Software Defect Prediction Based on Optimized Machine Learning Models: A Comparative Study

Muhammad Zain Fawwaz Nuruddin Siswantoro[1*], Umi Laili Yuhana[2]

[1,2] Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Jawa Timur
Email: [1*] 6025222009@mhs.its.ac.id, [2] yuhana@if.its.ac.id

(Naskah masuk: 24 Mei 2023, direvisi: 18 Jun 2023, diterima: 21 Jun 2023)

Abstract

Software defect prediction is crucial used for detecting possible defects in software before they manifest. While machine learning models have become more prevalent in software defect prediction, their effectiveness may vary based on the dataset and hyperparameters of the model. Difficulties arise in determining the most suitable hyperparameters for the model, as well as identifying the prominent features that serve as input to the classifier. This research aims to evaluate various traditional machine learning models that are optimized for software defect prediction on NASA MDP (Metrics Data Program) datasets. The datasets were classified using k-nearest neighbors (k-NN), decision trees, logistic regression, linear discriminant analysis (LDA), single hidden layer multilayer perceptron (SHL-MLP), and Support Vector Machine (SVM). The hyperparameters of the models were fine-tuned using random search, and the feature dimensionality was decreased by utilizing principal component analysis (PCA). The synthetic minority oversampling technique (SMOTE) was implemented to oversample the minority class in order to correct the class imbalance. k-NN was found to be the most suitable for software defect prediction on several datasets, while SHL-MLP and SVM were also effective on certain datasets. It is noteworthy that logistic regression and LDA did not perform as well as the other models. Moreover, the optimized models outperform the baseline models in terms of classification accuracy. The choice of model for software defect prediction should be based on the specific characteristics of the dataset. Furthermore, hyperparameter tuning can improve the accuracy of machine learning models in predicting software defects.

Keywords: Machine Learning Models, Software Defect Prediction, Random Search, Principal Component Analysis, Hyperparameter Tuning.

I. INTRODUCTION

As technology has advanced and consumer expectations for software have risen, the software development process has gotten increasingly intricate [1]. As a result, software engineers must now focus on improving their ability to detect and prevent software defects [2]. Software Defect Prediction (SDP) is a crucial technique that identifies potential software defects before they occur. In software engineering, SDP is an important and challenging task. Better software quality and reduced development costs are both linked to early defect detection in software development [3], [4].

Recently, machine learning models have been widely used to detect defects in software. This is because machine learning models have the ability to find patterns automatically from data by recognizing defects in software [5]. Predicting software defects using machine learning models has been demonstrated to be useful in several studies, such as decision tree (DT) [6], Naïve Bayes (NB) [7], K-nearest Neighbors (k-NN) [8], [9], Artificial Neural Network (ANN) [10], and Support Vector Machine (SVM) [11]. Different datasets and model hyperparameters can result in widely varying model performances in machine learning. A common challenge in machine learning is selecting the optimal model hyperparameters. However, almost all studies in SDP using machine learning models did not perform hyperparameter tuning to obtain the optimal model hyperparameters.

Another issue in SDP using a machine learning model is selecting prominent features to use as input to the classifier. The optimal feature subset has been chosen to use several feature selection techniques to avoid a decline in the performance of classification models for SDP caused by redundant and irrelevant features, as reported in [12]–[14]. The quality of results obtained by feature selection techniques is very dependent on datasets. Principal Component Analysis (PCA) is another approach that can be used to reduce irrelevant features. In machine learning, PCA reduces data dimensionality while maintaining as much information as possible to find patterns [15]. To achieve optimal classification performance, however, it is necessary to determine the optimal number of selected components.

