Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.047172

ARTICLE

Identification of Software Bugs by Analyzing Natural Language-Based Requirements Using Optimized Deep Learning Features

Qazi Mazhar ul Haq[1], Fahim Arif[2,3], Khursheed Aurangzeb[4], Noor ul Ain[3], Javed Ali Khan[5], Saddaf Rubab[6] and Muhammad Shahid Anwar[7,*]

[1]Department of Computer Science and Engineering and International Bachelor's Program in Informatics, Yuan Ze University, Zhongli, Taoyuan, 320315, Taiwan

[2]Department of Computer Science, National University of Pakistan, Rawalpindi, Punjab, 46000, Pakistan

[3]Department of Computer Software Engineering, MCS, National University of Sciences and Technology, Islamabad, 44000, Pakistan

[4]Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, P. O. Box 51178, Riyadh, 11543, Saudi Arabia

[5]Department of Computer Science, School of Physics, Engineering and Computer Science, University of Hertfordshire, Hatfield, AL10 9AB, UK

[6]Department of Computer Engineering, College of Computing and Informatics, University of Sharjah, Sharjah, 27272, United Arab Emirates

[7]Department of AI and Software, Gachon University, Seongnam-SI, 13120, South Korea

*Corresponding Author: Muhammad Shahid Anwar. Email: shahidanwar786@gachon.ac.kr

Received: 27 October 2023 Accepted: 11 December 2023 Published: 26 March 2024

ABSTRACT

Software project outcomes heavily depend on natural language requirements, often causing diverse interpretations and issues like ambiguities and incomplete or faulty requirements. Researchers are exploring machine learning to predict software bugs, but a more precise and general approach is needed. Accurate bug prediction is crucial for software evolution and user training, prompting an investigation into deep and ensemble learning methods. However, these studies are not generalized and efficient when extended to other datasets. Therefore, this paper proposed a hybrid approach combining multiple techniques to explore their effectiveness on bug identification problems. The methods involved feature selection, which is used to reduce the dimensionality and redundancy of features and select only the relevant ones; transfer learning is used to train and test the model on different datasets to analyze how much of the learning is passed to other datasets, and ensemble method is utilized to explore the increase in performance upon combining multiple classifiers in a model. Four National Aeronautics and Space Administration (NASA) and four Promise datasets are used in the study, showing an increase in the model's performance by providing better Area Under the Receiver Operating Characteristic Curve (AUC-ROC) values when different classifiers were combined. It reveals that using an amalgam of techniques such as those used in this study, feature selection, transfer learning, and ensemble methods prove helpful in optimizing the software bug prediction models and providing high-performing, useful end mode.

KEYWORDS

Natural language processing; software bug prediction; transfer learning; ensemble learning; feature selection

1 Introduction

The world is rapidly shifting towards software-based products, drastically increasing our reliance on software. Software houses and companies work hard to meet their needs and develop high-end software products. From autonomous vehicles to face emotion and smart cities, recognition systems are implemented in software-based solutions. Many software houses are creating and doing analysis or decision-making on arguments in user data. However, a software crisis or failure often occurs due to increased complexity, short time to market, and high customer demands. This situation gives rise to successful, error-free software emphasizing quality software production. Many models, principles, and techniques are followed to achieve this notion, such as small iterations, documentation, user interaction, and well-organized processes; still, some inevitable bugs occur, causing great distress to the software users and owners. The unexpected behavior of a system against the provided requirements shows the presence of a bug, and its timely identification facilitates the testing resources allocation efficiently and enables developers to improve the architectural design of a system by identifying the high-risk segments of the system [1], especially since testing the whole software system entirely and thoroughly is practically not possible with the limited testing resources [2]. To mitigate these defects, an analysis of predicting them before they are born is necessary. This inspired the birth of Software Bug Prediction methods, and for this purpose, machine-learning models have proved to be very effective in achieving the required results. These models help narrow down and reduce the testing hardships of faulty modules by identifying such components of software systems, whose chances are more likely to be fault-prone [3].

Prediction models serve a great advantage in development environments as once incorporated, they can give feedback to the developers while they are in the development process. This might give rise to the notion that models are 100% accurate, but in actuality, expecting 100% accuracy of prediction is unreasonable. In such a situation, trusting a model becomes difficult; therefore, it is necessary to get an optimal model, balancing the false predictions. Efforts have been made to examine the accuracy and complexity of models, although there are no standard benchmarks for comparing models. This brings about the compelling utilization of the ensemble method for software bug prediction, as it uses various methods for the provided dataset to give better prediction results. According to observations, various methods have resulted in varied levels of prediction performance, but none of them have consistently delivered the most accurate predictions across various datasets. In this regard, there was a lot of theoretical and empirical evidence in favor of using the ensemble method to get better results for fault prediction. The ensemble method promises to improve fault prediction by reducing the shortcomings of individual methods [3]. One of the key factors for success criteria of bug prediction methods involves the prediction of the correct occurrence of bugs and understanding the software development life cycle (SDLC) process flow. It can be used in any stage of SDLC, be it identifying problems, planning and design, development, testing phase, deployment, and maintenance, irrespective of the type of SDLC model employed [4]. Another way to achieve effectiveness in the domain of defect prediction comes from cross-project prediction, which helps to use a huge amount of labeled data present in the source project to solve different but related issues in the target project [5]. In other words, the information gained from one dataset is used to solve the problems in another dataset [6]. However, the heterogeneity of data from numerous projects becomes a challenging issue in cross-project defect prediction, making it an active research area [7].

Testing the whole software system entirely and thoroughly is practically not possible, especially with the limited testing resources [2]. The unexpected behavior of a system against the provided requirements shows the presence of bugs and promptly identifying them. Developers can efficiently

allocate testing resources and enhance a system's architectural design by determining the high-risk system components [1]. Therefore, it is essential to identify bugs in small sections. Therefore, we devise a novel machine-learning method and evaluate the results using comparison. The objectives of this research are:

- We proposed a hybrid machine learning feature selection and extraction method for software bug prediction of natural language processing-based requirements.
- We propose multiple hand-crafted features with famous machine learning classifiers and the performance is evaluated on open-source datasets.
- We achieved satisfactory performance of Promise and NASA datasets. The performance analysis of machine learning techniques provides the selection of optimal bug prediction methods.

The research involves four major steps: Cleaning the dataset followed by Feature Selection, training the source dataset with individual classifiers, and then using the ensemble learning method to draw a comparison between both the performances and finally testing the performance of the trained model on the target dataset. The remaining structure of the paper is organized as follows. Related work is described in Section 2 of the paper. Section 3 presents the Methodology used for the proposed framework, Section 4 discusses the achieved results, and Section 5 constitutes the Conclusion.

2 Bug Detection through Multiple Machine Learning Methods

This section provides background knowledge and gives an overview of the previous research that has been done on the subject at hand by various authors. With the rise in the demand for software-based products, providing customers with quality products has become critical. Therefore, such methods are smart enough to cater to this requirement. The main objective of software bug prediction is to ensure software quality without wasting resources. Even though many models and frameworks have been proposed, not a single technique comes without its limitations. The majority of the previous research on software fault prediction is restricted to using comparison methods for the analysis of each machine-learning technique. Some of them employed only a few techniques and offered a contrast between them, and others suggested methods as an extension of prior work. Among all the domains, the widely used approach is machine learning. Different machine learning algorithms are used to detect bugs, such as neural networks, support vector machines, and Bayesian networks. Different datasets are available publicly so that the practitioners can easily conduct their experiment without having any worry about data [8].

