASA Datasets. The dataset was split into 70-30 proportions with class preservation. To evaluate the performance; five statistical measures have been implemented i.e., precision, recall, F-Measure, the area under the curve (AUC), and accuracy. The findings of the research revealed that there is significant improvement in accuracy for each classifier. On average, decision tree improved from 88.1% to 95.4%; support vector machine enhanced the accuracy from 94.3% to 99.9%. While Naïve Bayes showed an accuracy boost from 74.9% to 85.3%. This research contributes to the field of machine learning by presenting comparative analysis of accuracy improvements using default parameters and optimized parameters through random search. The results presented that he performance of binary classifiers in the context of software prediction can be enhanced to a great extent by employing parameter optimization using random search.

Corresponding Author:

Misbah Ali, Department of Computer Science and Information Technology, Virtual University, Lahore, Pakistan
Email: talktomisbah.ali@gmail.com

1. INTRODUCTION

A software defect refers to any deviation from the intended behavior of the software that diminishes its overall quality. Identifying the defects at earlier stages of the software development life cycle (SDLC) is the most crucial task as it saves a bundle of resources including time, money, and manpower [1]–[3]. In the past two decades, the primary emphasis of researchers is to explore machine learning techniques for software defect prediction [4]–[6]. There exist several defect prediction methods that can distinguish between defective and non-defective software modules effectively. The state-of-the-art methods include deep learning, ensemble learning, transfer learning, and active learning [7], [8]. Deep learning techniques mostly employ convolutional neural networks, recurrent neural networks or graph-based learning; while ensemble learning techniques integrate multiple classifiers as base models [9]–[11]. The efficiency of a system for predicting software defects is subject to several factors. The main factor affecting the performance of software defect prediction models that utilize machine learning methods; is the selection of the classification algorithms. There exist numerous

classifiers both supervised and unsupervised including random forest (RF), linear regression (LR), k-nearest neighbor (KNN), and K-means clustering etc. that have been implemented by the researchers for software defect prediction and induce noticeable results [12]. However in current research, supervised classification techniques are preferred and implemented by researchers frequently for software defect prediction [13], [14]. The selection of an appropriate classifier also greatly influences the output of the defect prediction system. Dissimilar types of classifiers can better analyze the problem and lead to unbiased results. The most commonly used heterogeneous binary classifiers are Decision Tree (DT), Support Vector Machine (SVM), and Naïve Bayes (NB) [10], [13], [15]. The computational mechanism of these three classifiers varies distinctly which can analyze the classifiers effectively.

Machine learning classifiers comprise a set of parameters that direct the behavior of the classifiers. While implementing the classifiers, the researchers can access individually configurable parameters and adjust them to yield maximum results [16]–[18]. For example, in Linear Regression (LR), the value of the *intercept* can be adjusted to get maximum performance. Similarly, in KNN; the number of nearest neighbors *K* can be optimized [19], [20]. In recent times, many studies have taken place on parameter optimization that yields remarkable output [21]. There exist several parameter optimization techniques including grid search, random search, and Bayesian optimization [22], [23]. Grid search and Bayesian search methods are computationally expensive as they perform exhaustive search and build probabilistic model respectively. So that all the possible combinations of the parameters can be tried resulting in optimal results [24]. Whereas, random search offers a simple method for parameter optimization which is computationally efficient specifically when resources are limited [23].

Mostly, the researchers rely on default parameters which exhibit low accuracy for software defect prediction frameworks. In this research, random search has been employed to analyze configurable parameters of three heterogeneous binary classifiers namely DT, SVM, and NB. This optimization technique can be generalized in several software defect prediction contexts leading to improved accuracy of defect prediction frameworks. The behavior of classifiers is examined after optimizing the parameters using multiple statistical measures. The experiments are performed on seven widely used clean versions of NASA datasets including CM1, KC3, MC1, MC2, MW1, PC2, and PC3 [25]. The performance of the classifiers is measured using frequently employed five performance measures including precision, recall, F-measure, AUC, and accuracy [26].

This research makes the following contributions:

- Access the practical efficiency of random search method for parameter optimization
- Enhance the predictive power of software defect prediction systems by optimizing binary classifiers.
- Statistical analysis of the performance exhibited by the optimized binary classifiers.

2. RELATED WORK

Software defect prediction is a process of identifying and predicting potential defects or bugs in software code before it is released to production. This can be done using various techniques such as static code analysis, machine learning, and data mining. A hybrid framework for software defect prediction was developed, utilizing feature classification with twelve NASA datasets. The study included two approaches, one utilizing feature selection and the other without, both incorporating bagging and boosting techniques using a Random Forest base classifier for improved accuracy [27]. Similarly, Balogun *et al*. conducted research on utilizing Filter Feature Ranking and 14 different Filter Subset Selection techniques, using five NASA datasets and Best First Search as an FS technique. They found that the Filter Subset Selection strategy resulted in improved prediction accuracy and that Feature Filter Ranking methods were more reliable for prediction purposes [28]. Another research was conducted to evaluate the efficiency of four different classifiers, all utilizing a back-propagation approach, in the context of software defect prediction using NASA datasets. A comparison was made between the performance of the fused artificial neural network-Bayesian regularization (ANN-BR) classifier and other classifiers, using various NASA datasets. The results showed that the ANN-BR classifier performed exceptionally well [29]. Iqbal *et al*. conducted research by using twelve NASA datasets, where three classifiers were implemented and compared. The classifiers used in the study were Decision Tree (DT), Naive Bayes (NB), and K-Nearest Neighbor (KNN), and results were analyzed using various performance measures. The study found that the Naive Bayes classifier had an accuracy of 78.69%, while other classifiers such as Multilayer Perceptron, Radial Basis Function, Support Vector Machine, K-Star, OneR, PART, Decision Tree, and Random Forest had higher accuracy values ranging from 84.04% to 85.76% [30].