This paper compares the optimized machine learning models for SDP on NASA MDP (Metrics Data Program) datasets [16]. Some traditional machine learning models were used to classify 12 datasets from NASA MDP datasets: k-nearest neighbors (k-NN), decision trees (DT), logistic regression (LR), linear discriminant analysis (LDA), single hidden layer multilayer perceptron (SHL-MLP), and support vector machine (SVM). The hyperparameters of the model were optimized using random search [17] to obtain the best classifier for each dataset. Before being input to the classifier, the dimensionality of the features was reduced using PCA. The number of selected components was also optimized using random search. The synthetic minority oversampling technique (SMOTE) was used as an oversampling strategy for the minority class to deal with unbalanced samples on NASA MDP datasets [18].

The remaining sections of the paper are structured as follows: Section two provides a theoretical foundation for the study by reviewing the relevant literature. Section three explains the methods used in this study. Following that, the findings and discussion of their implications will be presented in Section 4. Finally, a summary of the key findings and suggestions for avenues for future research will be provided in the last section.

II. LITERATURE REVIEW

Iqbal et al. [5] analyzed the effectiveness of several machine learning models for SDP using NASA MDP datasets. The performance of ten machine learning models was measured using a variety of evaluation metrics. These models included k-NN, DT, LR, MLP, SVM, radial basis function (RBF), one rule (OR), kStart (PART), and random forest (RF). The findings indicate that the metrics used to evaluate the performance of the model change depending on the dataset, except for the ROC area score. Based on the ROC area score, RF achieved higher performance compared to other models.

A novel method for SDP based on a weighted naive Bayes classifier was proposed by Ji et al. [6]. The authors leverage the concept of information diffusion to assign weights to the features used in the classifier, resulting in improved prediction accuracy compared to traditional naive Bayes classifiers. While the approach shows promise, the authors' experimental evaluation could benefit from larger and more diverse datasets to better demonstrate the method's effectiveness.

Marian et al. [7] proposed a new approach to predicting software defects using fuzzy decision trees. The authors claim that this approach outperforms standard DT in AUC scores. While the concept of fuzzy decision trees is intriguing, the study lacks sufficient information about the implementation and evaluation processes, such as the selection of input features and the selection criteria for the best model. In addition, the dataset used in the experiments is limited to two software projects, which casts doubt on the generalizability of the proposed method.

Hammad et al. [8] presented a machine learning approach to predict software faults using k-NN. The authors conducted experiments and achieved promising results, with an accuracy rate of up to 87%. Kumar et al. [9] proposed a new approach for predicting software defects in Aspect-Oriented Programming (AOP). The authors use a combination of fuzzy c-means clustering with genetic algorithms (FCM-GM) and k-NN. The experimental results show that the proposed FCM-GM outperforms traditional FCM and k-NN. However, the classification models in [8], [9] were only evaluated using five datasets and one dataset from NASA MDP datasets, respectively.

Rong et al. [10] proposed a new method for software defect prediction using SVM and a bat algorithm with centroid strategy (CBA). CBA was used to optimize the parameters of SVM to enhance the accuracy of the prediction model. The central concept of the optimization algorithm involves treating SVM parameters as particles in CBA, which then undergoes self-updating until the algorithm achieves its final condition. The experimental results show that the proposed method outperforms other classifiers, including standard SVM. However, the proposed method was only evaluated with four datasets from the NASA MDP datasets.

Jayanthi et al. [11] proposed a method for SDP that uses an ANN and an enhanced version of PCA. Their improvement involves merging PCA with maximum likelihood in order to minimize the PCA reconstructed data. The experimental results reveal that the proposed approach surpasses other existing models, reaching an AUC of 97.20% and substantially improving classification accuracy. However, the authors did not explain how to decide on the number of principal components chosen in PCA and the ANN architecture used for each dataset, despite claiming that their method achieves good performance.

Nevertheless, most studies using traditional machine learning models for SDP did not perform hyperparameter tuning to obtain the best classifier. The number of components used in PCA was also not optimized so that the best performance was achieved by each model. Therefore, this research attempts to compare the performance of several traditional machine learning models by performing hyperparameter tuning both on the classifier and PCA to obtain the best classification performance.