2.1 Selection and Performance Analysis of Machine Learning Techniques

To enable the practice of machine learning techniques and feature extraction techniques in the context of bug prediction, it is required to review the experimental evidence gained on these techniques through the existing studies [8–14]. The research in [2] carried out proposed an approach to select the minimal number of best-performing software metrics for fault prediction using the Eclipse java development tool (JDT) Core dataset and Linear Regression Model using Marginal R square values. The researchers in [1] performed a comparative performance analysis of multiple machine-learning techniques for software bug prediction. The experiment involved 15 software bug datasets from the PROMISE data repository and used classifiers like Naive Bayes, multi-layer perceptron (MLP), support vector machine (SVM), AdaBoost, Bagging, Decision Tree, Random Forest, J48, k-nearest neigbours (KNN), radial basis kernel (RBF), and K-means [5,15]. Utilized KC1, CM1, PC3, and PC4 from PROMISE software defect dataset repository and found that the proposed hybrid

approach made of Genetic Algorithm and Deep Neural Network (12DNN) classification techniques, performs better than existing classification schemes and facilitates feature optimization, reducing the computational time.

2.2 Addressing Imbalanced Data and Feature Selection

The work in [16] involved a review investigation of the machine learning (ML) technique in semidefinite program (SDP) to highlight the recent and widely used activities in this domain. In [17], the researchers investigated the effect of feature selection and handling imbalanced data in defect prediction while proposing a framework with ensemble learning techniques. They used multiple search methods for feature subset selection. They used six NASA Metrics Data Program (MDP) Datasets. The researchers in [18] used generative deep learning models to manage data unbalancing with the help of Stack Sparse Auto-Encoder (SSAE) and Deep Belief Network (DBN). The experiment in [19] analyzed the impact of a reduced feature set using a hybrid feature selection method (Filter and wrapper FS method) on the performance of the learning algorithm with Naive Bayes (NB), Radial Basis Function Network (RN), J48 using NASA's KC1 data set from promise software engineering repository. Researchers in [20] found that the selection of a representative feature subset or setting a reasonable proportion of selected features improves the performance of Cross-Project Defect Prediction (CPDP), while another ensemble learning method has been proposed as a genetic algorithm in the fish firm [21]. They used feature subset selection and three feature-ranking approaches with K-Nearest Neighbors (KNN) and Naive Bayes (NB) classifiers. The study in [22] applied Synthetic Minority Oversampling Technique (SMOTE) and ensemble techniques on the prediction model to reduce the effect of class imbalance issues and misclassification. Previous methods investigated the effect of deep learning as a Neural Model and compared it with other techniques that improved fault prediction performance significantly [23], Hybrid Particle Swarm Optimization-Modified Genetic Algorithm (PSO- MGA) is used for feature selection and bagging techniques for increasing classification accuracy [24]. While in [25], they proposed an approach to have high performance of defect prediction model and applied three feature selection metaheuristic algorithms to get the most effective features. The experiment was carried out on CM1, MW1, PC1, PC3, and PC4 using SMOTE, Bagging, Ada Boost, and naive Bayes.

The summary of the previous methods with methodology, advantages, and disadvantages is presented in Table 1. Many empirical studies and systematic reviews are also conducted in this research area, such as in [16] and [26], where a review investigation has been performed over the ML technique in SBP to highlight the recent and widely used activities in this domain. They explored different datasets in multiple research studies to provide the latest trends and advances in the ensemble learning approach, concluding that these perform significantly. Many researchers have been made in these fields, and various techniques and methods are proposed. Numerous machine learning algorithms detect bugs, and datasets are made accessible to the public so that researchers can carry out their experiments without worrying about data. Despite all these efforts, the field of software bug prediction still suffers from a lot of ambiguity. Most of these efforts are made in reviews and surveys, highlighting new and widely used trends, proposing different frameworks or methods, discussing various software performance metrics, or suggesting improvements in previous techniques. In all of the above-mentioned ways, machine learning techniques are proving to be most helpful in bug prediction. However, it is necessary to examine the experimental evidence gathered from previous studies to further improve them for enhancing their performance.

Table 1: Summary of related work done in software bug prediction domain

Study	Dataset	Methodology	Advantages	Disadvantages
Hammouri et al. [4]	PROMISE software defect dataset repository KC1, CM1, PC3, PC4	Genetic Algorithm and Deep Neural Network (DNN) classification techniques using MATLAB tool	Found that the proposed hybrid approach performs better than existing classification schemes and facilitates feature optimization reducing the computational time.	Only a few datasets have been used for empirical investigations.
Sohan et al. [16]	Exploring multiple datasets in 165 research studies	Machine Learning Methods	A review investigation has been conducted over the ML technique in SDP to highlight the recent and widely used activities in this domain.	The study lacks details.
Yu et al. [20]	NASA and PROMISE datasets	Feature subset selection and three feature ranking approaches. Classifiers: K Nearest Neighbors (KNN) and Naive Bayes (NB)	Found that the selection of a representative feature subset or setting a reasonable proportion of selected features improves the performance of Cross Project Defect Prediction (CPDP).	The empirical study is not exhaustive. No comparison with other feature selection techniques has been provided.
Khan et al. [23]	PROMISE dataset (CM1, JM1, KC1, KC2, and PC1)	Bayes network, Random Forest, SVM, and the Deep Learning	Investigated the effect of deep learning as a Neural Model and compared it with other techniques that improved fault prediction performance significantly.	The dataset contains unbalanced classes. Lacks preprocessing of the dataset.

(Continued)

Table 1 (continued)

Study	Dataset	Methodology	Advantages	Disadvantages
Banga et al. [24]	NASA MDP Dataset	k-Nearest Neighbor (KNN), Random Forest (RF), SVM	Hybrid Particle Swarm Optimization-Modified Genetic Algorithm (PSO-MGA) is used for feature selection and bagging techniques for increasing classification accuracy.	More datasets of different domains are required to prove the usefulness of the proposed approach.
Ibrahim et al. [25]	PC1, PC2, PC3, and PC4 from PROMISE repository	Random Forest as a classifier and Bat-based search Algorithm (BA) for the feature selection	Proposed an approach to have high performance of defect prediction model and applied three feature selection metaheuristic algorithms to get the most effective features.	No concrete results have been provided.
Akalya Devi et al. [19]	KC1 data set (Promise software engineering repository)	Naive Bayes (NB), J48, Radial Basis Function Network (RN)	Utilized a hybrid feature selection method (Filter and wrapper FS method) to examine the effect of a smaller feature set on the learning algorithm's performance.	The research only made use of one fault prediction dataset.

3 Methodology

In this study, various techniques of Machine Learning are explored to find the optimized ones that will help to elevate the quality of software through software bug prediction. A machine learning model is proposed using the techniques described above on different datasets for exploration purposes. The entire process requires a clean dataset with relevant features, an efficient classifier, and valid training and testing of the model.