Parameter optimization is the practice of determining the optimal settings for the parameters of a classifier. This is accomplished to enhance the efficiency and stability of the classifier [31]. Tantithamthavorn *et al*. explored the application of automated techniques for adjusting the parameters of software defect

prediction models. The authors investigated the classifiers' efficiency and stability, parameter transferability, computational cost, and ranking of the different classifying methods when applying these techniques. They assessed 26 widely-utilized classification methods using 12 metrics of performance across 18 datasets derived from both proprietary and open-source systems. They found that optimization improves the performance of defect prediction models by up to forty percent and that the significance of adjusting parameters should be cautiously evaluated when selecting a technique for defect prediction studies [32]. The effects of applying hyperparameter optimization (HO) on ensemble learning algorithms for defect prediction performance were examined by Muhammed Maruf ¨Ozt¨urk [33]. A new ensemble learning algorithm called novel-Ensemble was presented and tested on 27 data sets, and compared with three alternatives. The results showed that ensemble methods featuring HO perform better than a single predictor, and the novel Ensemble yields promising results. The study also revealed that the success of HO is not dependent on the type of classifiers, but rather on the design of ensemble learners. A Hyper-Parameter Optimization technique for Classifiers in context of software defect prediction was explored by Khan *et al*. Using an Artificial Immune Network. The study found that utilizing AIN and its applications for predicting software bugs in machine learning classifiers improved performance when compared to using classifiers with their default hyper-parameters [34]. Mabayoje *et al*. examined the effect of parameter tuning on the k-NN algorithm in software defect prediction. They found that using a k value greater than 1 improved the average RMSE values. Using distance weighting also improved predictive performance by 8.82% and 1.7% based on AUC and accuracy respectively. Overall, parameter tuning was found to have a progressive impact on the predictive performance of k-NN in SDP [19]. Kang *et al*. discussed software quality assurance (SQA) in ships, where the safety-critical nature of ships makes SQA a fundamental prerequisite. The study presents a method called Just-in-time software defect prediction (JIT-SDP), which conducts software defect prediction on commit-level code changes to reach efficient SQA resource distribution. The study found that by applying an optimization algorithm called Harmony search (HS) to JIT-SDP, the prediction performance can be improved [31]. Smithson *et al*. analyzed the performance of Neural Networks based upon Hyper-Parameter Optimization and revealed that Hyper-parameters, such as the number of hidden layers and nodes per layer, play a crucial role in the performance and computational complexity of such models. Traditionally, these hyper-parameters have been optimized manually. To address this, a multi-objective design space exploration method was presented that reduced the number of solution networks trained and evaluated through response surface modeling [35]. The detection of code smells was examined by Shen *et al*. The researcher used hyper-parameter optimization techniques to enhance the efficiency of these methods specifically looking at two types of code smells (Data Class and Feature Envy) and using four different optimizers and six commonly used classifiers. The study finds that the use of hyper-parameter optimization considerably advances the efficiency of code smell detection and that the Differential Evolution optimizer is particularly effective when used with the Random Forest classifier [36]. Another framework was proposed for optimizing energy consumption in the distillation process using parameter optimization techniques [21]. The predictive framework consisted of three steps: learning, validation and improvement. It was witnessed that the framework performance is enhanced by optimizing its hyper-parameters. Researchers also explored the use of an alternative method, GHOST, for detecting code smells in order to reduce complexities and make code more maintainable. GHOST is a fusion of hyper-parameter optimization of feed forward neural networks and an innovative oversampling technique called "fuzzy sampling". The researchers suggested that this technique could be useful in other types of analytics as well [37].

3. MATERIALS and METHODS

This research examines the performance of binary classifiers using random search based upon default parameters and optimized parameters. It suggests the optimal parameters that contribute significantly towards the effectiveness of software defect prediction frameworks. The research consists of 3 stages: 1) Dataset selection 2) Classification through parameter optimization 3) Performance evaluation. This research has been conducted in two dimensions; first, the parameter optimization is skipped and the datasets are supplied directly to the classification algorithms. Whereas in the second dimension, the default parameters of the classifiers are first optimized using random search method and then configured within the classifiers. Hence, they yield maximum results for the selected datasets within the realm of predicting software defects or faults or bugs. The results achieved from both dimensions are compared using various statistical measures. The experiments have been performed using MATLAB a widely-used machine learning tool, which yields reliable outcomes. Fig. 1. shows a flowchart of the proposed methodology.

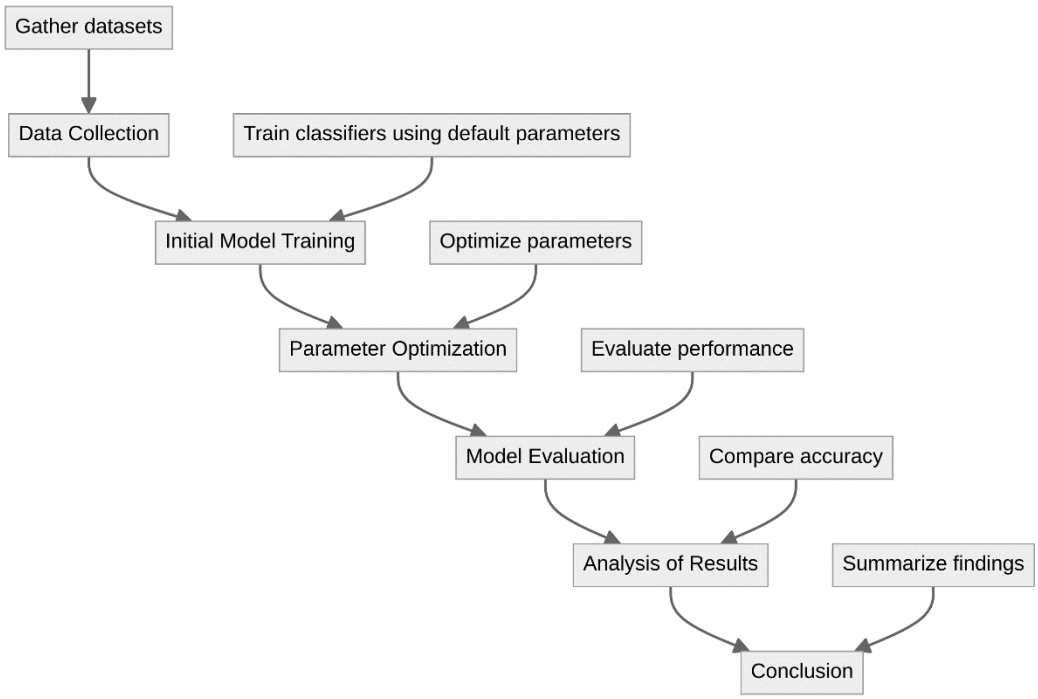

Fig. 1. Flowchart of proposed methodology [created using DALL-E]