III. METHODS

This research involved multiple steps to conduct a comparative analysis of optimized traditional machine learning models. These procedures consisted of gathering a dataset, oversampling the minority class, using PCA for dimensionality reduction, training various traditional machine learning models for classification with hyperparameter tuning, and evaluating the models as illustrated in Figure 1. The subsequent subsections provide a comprehensive explanation of each step.

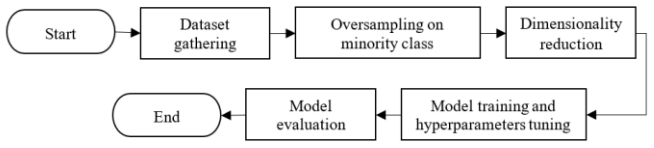
Start
Dataset gathering
Oversampling on minority class
Dimensionality reduction
Model training and hyperparameters tuning
Model evaluation
End

Figure 1. Steps for Conducting the Research.

A. Dataset

This research employed the dataset from the NASA Metrics Data Program (MDP) to evaluate the classification model used. The NASA MDP is a dataset about software defects in different NASA projects. This includes information like how many defects were found in each project, how big the code base is, and how much work it took to make the software. Software engineers frequently use this dataset to examine how different software metrics relate to software defects. The MDP dataset comprises both public and confidential data. The former provides information on 24 NASA software projects, while the latter contains additional project information that is accessible only to authorized users [19]. This research used the clean version of NASA MDP datasets from D" collection as described in [16]. The datasets consisted of datasets from 12 projects, namely CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5. The number of features varies in each dataset but has the same number of classes, namely defective Y and defective N. Table 1 shows the detailed description of the 12 datasets used in this research.

Table 1. The Description of 12 NASA MDP

Dataset	Instances			Features
	Total	Non-defective	Defective	
CM1	327	285	42	37
JM1	7782	6110	1672	21
KC1	1183	869	314	21
KC3	194	158	36	39
MC1	1988	1942	46	38
MC2	125	81	44	39
MW1	253	226	27	37
PC1	705	644	61	37
PC2	745	729	16	36
PC3	1077	943	134	37
PC4	1287	1110	177	37
PC5	1711	1240	471	38

B. Oversampling Strategy

As can be seen in Table 1, the number of defective instances is significantly smaller than the number of non-detective instances in all datasets. This condition is considered a class imbalance problem. If this condition is not addressed, it will affect the performance of the machine learning model. Therefore, this research applied an oversampling strategy to the minority class using the synthetic minority oversampling technique (SMOTE) [18]. SMOTE is a popular oversampling technique used in machine learning to address the class imbalance problem. SMOTE works by creating synthetic instances of the minority class by interpolating between the instances of the minority class. Specifically, SMOTE selects a minority class instance and finds its k-nearest neighbors in the feature space. It then creates synthetic instances by randomly selecting one of the k-neighbors and interpolating between the minority sample and the selected neighbor. This creates a new instance that is similar to the minority class but is not an exact copy of any existing instance.

C. Dimensionality Reduction

The presence of numerous features for training a machine learning model does not guarantee its good performance. Furthermore, a high number of features can also increase the amount of time and computational resources needed during the training process [20]. Therefore, this research employed principal component analysis (PCA) [15] to reduce the dimensionality of features. PCA is a commonly used technique in machine learning for dimensionality reduction. PCA is an unsupervised dimensionality reduction method that can reduce the high dimensionality of features into fewer significant and uncorrelated principal components while retaining the essential information of the original features [20]. PCA seeks to identify the most significant features or variables in a dataset and depict them in a space with fewer dimensions.

Reducing feature dimensions is done by transforming the features into new variables that are not correlated with each other but can still explain as much of the variation in the original data as possible. These variables are called principal components. PCA employed the covariance matrix of the original data to determine the principal components and the amount of variance explained by each component by calculating the eigenvectors and the eigenvalues of the matrix, respectively. The dataset can be projected onto a lower n-dimensional space by choosing only the top n eigenvectors, where n is the desired number of dimensions (primary components). In this research, the value of n was determined using a random search from $\{n \in \mathbb{N} | 5 \leq n \leq N\}$ such that the best performance of the model is achieved, where N is the number of features. Before being inputted to the classifier selected components were scaled into intervals $[0,1]$ to avoid the dominance of certain components using equation (1),

$$x_s = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{1}$$

where x_s is scaled component, x is the original component, min (x) and max (x) are the minimum and maximum of the selected component, respectively.