1. Initially, the feature selection technique is applied to both datasets to get rid of irrelevant and redundant features.
2. In the next phase, while choosing a machine learning classifier, at the first step, three individual classifiers, i.e., naïve Bayes, Decision Tree (J48), and Multi-Layer Perceptron (MLP) are chosen. These are said to be frequently used classifiers and give good performance in defect prediction [1,16].

3. Later, a combination of the ensemble-learning method and transfer learning is proposed to create the model. Random Forest is implemented for the ensemble method, and the meta classifiers included Bagging and AdaBoost, while the base classifiers employed are mentioned above.
4. The model is then trained on the source project, and for testing, another project is availed. The results achieved are compared to analyze the better-performing technique.

3.1 Preprocessing

A machine learning model requires large data, which is often not in a clean state and needs formatting. In such cases, the researcher carries out data preprocessing, which is a mandatory phase to clean the data and put it into usable format. Therefore, such operations are carried out, which ensure the data is suitable for a machine learning model, hence increasing its accuracy and efficiency. The study in [16] provided NASA datasets in two versions. The version of the dataset known as DS includes inconsistent and duplicated instances, whereas the version known as DS' is cleaner. Originally, these datasets were available on the NASA website; however, they have been removed from this source. NASA datasets consist of 12 clean subsections [20]. We have taken four cleaned and widely used datasets from the available datasets [20], which include CM1, MW1, PC1, and PC2. Previous studies have already discussed and used these cleaned versions of datasets in their experiments. The PROMISE dataset repository comprises four subsections and is provided. They contain 20 object-oriented metrics as independent features and defect-proneness of class as dependent variable. The criteria for cleaning, as stated in [16], is shown in Table 2.

Table 2: Selected features based on dataset

	Dataset name	Selected feature ID	No. of feature selected
NASA	CM1	5,16,18,34,36	5
	MW1	1,3,5,13,18,27,29,35,37	9
	PC1	1,4,5,8,16,17,18,30	8
	PC2	5,17,29	3
PROMISE	Ant-1.7	4,5,6,11,15,18,19	7
	Xalan-2.4	4,5,6,10,11	7
	Camel-1.6	17,19,2,3,4,6,7,9	11
	Ivy-2.0	15,16,17,18,20,1,4,5,8,9,11,13,18	8

3.2 Feature Selection

In recent years, the feature selection technique [27] has been vastly implemented in building software defect prediction models as the high-dimensionality of features can disturb the performance of the model; therefore, we select the features that are relevant to the class. For our study, we have employed feature subset selection using CFS (correlation-based feature selection). To calculate the correlation between each feature ((X_i)) and the target variable ((Y)), ranking the features based on their correlation coefficients and selecting the top features comprise the correlation-based feature selection

process. For linear relationships, the Pearson correlation coefficient ((r)) is commonly used

$$r_{i,Y} = \frac{\sum_{j=1}^{N} \left(X_{i,j} - \overline{X_i}\right)\left(Y_j - \overline{Y}\right)}{\sqrt{\sum_{j=1}^{N} \left(X_{i,j} - \overline{X_i}\right)^2 \sum_{j=1}^{N} \left(Y_j - \overline{Y}\right)^2}} \tag{1}$$

where ($X_{i,j}$) represents the (j)−th instance of feature (X_i), (Y_j) represents the (j)−th instance of the target variable, ($\bar{X}_i$) represents the mean of feature (X_i), ($\bar{Y}$) represents the mean of the target variable, and (N) represents the number of instances.

Rank features based on their absolute correlation coefficient values:

$$Rank\,(X_i) = |Corr(X_i, Y)| \tag{2}$$

For selecting features, we specify a number of top-ranked features ((k)) or a correlation threshold. Features that exceed this threshold are chosen for inclusion in the model:

$$Selected\ features = \left\{X_i \,\middle|\, Rank\,(X_i) \geq Threshold\right\} \tag{3}$$

This process aids in the identification of features that have the strongest linear relationships with the target variable, allowing for dimensionality reduction while retaining the most relevant information for predictive modeling.

It evaluates the redundancy between different features and analyzes their predictive ability. To get the optimal subset of features, the Best-first search is applied. Features with more relevancy to class are preferred over the features irrelevant to other features. While using the datasets in our prediction model, we only use the shared features of each source and target dataset, as the feature numbers are not the same in all datasets. The selected features of each dataset are listed in Table 2. The table represents the dataset name, ID of selected feature, and number of total selected features. The ID list in Tables 2 and 3 shows the ID of selected features for NASA and PROMISE datasets, respectively.

Table 3: Features in PROMISE dataset

ID	Feature name	Feature details
1.	*wmc*	Weighted method per class
2.	*dit*	Depth of inheritance tree
3.	*noc*	Number of children
4.	*cbo*	Coupling between object class
5.	*rfc*	Response for a class
6.	*lcom*	Lack of cohesion in methods
7.	*ca*	Afferent couplings
8.	*ce*	Efferent couplings
9.	*npm*	Number of people methods
10.	*lcom3*	Lack of cohesion in methods
11.	*loc*	Lines of code
12.	*dam*	Data access metric
13.	*moa*	Measure of aggregation
14.	*mfa*	Measure of functional abstraction
15.	*cam*	Cohesion among methods of class

(Continued)

Table 3 (continued)

ID	Feature name	Feature details
16.	*ic*	Inheritance coupling
17.	*cbm*	Coupling between methods
18.	*amc*	Average method complexity
19.	*mac__cc*	Max. McCabe's cyclomatic complexity
20.	*avg__cc*	Avg. McCabe's cyclomatic complexity

The transfer learning technique is unique in its form as it helps save knowledge gathered from solving one problem and applies it to a related but different problem. We can also say that a model created for one task can be used as the basis for another through transfer learning. For example, the knowledge acquired to identify light vehicles can also be applied to identify heavy vehicles. Thus, this technique serves as a great way to solve research problems in the machine learning domain. This study focuses on the cross-company bug prediction situation where source data and target data belong to various companies/projects [28]. The model is built using one project considered as a source project and employed for prediction on another project called the target project [20]. The set of features in both projects is kept the same, but we employ a feature selection approach to reduce irrelevant features.

3.3 Ensemble Technique

The Ensemble learning technique utilizes multiple classifiers in building a classification model to improve the overall performance and efficiency of bug prediction. It also improves the model's generalization capability and decreases the problem of class imbalance. Different data is trained with different classifiers so that each classifier generates its classification error. However, not all classifiers produce the same set of corresponding errors. The ensemble learning methods can reduce biased learning caused by class imbalance classification by combining these classifiers through certain mechanisms. In this scenario, the ensemble learning techniques of boosting (Bst) and bagging (Bag) are widely used [29,30]. The iterative process of ensemble learning uses each model's predictive power to enhance the overall performance of the ensemble. When combining these models, two popular methods are boosting and bagging (also known as bootstrap aggregating). In Bagging, distinct subsets of the training data are used to train Multiple Layer Perceptron (MLP) models independently, adding diversity to capture different patterns in the data. In contrast, boosting improves adaptability by gradually expanding the ensemble through the training of new MLP instances to highlight instances that earlier models misclassified. MLP is combined with both bagging and boosting techniques to produce a hybrid model. Bagging is used to introduce diversity into the training of multiple MLP instances, and Boosting is used iteratively to improve the model's performance by concentrating on difficult instances. The advantages of both approaches are combined in this hybrid method to create a strong, flexible ensemble that improves prediction accuracy. The specific research objectives and dataset characteristics may lead to variations in the implementation details.