3.1. Dataset Selection

In the first stage of this research, the dataset is selected. NASA datasets are the most widely used publicly available datasets that have been utilized by various researchers [38]. These datasets were analyzed by Shepperd *et al*. that produced two advanced versions of the datasets after passing them through a cleaning process i.e., D' and D''. D' version contains identical and inconsistent instances whereas D'' excludes all the inconsistent and duplicate instances. The criteria employed to preprocess and clean the datasets is mentioned in [39]. These datasets contain historical data of real-world software projects developed in different languages i.e. C++, Java, Perl, etc.[40]. This study has selected seven NASA datasets including CM1, KC3, MC1, MC2, MW1, PC2, and PC3. CM1 and KC3 belong to spacecraft instrument project, MC1 and MC2 present the mission control system, MW1 represents ground-based system related to zero-gravity features; and PC2, PC3 show features belonging to a science data processing system [41], [42]. The datasets comprise various quality metrics of the software in the form of independent attributes including LOCEXECUTABLE, MULTIPLECONDITIONCOUNT, and LOCTOTAL, etc. and one dependent attribute called the target attribute that tells whether a particular module in the dataset is defective or non-defective. The target attribute is predicted based on independent attributes. The value of the target attribute is either "Y" or "N"; where "Y" represents the respective module has defects and "N" represents that the respective module doesn't have defects [19], [43], [44]. In this research, a clean version of NASA datasets D'' is utilized and the performance of the binary classifiers is analyzed. Table 1 presents the datasets employed in this research.

Table 1. Details of NASA Datasets

Dataset	Modules	Attributes	Language	Defective	Non-Defective	Defective%	Description
CM1	38	327	C	42	285	12.8	Spacecraft instrument
KC3	40	194	JAVA	36	158	18.5	Spacecraft instrument
MC1	39	1952	C/C++	36	1916	1.8	Mission control
MC2	40	124	C	44	80	35.4	Mission control
MW1	38	250	C	25	225	10	ground-based
PC2	37	722	C	16	706	2.2	science data
PC3	38	1,053	C	130	923	12.3	science data

3.2. Classification using optimized parameters

In the second stage, the selection of binary classifiers is made and their parameters are optimized through random search. Parameter optimization which is also referred as hyperparameter tuning; is the process of finding the optimal combination of configurable parameters that produce maximum results. These parameters are optimized before model training and the performance is enhanced through an iterative optimization process. The integration of optimized parameters into machine learning model produces a robust and generalized model that exhibits significant performance [45].

Parameter optimization is a complex process which is greatly influenced by the number of parameters to optimize, range of parameters to be explored, the complexity of selected machine learning model, dataset size, and optimization technique. As the number of parameters, model complexity, or the dataset size increases, parameter optimization tends to become a more computationally complex process which demands more resources. In this study, random search has been selected as a parameter optimization technique due to its simplicity and less demand of resources [46].

Random search: Random search is a straightforward and efficient parameter optimization technique that explores the configurable parameters of machine learning algorithms using trial-and-error method [47]. The first step of random search involves identification of configurable parameters of a classifier. In the second step, the parameter optimization space is explored i.e., the analysis of possible values to be supplied to parameters to achieve optimal result. In the third step, a random combination of configurable parameters is sampled and the model is trained over that combination. In the fourth step, the results are analyzed using selected performance measures. in the fifth step, these results are optimized using various combinations of parameters. This process is repeated until optimal results are achieved [46], [48]. In the final step, the most suitable combination of parameters is identified and selected for classification. A step-by-step approach to execute random search is shown in Fig. 2.

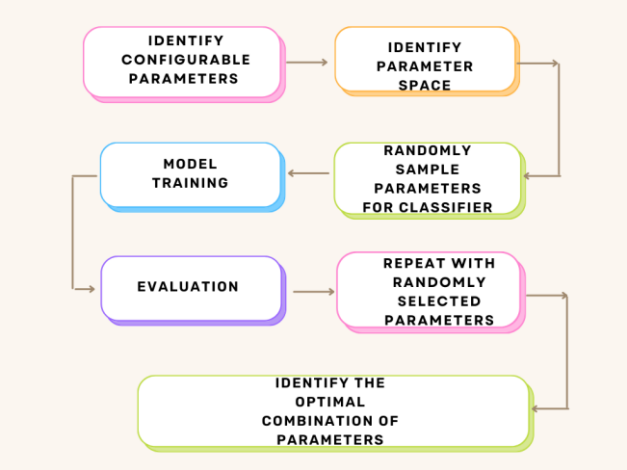

Fig. 2. step-by-step process of random search execution

Three binary classifiers of heterogeneous nature have been implemented in this research. The details of the classifiers are given as follows.

Decision Tree (DT): DT is a widely used binary classifier that follows a hierarchical structure consisting of root nodes, intermediate nodes, and terminal nodes [49]. It's a robust algorithm that runs efficiently for binary classification problems [50]. The root node holds all the instances that are to classify. Classification is accomplished by running several tests on the root and intermediate nodes. Based on the results of the tests achieved, the leaf node holds the output class label.

DT consists of several parameters including the MinLeafSize , MinParentSize , maximum number of splits, MaxDepth, and split criterion. The configurable parameters for optimized performance are identified as maximum number of splits and split criterion. DT shows the optimal performance when the "*maximum number of splits"* parameter is set as "50" and the "*split criterion"* parameter is kept as "Gini's diversity index".