D. Classification

In this research, six traditional machine learning models were employed to classify 12 datasets from NASA MDP datasets into two classes: defective Y and defective N. The models used include k-nearest neighbors (k-NN), logistic regression (LR), decision tree (DT), linear discriminant analysis (LDA), support vector machine (SVM), and single hidden layer multi-layer perceptron (SHL-MLP) [21].

Parameters are a part of every model; these are not learned by the model during training but rather are established by the user beforehand. Hyperparameters are another name for this type of parameter. Therefore, the model's hyperparameters must be adjusted for optimal performance.

This research employed random search [17] to determine the optimum hyperparameters for each model. Finding the optimal value of a function with random search optimization is a straightforward and efficient process. Several points are generated randomly at predetermined intervals. The function value is then evaluated at these points to determine the optimum point. This process is carried out iteratively until the stopping criterion is met. Random search optimization is a simple method that is easy to implement because it does not require a lot of information about the functions being optimized including function derivatives. The tuned hyperparameters and the search domains for each model are tabulated in Table 2.

E. Model Evaluation

To evaluate the classification models, each dataset was divided into two parts, which are the training data and the testing data, with a ratio of 70:30, using stratified random subsampling [22]. Stratified random subsampling ensured that every class in the dataset was equally represented in both the training and testing datasets, maintaining the same proportion as the original dataset. The main objective of employing this method was to ensure that the training and testing datasets reflected the overall dataset in its entirety. The model was then trained and evaluated using the training data and the testing, respectively. The model's performance was also measured using the testing data.

Table 2. Tuned Hyperparameters and Search Domains

Model	Hyper-parameter	Description	Search Domain
k-NN	k	The number of nearest neighbors	[1,10]
LR	Penalty	Norm of penalty	{'l1', 'l2', 'elasticnet', 'none'}
	C	Regularization parameter	(1,1000)
	Solver	Optimization algorithm	{'sag', 'saga', 'newton-cholesky', 'lbfgs', 'liblinear', 'newton-cg' }
DT	Criterion	Measurement function for split quality	{'gini', 'entropy'}
LDA	Solver	Estimation algorithms	{'svd', 'lsqr', 'eigen'}
SVM	C	Regularization parameter	(1,1000)
	Kernel	Kernel function	{'rbf','poly', 'sigmoid'}
	Gamma	Kernel coefficient	(0.01,1)
SHL-MLP	Activation	The hidden layer's activation function	{'identity', 'logistic','tanh', 'relu'}
	Neurons	The number of neurons in the hidden layer	$\{5,6,\ldots,1000\}$
	Solver	Optimization algorithm	{'lbfgs','sgd', 'adam'}

Several metrics were employed to evaluate the performance of the models using the testing dataset, including accuracy, precision, recall, and F1 score. Equation (2) was used to determine the classification accuracy, which is the proportion of right predictions throughout the whole dataset,

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

where TP is the number of instances that are actually positive and classified as positive, TN is the number of instances that are actually negative and classified as negative, FP is the number of instances that are actually negative but classified as positive, and FN is the number of instances that are actually positive but classified as negative. Precision measures the proportion of true positive predictions out of all positive predictions and is calculated using equation (3).

$$precision = \frac{TP}{TP + FP} \tag{3}$$

Recall measures the proportion of true positive predictions out of all actual positive instances and is calculated using equation (4).

$$recall = \frac{TP}{TP + FN} \tag{4}$$

Lastly, the F1 score is the harmonic mean of precision and recall and is calculated using equation (5).

$$F1\ score = \frac{2 \times precision \times recall}{precision + recall} \tag{5}$$

In this research, all models were trained and evaluated using Python language programming with some open source Python libraries, namely pandas [23], imbalanced-learn [24], and Scikit-learn [25]. Pandas were used to import the dataset from a source file. Imbalanced-learn was used to perform oversampling on minority classes using SMOTE. Scikit-learn was used for training and testing all machine learning models, as well as tuning the hyperparameters.