The ensemble learning process is characterized by an iterative methodology employing individual models to contribute to an overall ensemble result. Two commonly used strategies for combining these

models are Bagging (Bootstrap Aggregating) and Boosting. The formulation of Bagging (Bootstrap Aggregating) is as follows: The Ensemble prediction (Ebag) is

$$E_{bag}(D) = \frac{1}{M}\sum_{i=1}^{M} E_i(D_i) \tag{4}$$

where M is the total number of models (MLPs), D represents the dataset, and D_i denotes a bootstrap sample for the i-th model. E_i (D_i) is the prediction of the i-th model on its bootstrap sample. The boosting formulation is as follows: Ensemble prediction (Eboost) is

$$E_{boost}(D) = \sum_{i=1}^{M} w_i E_i(D_i) \tag{5}$$

where E_i represents the error of the i-th model and wi is the weight assigned to the i-th model. The weights are adjusted iteratively during the boosting process. Hybrid Model (Combining Bagging and Boosting with MLP) is Ensemble prediction (Ebag-boost) is

$$E_{bag-boost}(D) = \frac{1}{M}\sum_{i=1}^{M}\left(\sum_{J=1}^{T} w_j E_{i,j}(D_i)\right) \tag{6}$$

where T is the number of boosting iterations and $E_{i,j}(D_i)$ represents the error of the j-th iteration of the i-th model, reflecting the model's performance on its bootstrap sample at that iteration. This hybrid approach leverages the diversity introduced by Bagging and the adaptability of Boosting, combining both strategies with Multiple Layer Perceptron models (MLP). The ensemble predictions are formed by averaging or summing the predictions of individual models, with the weights adjusted during boosting iterations. This formulation comprehensively represents the ensemble learning process, incorporating both Bagging and Boosting strategies with MLPs.

3.4 Dataset

The prediction model is assessed on four NASA benchmark datasets (CM1, MW1, PC1, PC2) [31] and four datasets from the PROMISE Repository (ant-1.7, camel-1.6, ivy-2.0 and xalan-2.4) [32]. These datasets are publicly available and consist of historical data of software modules. Different studies have widely used these datasets, and this is the primary reason for our interest in them, as it will help in evaluation. They include several features and a known output class that determines the defectiveness of an instance. Based on data available for other features, the prediction model predicts this output class. The datasets have many projects with various attributes, sizes, and defective rates that help check the research's generality. All the features in the PROMISE repository and NASA dataset have been shown in Tables 3 and 4, respectively.

Table 4: Features in NASA dataset

ID	Feature name
1.	LOC_BLANK
2.	BRANCH_COUNT
3.	CALL_PAIRS
4.	LOC_CODE_AND_COMMENT
5.	LOC_COMMENTS
6.	CONDITION_COUNTS
7.	CYCLOMATIC_COMPLEXITY

(Continued)

Table 4 (continued)

ID	Feature name
8.	CYCLOMATIC_DENSITY
9.	DESIGN_COMPLEXITY
10.	DECISION_COUNT
11.	DESIGN_DENSITY
12.	DESIGN_COMPLEXITY
13.	EDGE_COUNT
14.	ESSENTIAL_COMPLEXITY
15.	ESSENTIAL_DENSITY
16.	LOC_EXECUTABLE
17.	PARAMETER_COUNT
18.	HALSTEAD_CONTENT
19.	HALSTEAD_DIFFICULTY
20.	HALSTEAD_EFFORT
21.	HALSTEAD_ERROR_EST
22.	HALSTEAD_LENGTH
23.	HALSTEAD_LEVEL
24.	HALSTEAD_PROG_TIME
25.	HALSTEAD_VOLUME
26.	MAINTENANCE_SEVERITY
27.	MODIFIED_CONDITION_COUNT
28.	MULTIPLE_CONDITION_COUNT
29.	NODE_COUNT
30.	NORMALIZED_CYCLOMATIC_COMPLEXITY
31.	NUM_OPERANDS
32.	NUM_OPERATORS
33.	NUM_UNIQUE_OPERANDS
34.	NUM_UNIQUE_OPERATORS
35.	NUMBER_OF_LINES
36.	PERCENT_COMMENTS
37.	LOC_TOTAL

The working of the prediction model is evaluated based on certain evaluation criteria, which include Accuracy, Recall, Precision, Area Under Receiver Operating Characteristics Curve (AUC-ROC), F-measure, etc. AUC and Recall, the most widely used performance metric [16], are used in this study. These metrics help to quantify the performance of machine learning models [33]. However, as accuracy is considered a poor performance measure for imbalanced defect data sets, we use the most widely used metric to estimate the performance of each classifier, the area under the ROC curve-AUC as the performance metric in our study.

This section has discussed the creation of a model that involves preprocessing, feature subset selection, transfer learning, and ensemble learning methods. We mentioned the data sets, all of their features, and then the selected features after the feature selection was done. We discussed the classifiers

used in the making of the model and the performance metrics that will used to calculate the efficiency of the model. In short, we have reviewed the technical approach to make the software defect prediction model. The overall framework of the proposed method is presented in Fig. 1.

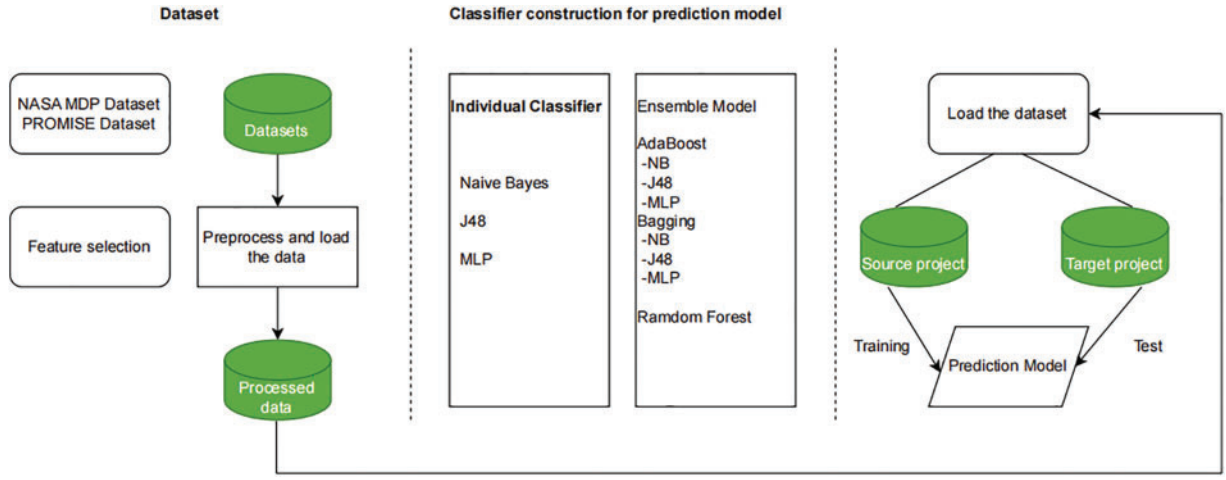

Figure 1: The overall framework architecture of the proposed model. The next section discusses the analysis of the methods implemented in the software bug prediction model