Support Vector Machine (SVM): SVM makes use of support vectors and a hyperplane for classification [51]. It's an extensively used binary classifier that produces effective results with the help of a hyperplane by drawing a boundary that separates the classes. SVM has several parameters including kernel function, Box constraint level, PolynomialOrder , and kernel scale. SVM performs well when the "*kernel function"* parameter

is chosen as "Quadratic" and the "*Box constraint level"* parameter is kept as "2".

Naïve Bayes (NB): NB is a probability-driven binary classification algorithm. It executes by assuming that there doesn't exist any dependency between dataset attributes. It executes remarkably to cope with data imbalance issues [52]. It has several parameters including width, support, distribution names, and kernel type. Tuning parameters of NB include Distribution names and kernel type. NB produces optimal results when the "*Distribution names"* parameter is set as "kernel" and the "*kernel type"* is "Gaussian". The parameters detail of all the selected classifiers is shown in Table 2. Finally, the third and the last stage reflect the results achieved after a comparative analysis of default and optimized classifiers. An overview of the proposed methodology is shown in Fig. 3. The research results are described in the next section.

Table 2. Optimized parameters for binary classifiers

Classifier	Optimized parameters	Values
Decision tree	maximum number of splits	50
	split criterion	Gini's diversity index
Support vector machine	kernel function	Quadratic
	Box constraint level	2
	Distribution names	kernel
Naïve Bayes	kernel type	Gaussian

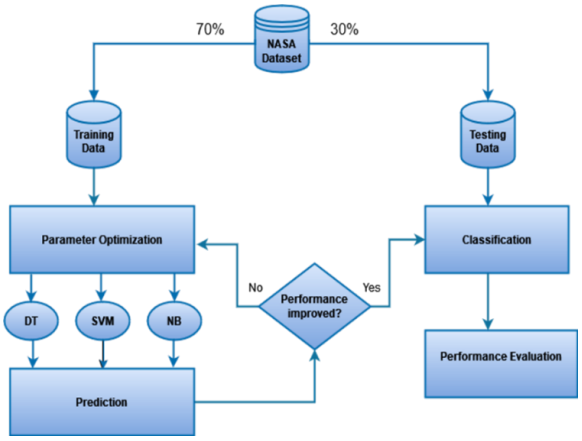

Fig. 3. Overview of the Proposed Methodology

4. RESULTS AND DISCUSSION

In this section, the findings of the experiments are examined. The proposed approach was validated a widely used 10-fold cross validation methodology which produced effective results. The performance of the optimized classifiers is evaluated through the most commonly used performance measures including precision, recall, F-measure, AUC, and accuracy [53] with the main focus on the accuracy; since it presents the overall effectiveness of the proposed approach. The research considered using these measures as they comprehensively capture the behavior of binary classifiers [54], [55]. Precision and recall present the correctness of defective modules predicted by the classifier. Whereas, f-measure shows the balance between correct and incorrect predictions. Accuracy is the most crucial measure which demonstrates the overall correct predictions (defective/non-defective modules) of a classifier. AUC differentiates between defective and non-defective modules based on the predictions of a classifier. The higher the values of all these measures, the better the performance of the classifier [54], [56]. All the measures are derived using the confusion matrix shown in Table 3.

Table 3. Confusion Matrix

	Classified Defective	Classified Non-Defective
Actual Defective	TP	FN
Actual Non-Defective	FP	TN

A confusion matrix is a technique widely implemented by researchers to evaluate the performance of classifiers [57]. The performance of both dimensions of the research is analyzed through a confusion matrix. It has four parameters as listed below:

1. True Positive (TP): This parameter holds the instances that are defective and also classifies them as defective. False Positive (FP): This parameter holds instances that are non-defective but incorrectly classified as defective.
2. False Negative (FN): This parameter holds the instances that are defective but classified as non-defective.
3. True Negative: This parameter holds instances that are non-defective and also categorized as negative.

The performance measures are derived in equation forms from the confusion matrix:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

$$\text{Recall} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{2}$$

$$\text{F} - \text{measure} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}} \tag{3}$$

$$AUC - ROC = \frac{1 + TPr - FPr}{2} \tag{4}$$

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})} \tag{5}$$

The classification was performed in two dimensions: 1) With default parameters and 2) With optimized parameters. In the first dimension, the datasets are supplied to the classifiers with default parameters; whereas in the second dimension, the datasets are supplied to the classifiers having optimized parameters. The below tables reflect the results of the research in both dimensions. Table 4 – Table 10 reflect the results of each dataset achieved by executing classifiers with optimized parameters.

Table 4. CM1 Dataset Results

Classifier	Precision	Recall	Accuracy %	F-Measure	AUC
DT	0.833	0.769	94.9	0.8	0.873
SVM	1	1	100	1	1
NB	0.471	0.615	85.9	0.533	0.755

Table 5. KC3 Dataset Results

Classifier	Precision	Recall	Accuracy %	F-Measure	AUC
DT	0.9	0.818	94.9	0.857	0.899
SVM	1	1	100	1	1
NB	0.5	0.455	81.4	0.476	0.675

Table 6. MC1 Dataset Results

Classifier	Precision	Recall	Accuracy %	F-Measure	AUC
DT	0.778	0.636	99	0.7	0.816
SVM	1	0.909	99.8	0.952	0.955
NB	0.286	0.545	96.6	0.375	0.759

Table 7. MC2 Dataset Results

Classifier	Precision	Recall	Accuracy %	F-Measure	AUC
DT	0.8	0.923	89.5	0.857	0.902
SVM	1	1	100	1	1
NB	1	0.462	81.6	0.632	0.731

Table 8. MW1 Dataset Results

Classifier	Precision	Recall	Accuracy %	F-Measure	AUC
DT	0.889	1	98.7	0.941	0.993
SVM	1	1	100	1	1
NB	0.467	0.875	88	0.609	0.878

Table 9. PC2 Dataset Results

Classifier	Precision	Recall	Accuracy %	F-Measure	AUC
DT	1	0.2	98.2	0.333	0.6
SVM	1	1	100	1	1
NB	0.12	0.6	88.9	0.2	0.748