IV. RESULTS AND DISCUSSION

This section presents the experiment results that aimed to compare the performance of various traditional machine learning models in predicting software defects. The accuracy, precision, recall, and F1 score of each model that are used to evaluate the effectiveness of the model in predicting software defects are presented in this section. The classification accuracy of the optimized models on all datasets is tabulated in Table 3.

As can be seen in Table 3, *k*-NN achieved the highest accuracy on seven datasets, which are JM1 77.91%, KC1 79.31%, KC3 91.58%, MC1 98.97%, MC2 83.67%, PC3 93.46%, and PC5 83.06%. In second place, SVM achieved the highest accuracy on four datasets, which are CM1 97.66%, KC3 91.58%, MW1 98.53%, and PC4 93.69%. In third place, SHL-MLP achieved the highest accuracy on three datasets, which are MC1 98.97%, PC1 96.12%, and PC3 93.46%. Logistic regression and decision tree achieved the highest accuracy only on a dataset, MC2 83.67% and PC2 98.86%, respectively. While LDA never achieved the highest accuracy. Based on the accuracy scores in Table 3, it can be concluded that k-NN consistently performs well across most datasets, achieving high accuracy rates. Additionally, SVM and SHL-MLP also demonstrate competitive performance in terms of accuracy.

Table 3. The Accuracy of Optimized Models on All Datasets

Dataset	**Accuracy (%)**					
	***k*-NN**	**LR**	**DT**	**LDA**	**SVM**	**SHL-MLP**
CM1	96.49	80.12	87.13	84.21	**97.66**	97.08
JM1	**77.91**	64.24	71.47	63.99	67.35	71.09
KC1	**79.31**	63.79	73.37	65.13	71.84	73.37
KC3	**91.58**	85.26	82.11	83.16	**91.58**	88.42
MC1	**98.97**	87.05	97.68	85.76	98.63	**98.97**
MC2	**83.67**	**83.67**	73.47	69.39	77.55	75.51
MW1	95.59	80.15	93.38	80.15	**98.53**	97.06
PC1	95.09	86.56	91.21	83.98	95.35	**96.12**
PC2	98.17	92.92	**98.86**	91.78	98.63	97.95
PC3	**93.46**	81.98	87.10	80.92	88.87	**93.46**
PC4	91.89	86.49	89.19	82.43	**93.69**	92.49
PC5	**83.06**	71.77	78.63	71.51	79.17	76.21

Table 4 shows the precision of the optimized models on all datasets. As can be seen in Table 4, *k*-NN achieved the highest precision on seven datasets, which are JM1 78.51%, KC1 79.70%, KC3 91.64%, MC1 98.99%, MC2 83.86%, PC3 93.70%, and PC5 83.28%. In second place, SVM achieved the highest precision on four datasets, which are CM1 97.68%, KC3 91.64%, MW1 98.57%, and PC4 94.05%. In third place, SHL-MLP achieved the highest precision on two datasets, which are MC1 98.99% and PC1 96.27%. The decision tree achieved the highest precision only on a dataset PC2 98.86%. While logistic regression and LDA never achieved the highest precision. Based on the precision values in Table 4, it can be concluded that k-NN consistently achieves high precision values across various datasets. These results indicate the effectiveness of k-NN in correctly identifying positive instances across all positive predictions. In addition, SVM and SHL-MLP also perform well in most of the datasets. However, when compared, LR, DT, and LDA show relatively lower precision values.