4 Discussion

Software Defect Prediction methods can forecast software bugs in the initial stages of development, increasing the efficiency and performance of the final product. Machine learning models have proved to be very effective in achieving the required results. This section discusses the results given by the model that we have created above and how it can be evaluated. The proposed framework is implemented on four cleaned NASA MDP datasets and four cleaned PROMISE repository datasets by using three individual classifiers and three ensemble methods. The statement 'Source! Target' represents defect prediction from the source project to the target project. For example, 'CM1! MW1' indicates that CM1 is regarded as the source project, and MW1 is regarded as the target project.

We took AUC values as the performance metric instead of accuracy, as accuracy sometimes gives biased results. We can easily evaluate the comparison of performance between the individual classifiers and the ensemble method. Apart from a few cases, we have noticed that when we trained our model with the source dataset and changed the target dataset to test the model, our results with the ensemble method were significantly improved. However, in a few cases, the individual classifier outperformed the ensemble method.

To analyze the performance results of NASA and Promise datasets, Tables 5 and 6 are referred to, respectively. Here again, we can see the performance comparison between the individual classifiers and the ensemble method where we observe the model showed good results with the ensemble method, but the deviation is also seen a few times where the individual classifier outperformed. Rest we see the ensemble method taking charge.

Table 5: Performance evaluation on NASA dataset

Sr. No.	Source ⟶ Target	Individual classifier			Ensemble method						
		NB	J48	MLP	Rf	Adaboost			Bagging		
1.	CM1 ⟶ MW1	0.708	0.500	0.704	0.748	0.564	0.653	0.643	0.775	0.618	0.730
2.	CM1 ⟶ PC1	0.757	0.539	0.750	0.797	0.708	0.651	0.713	0.690	0.584	0.746
3.	CM1 ⟶ PC2	0.210	0.491	0.145	**0.855**	0.486	**0.827**	0.350	0.404	0.584	**0.849**
4.	MW1 ⟶ CM1	0.729	0.446	0.627	0.709	0.694	0.648	0.691	0.725	0.648	0.728
5.	MW1 ⟶ PC1	0.767	0.485	0.536	0.796	0.594	0.668	0.686	0.763	0.673	0.697
6.	MW1 ⟶ PC2	0.786	0.552	0.740	0.798	0.737	0.674	0.570	0.763	0.683	0.690
7.	PC1 ⟶ CM1	0.674	0.580	0.702	0.695	0.583	0.699	0.664	0.689	**0.729**	0.694
8.	PC1 ⟶ MW1	0.755	**0.681**	0.652	0.745	0.563	0.498	0.624	0.748	0.663	0.725
9.	PC1 ⟶ PC2	**0.813**	0.399	**0.762**	0.835	**0.768**	0.681	**0.735**	**0.793**	0.725	0.766
10.	PC2 ⟶ CM1	0.636	0.500	0.601	0.743	0.496	0.754	0.614	0.695	0.664	0.748
11.	PC2 ⟶ MW1	0.559	0.500	0.503	0.652	0.500	0.570	0.530	0.565	0.591	0.762
12.	PC2 ⟶ PC1	0.756	0.500	0.748	0.763	0.508	0.724	0.570	0.767	0.721	0.799

Table 6: Performance evaluation on PROMISE dataset

Sr. No.	Source ⟶ Target	Individual classifier			Ensemble method						
		NB	J48	MLP	Rf	Adaboost			Bagging		
1.	Ant-1.7 ⟶ Camel-1.6	**0.901**	0.814	0.492	0.588	0.554	0.562	0.567	0.599	0.590	0.606
2.	Ant-1.7 ⟶ Xalan-2.4	0.695	**0.820**	0.754	0.752	0.661	0.807	0.717	0.725	0.739	0.777
3.	Ant-1.7 ⟶ Ivy-2.0	0.750	0.740	0.647	0.798	0.740	0.797	0.762	0.784	0.794	0.789
4.	Xalan-2.4 ⟶ Camel-1.6	0.628	0.653	0.705	0.583	0.507	0.563	0.588	0.576	0.603	0.619
5.	Xalan-2.4 ⟶ Ant-1.7	0.799	0.762	0.812	0.800	0.604	0.683	0.738	0.782	0.772	0.813
6.	Xalan-2.4 ⟶ Ivy-2.0	0.810	0.748	0.725	0.705	0.620	0.644	0.776	**0.811**	0.779	0.829
7.	Camel-1.6 ⟶ Ant-1.7	0.725	0.626	0.734	0.652	0.667	0.594	0.656	0.736	0.645	0.688
8.	Camel-1.6 ⟶ Xalan-2.4	0.698	0.603	0.700	0.649	0.620	0.577	0.638	0.708	0.642	0.663
9.	Camel-1.6 ⟶ Ivy-2.0	0.722	0.634	0.689	0.718	0.670	0.658	0.655	0.746	0.649	0.692
10.	Ivy-2.0 ⟶ Ant-1.7	0.769	0.737	0.712	0.776	0.702	0.707	0.669	0.777	0.746	0.768
11.	Ivy-2.0 ⟶ Camel-1.6	0.815	0.801	**0.864**	**0.985**	**0.833**	**0.953**	**0.924**	0.810	**0.952**	**0.905**
12.	Ivy-2.0 ⟶ Xalan-2.4	0.602	0.545	0.584	0.557	0.568	0.498	0.524	0.618	0.595	0.605

4.1 Effect of Individual Classifiers on the Performance of Model

In a machine learning model, data is classified using classifiers that help train the model and then test it on the data to check its performance. For this purpose, in the proposed framework, the model is first built using individual classifiers and observing their efficiency. When the model is trained using the NASA dataset with individual classifiers, we observe certain trends in the performance. The individual classifiers used were Naive Bayes (NB), J48, and Multi-Layer Perceptron (MLP). Out of twelve iterations, they outperformed the ensemble method only two times. Both these times, Naive Bayes (NB) gave better results. This shows that a combination of multiple classifiers can improve the performance of the software bug prediction model. During the training and testing of the Promise

dataset with NB, MLP, and J48, we observed that these individual classifiers again gave better results twice. But this time NB outperformed, and the other time, MLP performed well. The achieved AUC-ROC values were remarkably good.

4.2 Effect of the Ensemble Method on the Performance of the Model

Upon employing Bagging and Boosting on the individual classifiers mentioned above and using Random Forest as the third ensemble method, we tried to investigate how the performance of the model varies with the use of multiple classification methods.