Table 10. PC3 Dataset Results

Classifier	Precision	Recall	Accuracy %	F-Measure	AUC
DT	1	0.795	97.5	0.886	0.898
SVM	1	1	100	1	1
NB	0.287	0.692	75	0.406	0.725

4.1. Comparative analysis

The integration of optimized parameters shows a substantial improvement in the prediction accuracy of the classifiers. Table 11 represents the results of both dimensions; accuracy achieved from classifiers with default parameters, and optimized parameters. Decision tree shows accuracy improvement for all datasets with maximum accuracy improved from 81.4% to 91.9% for MC2 dataset. Similarly, support vector machine exhibits enhanced accuracy for all datasets with most optimal results for PC3 dataset with an improvement from 89.2% to 100%. The significant accuracy results for both decision tree and support vector machine reflect that both the classifiers are sensitive to parameter optimization. However, naïve bayes shows varied performance across datasets after parameter optimization. It shows a maximum accuracy boost from 20.6% to 75% for PC3 dataset. The minimum accuracy increase is for CM1 dataset where accuracy increased from 84.8% to 85.9%. It is also evident that the accuracy remains same for KC3 dataset i.e., 81.4% demonstrating that the optimization technique didn't affect the simplicity of the naïve bayes classifier. It can also be seen that naïve bayes doesn't produce results for MC1 and MW1 datasets in their default configurations. It is because these datasets show zero or approaching to zero variance in features across all the instances. Since naïve bayes works on the principle of probability, and it relies on variance of features for probability calculation [58]. Fig. 4 shows a DT-based accuracy comparison of the default classifier and optimized classifier for all datasets. Fig. 5 shows an SVM-based accuracy comparison of the default classifier and optimized classifier for all datasets. Fig. 6 shows an NB-based accuracy comparison of the default classifier and optimized classifier for all datasets. Fig. 7 shows the AUC comparison with default classifiers for all datasets. Fig. 8 shows the AUC comparison with optimized classifiers for all datasets.

Table 11. Result of parameter optimization using accuracy

Dataset	CM1		KC3		MC1		MC2		MW1		PC2		PC3	
Classifier	**Optimized**	**Default**	**Default**	**Optimized**	**Default**	**Optimized**	**Default**	**Optimized**	**Default**	**Optimized**	**Default**	**Optimized**	**Default**	**Optimized**
DT	86	94	81.4	94.9	98.3	99	76.3	89.5	89.3	98.7	97.7	98.2	87.7	97.5
SVM	91.9	100	94.9	100	98.3	99	89.5	100	98.7	100	97.7	100	89.2	100
NB	84.8	85.9	81.4	81.4	-	96.6	73.7	81.6	-	88	86.2	88.9	20.6	75

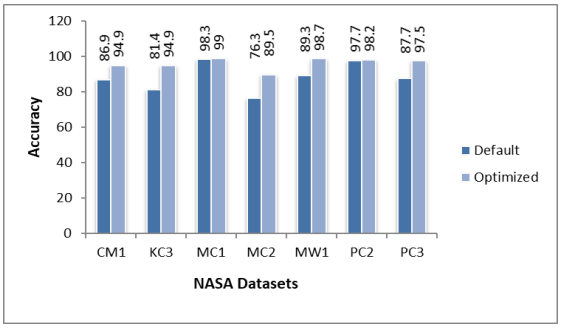

Fig. 4. Accuracy comparison of default and optimized parameters for DT

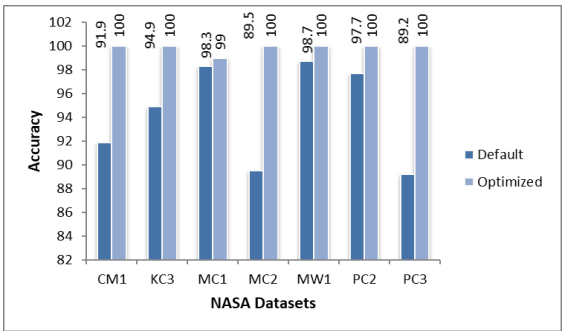

Fig. 5. Accuracy comparison of default and optimized parameters for SVM

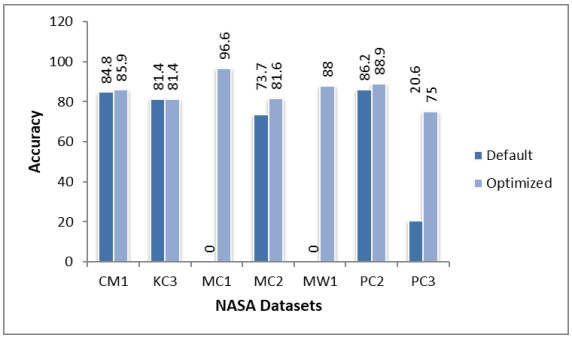

Fig. 6. Accuracy comparison of default and optimized parameters for NB

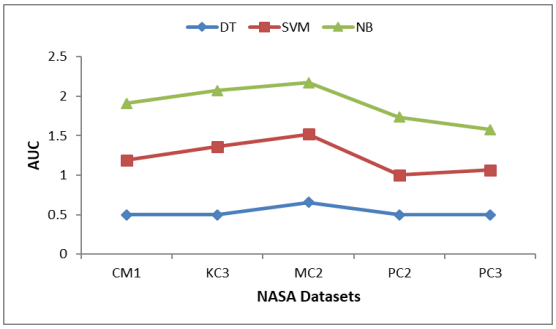

Fig. 7. Default AUC comparison of DT, SVM, and NB

This study has utilized seven diverse defect datasets belonging to NASA matric dataset program. These datasets have been derived from real-world projects that were developed using multiple programming languages including C, C++, and JAVA. Hence, the findings of the study can be effectively applied to other defect datasets from Eclipse, AEEEM, PROMISE or ReLink repositories [59]–[61]. These finding can also be applied to other defect prediction contexts i.e. within -project/cross-project defect prediction, and cross version defect prediction considering the dataset size, class-balanced ratio and the model complexity, and [62].