Table 5 shows the recall of the optimized models on all datasets. As can be seen in Table 5, *k*-NN achieved the highest recall on seven datasets, which are JM1 77.91%, KC1 79.31%, KC3 91.60%, MC1 98.97%, MC2 83.75%, PC3 93.46%, and PC5 83.06%. In second place, SVM achieved the highest recall on four datasets, which are CM1 97.67%, KC3 91.60%, MW1 98.53%, and PC4 93.69%. In third place, SHL-MLP achieved the highest recall on three datasets, which are MC1 98.97%, PC1 96.13%, and PC3 93.46%. The decision tree achieved the highest recall only on a dataset PC2 98.86%. While logistic regression and LDA never achieved the highest recall. Based on the recall values in Table 5, it can be concluded that k-NN consistently achieves the highest recall rate across all datasets, followed by SVM and SHL-MLP. Logistic Regression, DT, and LDA generally show lower recall rates compared to k-NN, SVM, and SHL-MLP. These results indicate k-NN is more effective in correctly identifying positive examples across all actual positive data.

Table 4. The Precision of Optimized Models on All Datasets

Dataset	**Precision (%)**					
	***k*-NN**	**LR**	**DT**	**LDA**	**SVM**	**SHL-MLP**
CM1	96.70	80.26	87.31	85.27	**97.68**	97.22
JM1	**78.57**	65.17	71.53	64.55	68.03	71.09
KC1	**79.70**	63.88	73.85	65.66	72.08	73.38
KC3	**91.64**	85.31	82.24	83.22	**91.64**	88.43
MC1	**98.99**	87.27	97.69	86.52	98.66	**98.99**
MC2	**83.86**	83.67	73.82	69.42	77.59	75.50
MW1	95.95	80.31	93.47	80.15	**98.57**	97.10
PC1	95.35	86.58	91.22	83.98	95.65	**96.27**
PC2	98.24	93.80	**98.86**	92.94	98.67	98.03
PC3	**93.70**	82.25	87.18	81.11	89.34	93.66
PC4	92.51	86.49	89.28	82.79	**94.05**	92.57
PC5	**83.28**	72.14	78.63	71.73	79.19	76.21

Table 5. The Recall of Optimized Models on All Datasets

Dataset	**Recall (%)**					
	***k*-NN**	**LR**	**DT**	**LDA**	**SVM**	**SHL-MLP**
CM1	96.51	80.14	87.15	84.26	**97.67**	97.09
JM1	**77.91**	64.24	71.47	63.99	67.35	71.09
KC1	**79.31**	63.79	73.37	65.13	71.84	73.37
KC3	**91.60**	85.28	82.07	83.13	**91.60**	88.43
MC1	**98.97**	87.05	97.68	85.76	98.63	**98.97**
MC2	**83.75**	83.67	73.58	69.42	77.50	75.50
MW1	95.59	80.15	93.38	80.15	**98.53**	97.06

PC1	95.10	86.57	91.22	83.98	95.36	**96.13**
PC2	98.17	92.92	**98.86**	91.78	98.63	97.95
PC3	**93.46**	81.98	87.10	80.92	88.87	**93.46**
PC4	91.89	86.49	89.19	82.43	**93.69**	92.49
PC5	**83.06**	71.77	78.63	71.51	79.17	76.21

Table 6 shows the F1 score of the optimized models on all datasets. As can be seen in Table 6, *k*-NN achieved the highest F1 score on six datasets, which are JM1 77.78%, KC1 79.24%, KC3 91.58%, MC1 98.97%, MC2 83.67%, and PC5 83.04%. In second place, SVM achieved the highest F1 score on four datasets, which are CM1 97.66%, KC3 91.58%, MW1 98.53%, and PC4 93.69%. In third place, SHL-MLP achieved the highest F1 score on three datasets, which are MC1 98.97%, PC1 96.12%, and PC3 93.46%. Logistic regression and decision tree achieved the highest F1 score only on a dataset MC2 83.67% and PC2 98.86%, respectively. While logistic regression and LDA never achieved the highest F1 score. Based on the F1 scores in Table 6, it can be concluded that k-NN generally performs well and LR and LDA achieve lower scores. In addition, SVM and SHL-MLP demonstrate good performance on most datasets but are not consistently superior to other algorithms. These results indicate k-NN performs well in terms of both correctly identifying positive instances and capturing all positive instances.