The model, when trained and tested on the NASA dataset using Bagged and Boosted versions of NB, J48, MLP, and Random Forest, we found that Random Forest showed excellent results most of the time. The lowest performance was given with the Boosting algorithm, while Bagged MLP also performed well in three places. The promise dataset showed a different trend. On using dataset projects from the Promise dataset, Bagged NB performed better than all, while RF and Bagged MLP showed good results in two places each.

5 Result Analysis

5.1 Tool Used

In this study, experiments are performed using Weka, an open-source data mining tool under GNU (General Public License), developed in Java language at the University of Waikato, New Zealand. As a collection of machine learning algorithms with a variety of tools for data preparation, classification, regression, clustering, association rules mining, and visualization, this tool has been widely used in data mining studies [34]. Weka is usually preferred due to the ease it provides because of its graphical user interface. It contains numerous algorithms from which any of choice can be selected, their parameters can be tuned, and finally run on the desired dataset. One dataset can have different models applied to it, and the output that meets the requirement can be chosen. Most of the functions are built therefore, the researcher does not have to worry about learning languages and can focus on his work alone. Weka requires the input in the formatting of Attribute Relational File Format while the filename should have the extension of raff. The output received is easily readable and can also be visualized.

5.2 Performance Evaluation on NASA and PROMISE Datasets

The performance of machine learning model is dependent on many factors. Our goal in this study was to check if optimizing the model using multiple classifiers works well. A relative comparison of individual classifiers and ensemble-based methods shows that, generally, the use of multiple classifiers in ensemble methods performed better as compared to the individual ones across all the datasets. However, our results indicate that minor deviations are there which can be corrected with a few more optimizations such as tuning of parameters, class balancing, and feature selection using different techniques. The findings also show that there was a vulnerability in the performance of classifiers, as certain classifiers exhibited good performance in some datasets but compromised in others. This makes it abundantly clear that there is no one predominant classifier, which may be due to the nature of the classifier used to train or test a particular dataset or even the nature of the dataset that was trained on the classifier and also the dataset that was tested with it.

To further review our results, we have performed a box-plot examination of the results obtained. The box-plot analysis is considered a non-parametric test that presents the variation in the samples

without making any assumptions about the statistical distribution of the data [35]. Figs. 2 and 3 demonstrate the results obtained from box-plot analysis for both NASA and Promise datasets. The box plot for the NASA dataset is shown in Fig. 4, and the box plot for the Promise dataset is shown in Fig. 5. The diagrams of the Box-plot yield the minimum, first quartile, maximum, and third quartile values of a sample, whereas the center of the box plot shows the median value. The varying heights, medians, and tails reveal that there are variations in the performance of the classifiers, and no one classifier performed consistently. This increases the odds of considering other factors of a prediction model and exploring those for better results.

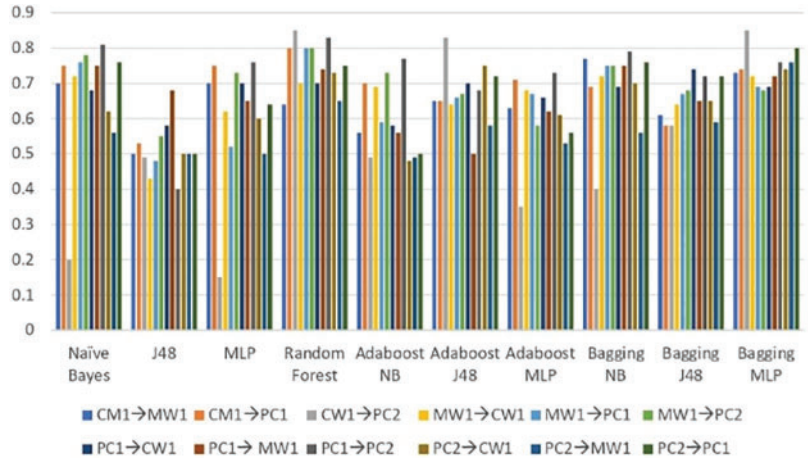

Figure 2: Accuracy of multiple machine learning techniques on PROMISE datasets

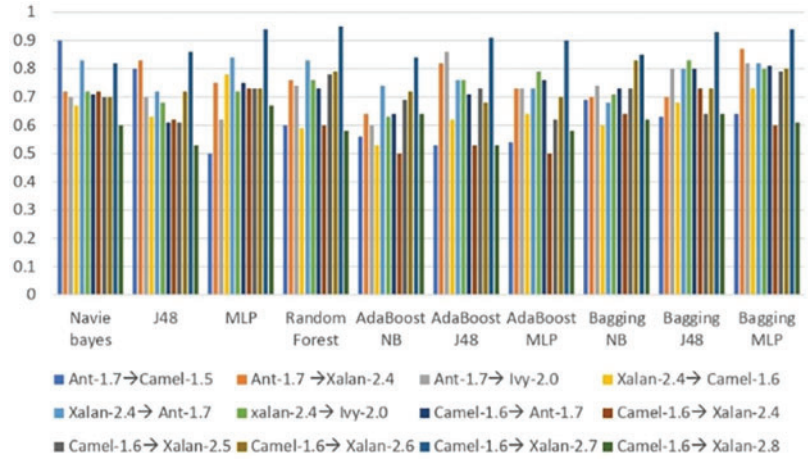

Figure 3: Accuracy of multiple machine learning techniques on NASA datasets

The selected classifiers have been chosen for their different nature of work as Naive Bayes works on probability, J48, which is a trimmed version of C4.5 decision tree [36], MLP, which is an artificial neural network, Random Forest that consists of decisions trees while using bagging and feature randomness during classification, Bagging and Adaboost are the ensemble learning methods. So, these classifiers perform differently yet are strong enough to give us good insight into our model performance. Although with our currently used datasets, J48 and Adaboost have not performed very

well, whereas Bagged NB, Bagged MLP, and Random Forest have given good results. Therefore, our outcomes suggest that the detection of software defects should make use of ensembles as predictive models. The BAR plots, attained by all classifiers, of ROC-AUC scores are shown in Figs. 4 and 5. It tells us there is no single dominant classifier, which may be due to the nature of the datasets. For instance, Random Forest and bagging MLP classifiers performed well on multiple NASA datasets, while Bagging MLP and Naive Bayes classifiers achieved the highest ROC-AUC scores for Promise datasets.