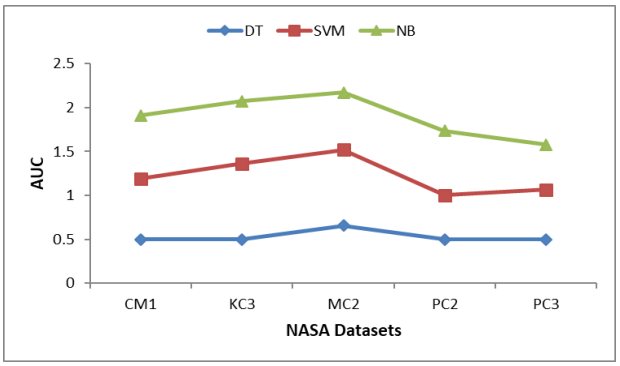

Fig. 8. Optimized AUC comparison of DT, SVM, and NB

4.2. Limitations

While the proposed methodology highlights the significant role of random search for parameter optimization for software defect prediction, there are some limitations associated with it as well. The experiments are specific to NASA defect datasets, the proposed methodology may deviate from the expected results on other defect datasets. The quality of the selected dataset is another concern to execute the proposed methodology. The study selected already preprocessed, clean version of NASA defect dataset [39]. Moreover, while performing experiments, the class-imbalance problem has not been addressed to maintain the originality of the datasets. This problem can lead to biased results. Addressing these limitations can enhance the efficiency of random search-based parameter optimization for binary classifiers.

5. CONCLUSION

Parameter optimization is a crucial aspect to enhance the efficiency of machine learning classifiers. The study aimed to systematically analyze and optimize the performance of machine learning classifiers using random search. This research presented the impact of parameter optimization on three heterogonous binary classifiers namely DT, SVM, and NB. The experiments were performed using a clean version of seven publicly available NASA datasets. The research was carried out in two dimensions; one with default classifiers and the second with optimized classifiers. The efficiency of the resultant classifiers was accessed using five performance measures i.e., precision, recall, F-measure, AUC, and accuracy. Results exhibit that the optimized classifiers outperform default classifiers. Hence. random search can help to identify the optimal parameters of binary classifiers leading to produce enhanced performance. The results reveal that random search can improve prediction accuracy, robustness, and generalization of binary classifiers. It is particularly useful when there is high-dimensional parameter space where grid-search or other searching techniques become impractical. By timely identifying potential software defects, organizations can do risk management and save their time, money, and manpower.

For future work, it is recommended that the resampling technique such as Synthetic Minority Over-sampling Technique (SMOTE) along with the feature selection using genetic algorithm should be applied to NASA datasets before experimenting with parameter optimization to attain improved results.

REFERENCES

[1] K. Wang, L. Liu, C. Yuan, and Z. Wang, "Software defect prediction model based on LASSO–SVM," *Neural Comput & Applic*, vol. 33, no. 14, pp. 8249–8259, Jul. 2021, https://doi.org/10.1007/s00521-020-04960-1.

[2] M. Ali *et al.*, "Software Defect Prediction Using an Intelligent Ensemble-Based Model," *IEEE Access*, vol. 12, pp. 20376–20395, 2024, https://doi.org/10.1109/ACCESS.2024.3358201.

[3] M. Ali, "Optimizing Software Defect Prediction: A Genetic Algorithm Based Comparative Analysis," *International* Journal *of Computational and Innovative Sciences-http://ijcis.com/index.php/IJCIS/article/view/86*, vol. 2, no. 4, pp. 56–73, 2024, https://ijcis.com/index.php/IJCIS/article/view/86.

[4] A. T. Elbosaty, W. M. Abdelmoez, and E. Elfakharany, "Within-Project Defect Prediction Using Improved CNN Model via Extracting the Source Code Features," in *2022 International Arab Conference on Information Technology (ACIT)*, pp. 1–8, 2022, https://doi.org/10.1109/ACIT57182.2022.9994220.

[5] U. S. Bhutamapuram and R. Sadam, "With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 8675–8691, Nov. 2022, https://doi.org/10.1016/j.jksuci.2021.09.010.

[6] N. Zhang, S. Ying, W. Ding, K. Zhu, and D. Zhu, "WGNCS: A robust hybrid cross-version defect model via multi-objective optimization and deep enhanced feature representation," *Information Sciences*, vol. 570, pp. 545–576, Sep. 2021, https://doi.org/10.1016/j.ins.2021.05.008.

[7] A. Soni, "Graph-Based and Anomaly Detection Learning Models for Just-in-Time Defect Prediction," *TRACE*, 2024, Accessed: Jul. 03, 2024. [Online]. Available: https://trace.tennessee.edu/utkgraddiss/10168/.

[8] C. Zhang and J. Wu, "Software Defect Prediction Based On Effective Fusion Of Multiple Features," *IEEE Access*, 2024, Accessed: Jul. 03, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10549525/.

[9] M. Nevendra and P. Singh, "Defect count prediction via metric-based convolutional neural network," *Neural Comput & Applic*, vol. 33, no. 22, pp. 15319–15344, Nov. 2021, https://doi.org/10.1007/s00521-021-06158-5.

[10] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Yasin Ghadi, and M. Amir Khan, "Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning," *PeerJ Computer Science*, vol. 10, p. e1860, Feb. 2024, https://doi.org/10.7717/peerj-cs.1860.

[11] A. J. Anju and J. E. Judith, "Adaptive recurrent neural network for software defect prediction with the aid of quantum theory- particle swarm optimization," *Multimed Tools Appl*, vol. 82, no. 11, pp. 16257–16278, May 2023, https://doi.org/10.1007/s11042-022-14065-7.

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

[13] M. Ali *et al.*, "*Analysis* of Feature Selection methods in Software Defect Prediction Models," *IEEE Access*, pp.