Table 6. The F1 Score of Optimized Models on All Datasets

Dataset	**F1 Score (%)**					
	k-NN	**LR**	**DT**	**LDA**	**SVM**	**SHL-MLP**
CM1	96.49	80.10	87.12	84.10	**97.66**	97.07
JM1	**77.78**	63.68	71.45	63.64	67.04	71.08
KC1	**79.24**	63.73	73.24	64.84	71.76	73.37
KC3	**91.58**	85.26	82.07	83.14	**91.58**	88.42
MC1	**98.97**	87.03	97.68	85.69	98.63	**98.97**
MC2	**83.67**	**83.67**	73.43	69.39	77.51	75.50
MW1	95.58	80.12	93.38	80.15	**98.53**	97.06
PC1	95.08	86.56	91.21	83.98	95.34	**96.12**
PC2	98.17	92.89	**98.86**	91.72	98.63	97.94
PC3	93.45	81.94	87.10	80.89	88.84	**93.46**
PC4	91.86	86.49	89.18	82.38	**93.68**	92.49
PC5	**83.04**	71.66	78.63	71.43	79.16	76.21

The results presented in Tables 3-6 provide insights into the performance of various optimized traditional machine learning models for software defect prediction. It can be observed that k-NN, SVM, and SHL-MLP are the most effective algorithms for predicting software defects, based on the highest accuracy, precision, recall, and F1 scores achieved on several datasets. k-NN outperformed all other algorithms in terms of accuracy, precision, recall, and F1 score on seven datasets, including JM1, KC1, KC3, MC1, MC2, PC3, and PC5. This suggests that k-NN is a suitable algorithm for software defect prediction and can be relied upon for accurate and precise predictions on these datasets.

SVM achieved the highest accuracy, precision, recall, and F1 score on four datasets, including CM1, KC3, MW1, and PC4. SHL-MLP achieved the highest accuracy, precision, recall, and F1 score on three datasets, including MC1, PC1, and PC3. This suggests that SVM and SHL-MLP are other effective models for software defect prediction and can be used in scenarios where k-NN may not be the best choice. It is also interesting to note that logistic regression and LDA never achieved the highest accuracy, precision, recall, or F1 score on any of the datasets. This suggests that these algorithms may not be the best choice for software defect prediction, at least in the context of the datasets used in this research.

Furthermore, most of the classification accuracy of the optimized machine learning models significantly outperforms the baseline models reported in [5] which do not employ hyperparameter tuning, as tabulated in Table 7. For example, the accuracy of optimized k-NN ranges from 77.91% to 98.97%. On the other hand, the accuracy achieved by k-NN in [5] ranges from 69.34% to 97.27%. Similarly, the optimized SVM achieved higher accuracy between 79.31% and 98.53%, when compared to the SVM accuracy reported in [5], which is between 62.16% and 90.82%. Therefore, the optimized models are much more accurate and reliable than the baseline models.

Table 7. The Accuracy of Unoptimized Models on All Datasets Reported in [5]

Dataset	**Accuracy (%)**			
	***k*-NN**	**DT**	**SVM**	**SHL-MLP**
CM1	77.55	77.55	90.82	86.73
JM1	73.96	79.10	79.19	80.35
KC1	69.34	75.64	75.36	77.36
KC3	75.86	75.86	82.76	82.76
MC1	97.27	97.61	97.61	97.61
MC2	72.97	64.86	62.16	64.86
MW1	86.67	86.67	89.33	90.67
PC1	92.65	93.14	95.10	96.57
PC2	96.77	97.70	97.70	96.77
PC3	24.00	86.39	86.39	83.86
PC4	85.83	86.88	88.19	89.76
PC5	73.03	75.00	74.21	74.21