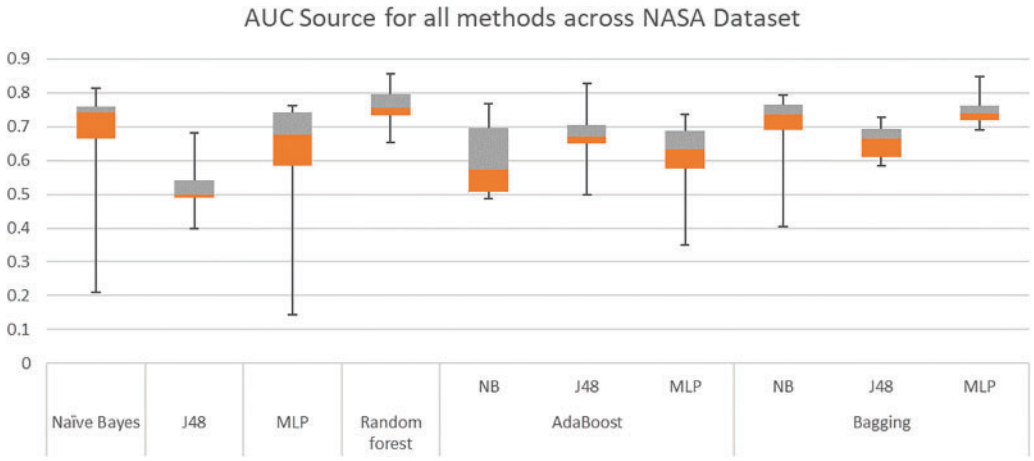

Figure 4: Boxplot NASA dataset

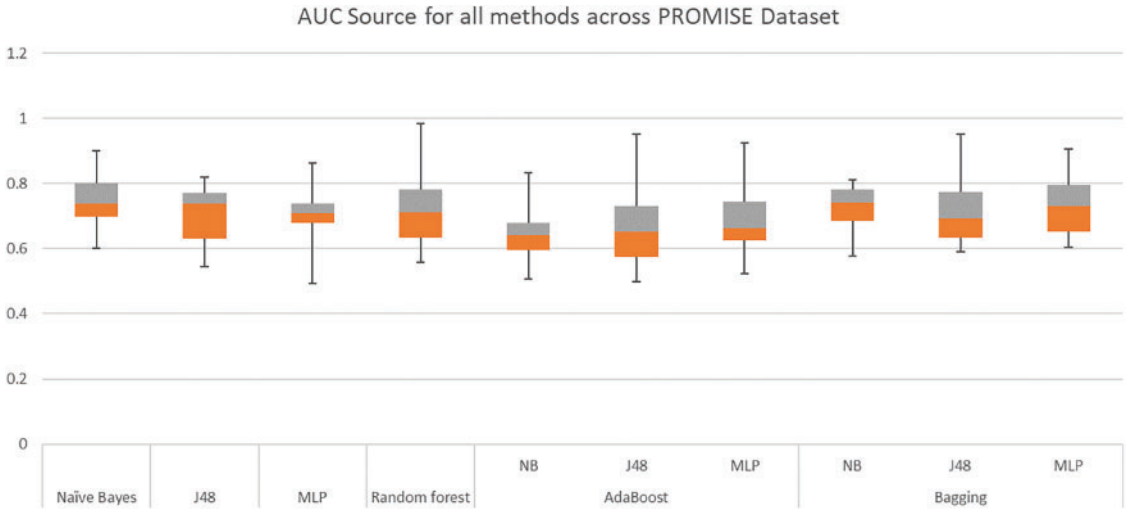

Figure 5: Boxplot PROMISE dataset

6 Conclusion

Software Bug Prediction is an active research area and is being widely explored using machine Learning. This paper proposes a framework based on ML techniques like feature selection, transfer learning, and classification through ensemble methods. The experiment is carried out on four NASA

MDP and four PROMISE datasets. The results revealed that, generally, the use of multiple classifiers in ensemble methods performed better as compared to the individual ones. So, we compared the results by using individual NB, J48, MLP, and ensemble methods that included Random Forest, AdaBoost, and Bagging, where the latter two served as meta classifiers with NB, J48, and MLP as their base classifier. However, our results indicate that minor deviations are there, which reveal different dimensions of a prediction model. Another observation made during the development and analysis is that to get the best results, we need to have a fine-tuned data set, prepare the dataset with good features for the model to be trained and optimized on, and choose excellent classifiers to increase the predictive ability of the model. Future work can be carried out on the above-mentioned dimensions to enhance the predictive ability of the model as multiple factors are involved to create a generalized model.

Acknowledgement: The authors wish to express their appreciation to the reviewers for their helpful suggestions, which greatly improved the presentation of this paper.

Funding Statement: This Research is funded by Researchers Supporting Project Number (RSPD2024R 947), King Saud University, Riyadh, Saudi Arabia.

Author Contributions: Conceptualization, Q. M. Haq, N. Ain, F. Arif; methodology, Q. M. Haq, N. Ain, S. R. F. Arif; software, Q. M. Haq, N. Ain, M. S. Anwar; validation, Q. M. Haq, M. S. Anwar, S. Rubab, F. Arif; formal analysis, Q. M. Haq, M. S. Anwar, J. A. Khan; investigation, Q. M. Haq, M. S. Anwar, S. Rubab, F. Arif, J. A. Khan, K. Aurangzeb; resources, Q. M. Haq, M. S. Anwar, S. Rubab, F. Arif, J. A. Khan, K. Aurangzeb; data curation, Q. M. Haq, N. Ain, S. Rubab, F. Arif; writing—original draft preparation, Q. M. Haq, N. Ain; writing review and editing, Q. M. Haq, M. S. Anwar, S. Rubab, F. Arif, J. A. Khan, K. Aurangzeb; visualization, Q. M. Haq, N. Ain; supervision, Q. M. Haq, M. S. Anwar, S. Rubab, F. Arif, J. A. Khan, K. Aurangzeb; project administration, Q. M. Haq, M. S. Anwar, S. Rubab, F. Arif, J. A. Khan, K. Aurangzeb; funding acquisition, K. Aurangzeb. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Aleem, L. F. Capretz, and F. Ahmed, "Benchmarking machine learning technologies for software defect detection," arXiv preprint arXiv:1506.07563, 2015.

[2] S. Puranik, P. Deshpande, and K. Chandrasekaran, "A novel machine learning approach for bug prediction," *Procedia Comput. Sci.*, vol. 93, pp. 924–930, 2016. doi: 10.1016/j.procs.2016.07.271.

[3] S. S. Rathore and S. Kumar, "Towards an ensemble-based system for predicting the number of software faults," *Expert Syst. Appl.*, vol. 82, pp. 357–382, 2017. doi: 10.1016/j.eswa.2017.04.014.

[4] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 2, pp. 78–83, 2018. doi: 10.14569/IJACSA.2018.090212.

[5] C. Manjula and L. Florence, "Deep neural network-based hybrid approach for software defect prediction using software metrics," *Cluster Comput.*, vol. 22, pp. 9847–9863, 2019. doi: 10.1007/s10586-018-1696-z.

[6] A. Alsaeedi and M. Z. Khan, "Software defect prediction using supervised machine learning and ensemble techniques: A comparative study," *J. Softw. Eng. Appl.*, vol. 12, no. 5, pp. 85–100, 2019. doi: 10.4236/jsea.2019.125007.