145954–145974, 2023, https://doi.org/10.1109/ACCESS.2023.3343249.
[14] C. Watson, N. Cooper, D. N. Palacio, K. Moran, and D. Poshyvanyk, "A Systematic Literature Review on the Use of Deep Learning *in* Software Engineering Research," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 2, pp. 1–58, Apr. 2022, https://doi.org/10.1145/3485275.
[15] S. Y. Kim and A. Upneja, "Majority voting ensemble with a decision trees for business failure prediction during economic *downturns*," *Journal of Innovation & Knowledge*, vol. 6, no. 2, pp. 112–123, Apr. 2021, https://doi.org/10.1016/j.jik.2021.01.001.
[16] J. Deng, L. *Lu*, and S. Qiu, "Software defect prediction via LSTM," *IET softw.*, vol. 14, no. 4, pp. 443–450, Aug. 2020, https://doi.org/10.1049/iet-sen.2019.0149.
[17] C. Peng, X. Wu, W. Yuan, X. Zhang, Y. Zhang, and Y. Li, "MGRFE: Multilayer Recursive Feature Elimination Based on an *Embedded* Genetic Algorithm for Cancer Classification," *IEEE/ACM Trans. Comput. Biol. and Bioinf.*, vol. 18, no. 2, pp. 621–632, Mar. 2021, https://doi.org/10.1109/TCBB.2019.2921961.
[18] X. Liu *et al.*, "Adapting Feature Selection Algorithms for the Classification of Chinese Texts," *Systems*, vol. 11, no. 9, p. 483, Sep. 2023, https://doi.org/10.3390/systems11090483.
[19] 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, Oct. 2019, https://doi.org/ 10.14710/jtsiskom.7.4.2019.121-126.
[20] C. Ni, K. Yang, Y. Zhu, X. Chen, and X. Yang, "Unifying Defect Prediction, Categorization, and Repair by Multi-Task Deep Learning," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Luxembourg, Luxembourg: IEEE, Sep. 2023, pp. 1980–1992. https://doi.org/10.1109/ASE56229.2023.00083.
[21] H. Park, H. Kwon, H. Cho, and J. Kim, "A framework for energy optimization of distillation process using machine learning-based predictive model," *Energy Science & Engineering*, vol. 10, no. 6, pp. 1913–1924, Jun. 2022, https://doi.org/10.1002/ese3.1134.
[22] E. K. Yilmaz and H. Bakir, "Hyperparameter tunning and feature selection methods for malware detection," *Politeknik* Dergisi, pp. 1–1, 2023, https://trace.tennessee.edu/utk_graddiss/10168/.
[23] Y. A. Ali, E. M. Awwad, M. Al-Razgan, and A. Maarouf, "Hyperparameter search for machine learning algorithms for optimizing the computational complexity," *Processes*, vol. 11, no. 2, p. 349, 2023, https://doi.org/10.3390/pr11020349.
[24] B. Bischl *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *WIREs Data Min & Knowl*, vol. 13, no. 2, p. e1484, Mar. 2023, https://doi.org/10.1002/widm.1484.
[25] H. Alsawalqah *et al.*, "Software Defect Prediction Using Heterogeneous Ensemble Classification Based on Segmented Patterns," *Applied Sciences*, vol. 10, no. 5, p. 1745, Mar. 2020, https://doi.org/10.3390/app10051745.
[26] H. S. Yadav, "Increasing Accuracy of Software Defect Prediction using 1-dimensional CNN with SVM," in *2020 IEEE International* Conference *for Innovation in Technology (INOCON)*, pp. 1–6, Nov. 2020, https://doi.org/10.1109/INOCON50539.2020.9298189.
[27] A. Iqbal, S. Aftab, I. Ullah, M. Salman Bashir, and M. Anwaar Saeed, "A Feature Selection based Ensemble Classification Framework for Software Defect Prediction," *IJMECS*, vol. 11, no. 9, pp. 54–64, Sep. 2019, https://doi.org/10.5815/ijmecs.2019.09.06.
[28] A. O. Balogun, S. Basri, S. J. Abdulkadir, and A. S. Hashim, "Performance Analysis of Feature Selection Methods in Software Defect Prediction: A Search Method Approach," *Applied Sciences*, vol. 9, no. 13, p. 2764, Jul. 2019, https://doi.org/10.3390/app9132764.
[29] M. S. Daoud *et al.*, "Machine Learning Empowered Software Defect Prediction System," *Intelligent Automation & Soft Computing*, vol. 31, no. 2, pp. 1287–1300, 2022, https://doi.org/10.32604/iasc.2022.020362.
[30] A. Iqbal *et al.*, "Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets," *IJACSA*, vol. 10, no. 5, 2019, https://doi.org/10.14569/IJACSA.2019.0100538.
[31] J. Kang, S. Kwon, D. Ryu, and J. Baik, "HASPO: Harmony Search-Based Parameter Optimization for Just-in-Time Software Defect Prediction in Maritime Software," *Applied Sciences*, vol. 11, no. 5, p. 2002, Feb. 2021, https://doi.org/10.3390/app11052002.
[32] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The Impact of Automated Parameter Optimization on Defect Prediction Models," *IIEEE Trans. Software Eng.*, vol. 45, no. 7, pp. 683–711, Jul. 2019, https://doi.org/10.1109/TSE.2018.2794977.
[33] O. M. Maruf, "The impact of parameter optimization of ensemble learning on defect prediction," *Computer Science Journal of Moldova,* vol. 79, no. 1, pp. 85-128, 2019, https://ibn.idsi.md/vizualizare_articol/78371.
[34] 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, https://doi.org/10.1109/ACCESS.2020.2968362.
[35] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: multi-objective hyper-parameter optimization," in *Proceedings of the 35th International Conference on Computer-Aided Design*, pp. 1–8, Nov. 2016, https://doi.org/10.1145/2966986.2967058.
[36] L. Shen, W. Liu, X. Chen, Q. Gu, and X. Liu, "Improving Machine Learning-Based Code Smell Detection via Hyper-Parameter Optimization," in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 276–285, Dec. 2020, https://doi.org/10.1109/APSEC51365.2020.00036.
[37] R. Yedida and T. Menzies, "How to improve deep learning for software analytics: (a case study with code smell