V. CONCLUSION

This paper presents a comparative study of optimized machine learning models for software defect prediction on NASA MDP datasets. The hyperparameters of models were optimized using random search to obtain the best classifier for each dataset. Before input to the classifier, the dimensionality of features was reduced using PCA, and the number of selected components was also optimized using random search. The experiment results showed that k-NN achieved the highest accuracy, precision, and recall on the majority of the datasets, followed by SVM and SHL-MLP. These findings suggest that optimized machine learning models, combined with dimensionality reduction, can improve the effectiveness of software defect prediction, which can reduce costs and improve the quality of software products. To improve the performance of traditional machine learning models, the use of classification ensembles should be considered for further research.

REFERENCES

[1] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *Int. J. Appl. Sci. Eng.*, vol. 17, no. 4, pp. 331–344, 2020.

[2] P. Roy, G. S. Mahapatra, P. Rani, S. K. Pandey, and K. N. Dey, "Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction," *Appl. Soft Comput.*, vol. 22, pp. 629–637, 2014.

[3] Z. Xu *et al.*, "Software defect prediction based on kernel PCA and weighted extreme learning machine," *Inf. Softw. Technol.*, vol. 106, pp. 182–200, 2019.

[4] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, 2011.

[5] A. Iqbal *et al.*, "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 5, 2019.

[6] Z. Marian, I.-G. Mircea, I.-G. Czibula, and G. Czibula, "A novel approach for software defect prediction using fuzzy decision trees," in *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2016, pp. 240–247.

[7] H. Ji, S. Huang, Y. Wu, Z. Hui, and C. Zheng, "A new weighted naive Bayes method based on information diffusion for software defect prediction," *Softw. Qual. J.*, vol. 27, no. 3, pp. 923–968, 2019.

[8] M. Hammad, A. Alqaddoumi, and H. Al-Obaidy, "Predicting software faults based on k-nearest neighbors classification," *Int. J. Comput. Digit. Syst.*, vol. 8, no. 5, pp. 462–467, 2019.

[9] P. Kumar and S. K. Singh, "Defect prediction model for aop-based software development using hybrid fuzzy c-means with genetic algorithm and k-nearest neighbors classifier," *Int. J. Appl. Inf. Syst*, vol. 11, no. 2, pp. 26–30, 2016.

[10] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Comput.*, vol. 22, pp. 77–88, 2019.

[11] X. Rong, F. Li, and Z. Cui, "A model for software defect prediction using support vector machine based on CBA," *Int. J. Intell. Syst. Technol. Appl.*, vol. 15, no. 1, pp. 19–34, 2016.

[12] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, and D. Chen, "FECAR: A feature selection framework for software defect prediction," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 426–435.

[13] 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," *Appl. Sci.*, vol. 9, no. 13, p. 2764, 2019.

[14] A. O. Balogun *et al.*, "Impact of feature selection methods on the predictive performance of software defect prediction models: an extensive empirical study," *Symmetry (Basel).*, vol. 12, no. 7, p. 1147, 2020.

[15] I. T. Jolliffe, *Principal Component Analysis*, Ke-2. Springer, 2011.

[16] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013.

[17] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, 2012.

[18] N. V Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[19] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Reflections on the NASA MDP data sets," *IET Softw.*, vol. 6, no. 6, pp. 549–558, 2012.

[20] J. Siswantoro, H. Arwoko, and M. Widiasri, "Indonesian fruits classification from image using MPEG-7 descriptors and ensemble of simple classifiers," *J. Food Process Eng.*, vol. 43, no. 7, pp. 1–13, 2020, doi: 10.1111/jfpe.13414.

[21] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[22] E. Alpaydin, *Introduction to machine learning*, 2nd ed. Cambridge, Massachusetts: MIT press, 2014.

[23] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010, vol. 445, no. 1, pp. 51–56.

[24] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 559–563, 2017.

[25] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.