[7] Y. Zhang, D. Lo, X. Xia, and J. Sun, "Combined classifier for cross-project defect prediction: An extended empirical study," *Front. Comput. Sci.*, vol. 12, pp. 280–296, 2018. doi: 10.1007/s11704-017-6015-y.
[8] S. N. Saharudin, K. T. Wei, and K. S. Na, "Machine learning techniques for software bug prediction: A systematic review," *J. Comput. Sci.*, vol. 16, no. 11, pp. 1558–1569, 2020. doi: 10.3844/jcssp.2020.1558.1569.
[9] Y. Fan, J. Liu, and S. Wu, "Exploring instance correlations with local discriminant model for multi-label feature selection," *Appl. Intell.*, vol. 52 pp. 8302–8320, 2022. doi: 10.1007/s10489-021-02799-0.
[10] Y. Fan, J. Liu, J. Tang, P. Liu, Y. Lin and Y. Du, "Learning correlation information for multi-label feature selection," *Pattern Recognit.*, vol. 145, pp. 109899, 2024. doi: 10.1016/j.patcog.2023.109899.
[11] Y. Fan, J. Liu, P. Liu, Y. Du, W. Lan and S. Wu, "Manifold learning with structured subspace for multi-label feature selection," *Pattern Recognit.*, vol. 120, pp. 108169, 2021. doi: 10.1016/j.patcog.2021.108169.
[12] Y. Fan, B. Chen, W. Huang, J. Liu, W. Weng and W. Lan, "Multi-label feature selection based on label correlations and feature redundancy," *Knowl.-Based Syst.*, vol. 241, pp. 108256, 2022. doi: 10.1016/j.knosys.2022.108256.
[13] Y. Fan, J. Liu, W. Weng, B. Chen, Y. Chen and S. Wu, "Multi-label feature selection with local discriminant model and label correlations," *Neurocomput.*, vol. 442, pp. 98–115, 2021. doi: 10.1016/j.neucom.2021.02.005.
[14] Y. Fan, J. Liu, W. Weng, B. Chen, Y. Chen and S. Wu, "Multi-label feature selection with constraint regression and adaptive spectral graph," *Knowl.-Based Syst.*, vol. 212, pp. 106621, 2021. doi: 10.1016/j.knosys.2020.106621.
[15] Q. M. ul Haq, C. H. Lin, S. J. Ruan, and D. Gregor, "An edge-aware based adaptive multi-feature set extraction for stereo matching of binocular images," *J. Ambient Intell. Humaniz. Comput.*, vol. 2022, no. 17, pp. 1–15, 2023. doi: 10.1007/s12652-021-02958-8.
[16] M. F. Sohan, M. A. Kabir, M. Rahman, T. Bhuiyan, M. I. Jabiullah and E. A. Felix, "Prevalence of machine learning techniques in software defect prediction," in *Cyber Secur. Comput. Sci.: Second EAI Int. Conf., ICONCS 2020*, Feb. 15–16, 2020, Dhaka, Bangladesh, Springer, pp. 257–269.
[17] F. Matloob, S. Aftab, and A. Iqbal, "A framework for software defect prediction using feature selection and ensemble learning techniques," *Int. J. Modern Educ. Comput. Sci.*, vol. 11, no. 12, pp. 14–20, 2019. doi: 10.5815/ijmecs.2019.12.02.
[18] A. Hasanpour, P. Farzi, A. Tehrani, and R. Akbari, "Software defect prediction based on deep learning models: Performance study," arXiv preprint arXiv:2004.02589, 2020.
[19] C. Akalya Devi, K. E. Kannammal, and B. Surendiran, "A hybrid feature selection model for software fault prediction," *Int. J. Comput. Sci. Appl.*, vol. 2, pp. 25–35, 2012. doi: 10.5121/ijcsa.2012.2203.
[20] Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, "An empirical study on the effectiveness of feature selection for cross-project defect prediction," *IEEE Access*, vol. 7, pp. 35710–35718, 2019. doi: 10.1109/ACCESS.2019.2895614.
[21] P. W. Khan and Y. C. Byun, "Optimized dissolved oxygen prediction using genetic algorithm and bagging ensemble learning for smart fish farm," *IEEE Sens. J.*, vol. 23, no. 13, pp. 15153–15164, 2023. doi: 10.1109/JSEN.2023.3278719.
[22] A. Saifudin, S. W. H. L. Hendric, B. Soewito, F. L. Gaol, E. Abdurachman and Y. Heryadi, "Tackling imbalanced class on cross-project defect prediction using ensemble smote," in *IOP Conf. Series: Mater. Sci. Eng.,* IOP Publishing, 2019, vol. 662, pp. 062011. doi: 10.1088/1757-899X/662/6/062011.
[23] R. U. Khan, S. Albahli, W. Albattah, and M. N. I. Khan, "Software defect prediction via deep learning," *Int. J. Innov. Technol. Explor. Eng.*, 2020, vol. 9, no. 5, pp. 343–349. doi: 10.35940/ijitee.D1858.039520.
[24] M. Banga and A. Bansal, "Proposed software faults detection using hybrid approach," *Secur. Privacy*, vol. 6, no. 4, pp. e103, 2023. doi: 10.23940/ijpe.19.08.p4.20492061.
[25] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *2017 Int. Conf. New Trends Comput. Sci. (ICTCS)*, IEEE, 2017, pp. 252–257.
[26] F. Matloob *et al.*, "Software defect prediction using ensemble learning: A systematic literature review," *IEEE Access*, vol. 9, pp. 98754–98771, 2021. doi: 10.1109/ACCESS.2021.3095559.

[27] S. Khan, I. Ali, F. Ghaffar, and Q. M. U. Haq, "Classification of macromolecules based on amino acid sequences using deep learning," *Eng., Technol. Appl. Sci. Res.*, vol. 12, no. 6, pp. 9491–9495, 2022. doi: 10.48084/etasr.5230.

[28] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Inform. Softw. Technol.*, vol. 54, no. 3, pp. 248–256, 2012. doi: 10.1016/j.infsof.2011.09.007.

[29] S. Qiu, L. Lu, S. Jiang, and Y. Guo, "An investigation of imbalanced ensemble learning methods for cross-project defect prediction," *Int. J. Pattern. Recogn.*, vol. 33, no. 12, pp. 1959037, 2019. doi: 10.1142/S0218001419590377.

[30] Q. M. U. Haq, L. Yao, W. Rahmaniar, Fawad and F. Islam, "A hybrid hand-crafted and deep neural spatio-temporal EEG features clustering framework for precise emotional status recognition," *Sens*, vol. 22, no. 14, pp. 5158, 2022. doi: 10.3390/s22145158.

[31] 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. doi: 10.1109/TSE.2013.11.

[32] S. Karim *et al.*, "Software metrics for fault prediction using machine learning approaches: A literature review with promise repository dataset," in *2017 IEEE Int. Conf. Cybern. Comput. Intell. (CyberneticsCom)*, 2017, pp. 19–23.

[33] E. K. Ampomah, Z. Qin, and G. Nyame, "Evaluation of tree-based ensemble machine learning models in predicting stock price direction of movement," *Inform.*, vol. 11, pp. 332, 2020. doi: 10.3390/info11060332.

[34] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, "Data mining: Practical machine learning tools and techniquesâ," in *Morgan Kaufmann Ser. Data Manag. Syst.*, 2005.

[35] N. Saravanan and V. Gayathri, "Performance and classification evaluation of j48 algorithm and Kendallâs based J48 algorithm (KNJ48)," *Int. J. Comput. Trends Technol.*, vol. 59, no. 2, pp. 73–80, 2018. doi: 10.14445/22312803/IJCTT-V59P112.

[36] Y. Benjamini, "Opening the box of a boxplot," *Am. Stat.*, vol. 42, no. 4, pp. 257–262, 1988. doi: 10.1080/00031305.1988.10475580.