detection)," in *Proceedings of* the *19th International Conference on Mining Software Repositories*, pp. 156–166, May 2022, https://doi.org/10.1145/3524842.3528458.
[38] A. O. Balogun *et al.*, "Software Defect Prediction Using Wrapper Feature Selection Based on Dynamic Re-Ranking Strategy," *Symmetry*, vol. 13, no. 11, p. 2166, Nov. 2021, https://doi.org/10.3390/sym13112166.
[39] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," *IIEEE Trans. Software Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013, https://doi.org/10.1109/TSE.2013.11.
[40] M. Cetiner and O. K. Sahingoz, "A Comparative Analysis for Machine Learning based Software Defect Prediction Systems," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–7, Jul. 2020, https://doi.org/10.1109/ICCCNT49239.2020.9225352.
[41] O. A. 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, https://doi.org/10.1109/ACCESS.2020.2985290.
[42] A. J. Anju and J. E. Judith, "A reliable impact factor for feature Selection," *presented at the International Scientific And Practical Conference "Innovative Technologies In Agriculture," Orel City, Russian Federation,* p. 020004, 2023, https://doi.org/10.1063/5.0170391.
[43] R. Yedida and T. Menzies, "On the Value of Oversampling for Deep Learning in Software Defect Prediction," *IIEEE Trans. Software Eng.*, vol. 48, no. 8, pp. 3103–3116, Aug. 2022, https://doi.org/10.1109/TSE.2021.3079841.
[44] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *Journal of Systems and Software*, vol. 195, p. 111537, Jan. 2023, https://doi.org/10.1016/j.jss.2022.111537.
[45] H. Alibrahim and S. A. Ludwig, "Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1551–1559, Jun. 2021, https://doi.org/10.1109/CEC45853.2021.9504761.
[46] L. V, -Arias, C. Q. -López, J. G. -Coto, A. Martínez, and M. Jenkins, "Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation," in *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 31–40, Nov. 2020, https://doi.org/10.1145/3416508.3417121.
[47] A. R. M. Rom, N. Jamil, and S. Ibrahim, "Multi objective hyperparameter tuning via random search on deep learning models," *TELKOMNIKA*, vol. 22, no. 4, p. 956, Aug. 2024, https://doi.org/10.12928/telkomnika.v22i4.25847.
[48] R. Aschauer, "Predictive Modeling of Next Product to Buy in the Banking Sector Using Boosting Techniques," p. 72 pages, 2024, https://doi.org/10.34726/HSS.2024.121688.
[49] S. Mehta and K. S. Patnaik, "Improved prediction of software defects using ensemble machine learning techniques," Neural *Comput & Applic*, vol. 33, no. 16, pp. 10551–10562, Aug. 2021, https://doi.org/10.1007/s00521-021-05811-3.
[50] Y. Lan, T. T. -Huu, J. Wu, and S. G. Teo, "Cascaded Multi-Class Network Intrusion Detection With Decision Tree and Self-attentive Model," *IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 1-7, 2022, https://doi.org/10.1109/ICDMW58026.2022.00081.
[51] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, Apr. 2020, https://doi.org/10.1016/j.neucom.2019.11.067.
[52] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Machine Learning-Based Software Defect Prediction for Mobile Applications: A Systematic Literature Review," *Sensors*, vol. 22, no. 7, p. 2551, Mar. 2022, https://doi.org/10.3390/s22072551.
[53] P. Suresh Kumar, H. S. Behera, J. Nayak, and B. Naik, "Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature," *Innovations Syst Softw Eng*, vol. 17, no. 4, pp. 355–379, Dec. 2021, https://doi.org/10.1007/s11334-021-00399-2.
[54] N. Monga and P. Sehga, "Effective Software Defect Prediction: Evaluating Classifiers and Feature Selection with Firefly Algorithm.," *International Journal of Performability Engineering*, vol. 20, no. 4, 2024, Accessed: Jun. 30, 2024. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=09731318&AN=176892169&h=y%2BIxC7kNfAjlZxWWmZFZdLLqtjCe3FdxJeYuiizFbQAnFqfbbIuYLKaNrUsz9uMUnB%2F2w6Gt0s%2FzhBkmB5nddA%3D%3D&crl=c.
[55] N. A. A. Khleel and K. Nehéz, "Software defect prediction using a bidirectional LSTM network combined with oversampling techniques," *Cluster Comput*, vol. 27, no. 3, pp. 3615-3638, Oct. 2023, https://doi.org/10.1007/s10586-023-04170-z.
[56] R. Mo, Y. Wang, Y. Zhang, and Z. Li, "Just-in-Time Defect Severity Prediction (S).," in *SEKE*, pp. 232–237. Accessed: Jun. 30, 2024. [Online]. Available: https://ksiresearch.org/seke/seke23paper/paper218.pdf.
[57] U. S. B and R. Sadam, "Towards Developing and Analysing Metric-Based Software Defect Severity Prediction Model," *arXiv preprint arXiv:2210.04665*, 2022, https://arxiv.org/abs/2210.04665.
[58] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software Defect Prediction Analysis Using Machine Learning Techniques," *Sustainability*, vol. 15, no. 6, p. 5517, Mar. 2023, https://doi.org/10.3390/su15065517.
[59] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *Journal of Systems and Software*, vol. 180, p. 111026, Oct. 2021, https://doi.org/10.1016/j.jss.2021.111026.
[60] L. Chen, C. Wang, and S. Song, "Software defect prediction based on nested-stacking and heterogeneous feature

selection," *Complex Intell. Syst.*, vol. 8, no. 4, pp. 3333–3348, Aug. 2022, https://doi.org/10.1007/s40747-022-00676-y.

[61] M. *Mafarja et al.*, "Classification framework for faulty-software using enhanced exploratory whale optimizer-based feature selection scheme and random forest ensemble learning," *Appl Intell*, vol. 53, no. 15, pp. 18715–18757, Aug. 2023, https://doi.org/10.1007/s10489-022-04427-x.

[62] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, "Ensemble Machine Learning Paradigms in Software Defect Prediction," *Procedia Computer Science*, vol. 218, pp. 199–209, 2023, https://doi.org/10.1016/j.procs.2023.01.002