Received: 10 March 2025 | Revised: 22 April 2025 | Accepted: 12 May 2025 | Published online: 27 May 2025

RESEARCH ARTICLE

Smart Wearable Technology
2025, Vol. 1, A9, 15 Pages
DOI: 10.47852/bonviewSWT52025645

BON VIEW PUBLISHING

Software Defect Prediction Using Traditional Machine Learning and Ensemble Learning Algorithms

S. M. Hasan Kabir[1], Md. Tanim Rahman[1] and Aunik Hasan Mridul[2,*]

[1]*Software Engineering, Daffodil International University, Bangladesh*

[2]*Computer Science and Engineering, Daffodil International University, Bangladesh*

Abstract: Software defect prediction (SDP) leverages machine learning to identify potential defects in software systems, enhancing software quality and reducing maintenance costs. Traditional techniques like Naïve Bayes, Decision Trees, and support vector machines (SVM) have been widely used for defect prediction due to their simplicity and interpretability. However, ensemble techniques such as Random Forest (RF), AdaBoost, and Bagging have gained prominence for their ability to improve accuracy by combining multiple models. While these methods have demonstrated promising results, challenges like overfitting, imbalanced datasets, and inadequate model tuning remain critical for further improvement in SDP. There are various types of datasets available in online. We have selected four datasets. We employed RF, Logistic Regression (LR), Gradient Boosting (GB), K-Nearest Neighbor (KNN), Decision Tree (DT), and XGB Classifier. In order to thoroughly investigate and assess each model's predictive potential for identifying software defects and the issues they cause, we also used ensemble approaches. The results were promising, with the KNN achieving a noteworthy test accuracy of 87.08% for Dataset PC1, KNN of 76.49% for dataset JM1, KNN of 81.45% for dataset KC1 and GB of 92.89% for dataset CM1. Furthermore, Boosting GB also demonstrated accuracy matching this high standard of 94.5% for dataset CM1. To further enhance performance, we implemented a range, including Bagging, Boosting, Stacking, and Voting algorithms, optimizing each classifier with the best parameters through hyperparameter tuning. Through our experimental investigation, we not only contributed to the body of knowledge on SDP but also identified the Gradient Boosting with Boosting model as the most accurate, achieving an outstanding accuracy rate of 94.5% for SDPs. This research endeavors to provide invaluable insights into software defect management, offering a potential solution for early intervention and ultimately improving software outcomes.

Keywords: software defect, bagging, boosting, ensemble, machine learning, recall

1. Introduction

Software defect prediction (SDP) has become an essential aspect of software engineering, playing a crucial role in identifying defective modules or components during the software development process [1]. The early detection of software defects can significantly enhance software quality, reduce development costs, and ensure timely delivery of projects. With the increasing complexity and size of software systems, manual approaches to defect identification have proven inefficient. As a result, machine learning (ML) techniques have emerged as a promising solution to automate the defect prediction process. These techniques are capable of learning from historical data, extracting meaningful patterns, and making predictions about future defect occurrences [2].

The traditional ML approaches used for SDP include algorithms such as Naïve Bayes, Decision Trees (DT), support vector machines (SVM), K-Nearest Neighbor (KNN), and artificial neural networks (ANN). These algorithms have been widely applied due to their simplicity, interpretability, and ability to handle diverse datasets. For instance, Naïve Bayes is favored for its efficiency in handling large datasets and its probabilistic approach to classification, while DTs offer an intuitive model structure that is easy to interpret and visualize [3]. SVM, on the other hand, is known for its robustness in high-dimensional spaces, making it suitable for defect prediction tasks where a large number of features are involved [4].

Despite the success of these traditional ML techniques in defect prediction, they are often limited by challenges such as overfitting, imbalanced datasets, and poor generalization across different software projects. Overfitting occurs when a model becomes too complex, fitting the training data too closely and failing to generalize to new data [5]. Imbalanced datasets, where the number of defective modules is much smaller than non-defective ones, can cause models to be biased toward the majority class, resulting in poor predictive performance for the minority (defective) class. Moreover, the performance of traditional ML models can vary significantly when applied to different datasets, making it difficult to develop a generalizable solution [6].

To address these limitations, researchers have increasingly turned to ensemble techniques for SDP. Ensemble methods

*Corresponding author: Aunik Hasan Mridul, Computer Science and Engineering, Daffodil International University, Bangladesh. Email: Aunik15-2732@diu.edu.bd

combine the predictions of multiple base models to create a more accurate and robust predictor. The underlying idea behind ensemble learning is that a group of weak learners can be combined to form a strong learner. This approach helps mitigate the issues of overfitting and variance in predictions, leading to improved accuracy and stability.

There are several popular ensemble techniques used in SDP, including Bagging, Boosting, and Stacking. Bagging (Bootstrap Aggregating) is an ensemble technique that trains multiple models on different subsets of the training data, obtained through bootstrapping [7]. Each model makes a prediction, and the final output is determined by aggregating the predictions (e.g., through majority voting). Random Forest (RF), a widely used bagging-based technique, consists of an ensemble of DTs. By combining the predictions of multiple trees, RF reduces the variance in predictions and improves generalization [8].

Boosting, another prominent ensemble method, focuses on creating a strong model by sequentially training weak models. In Boosting, each subsequent model tries to correct the errors made by the previous models. The models are trained in a weighted manner, with more emphasis placed on the misclassified instances. AdaBoost and Gradient Boosting (GB) are two popular variants of Boosting that have been successfully applied to SDP. AdaBoost combines multiple weak classifiers (typically decision stumps) to create a strong classifier, while GB builds models in a stage-wise manner to optimize a loss function [9].

Stacking is a more advanced ensemble technique that involves training multiple base learners and combining their predictions using a meta-learner. The base learners are trained on the original dataset, and their predictions are used as input to the meta-learner, which makes the final prediction. Stacking allows the model to capture diverse patterns from different algorithms and leverage their strengths to improve predictive performance [10].

Ensemble methods have shown great promise in overcoming the limitations of traditional ML techniques. By combining multiple models, ensemble methods reduce the risk of overfitting, handle imbalanced datasets more effectively, and produce more stable and accurate predictions [11]. For example, RF, an ensemble of DTs, is less prone to overfitting than a single DT and can achieve higher accuracy in defect prediction tasks. Similarly, Boosting techniques like AdaBoost and GB have been shown to improve the performance of weak classifiers by focusing on the difficult-to-predict instances, resulting in more accurate predictions [12].

However, ensemble techniques are not without their challenges. One of the main issues with ensemble methods is their computational complexity. Training multiple models and combining their predictions can be computationally expensive, especially for large datasets. This can make ensemble techniques less feasible for real-time defect prediction tasks or for applications where computational resources are limited [13]. Additionally, ensemble methods can be more difficult to interpret than traditional ML models. While a single DT is easy to visualize and understand, a RF consisting of hundreds of trees can be much more complex to analyze and interpret [14].

Despite these challenges, the advantages of ensemble techniques in terms of predictive accuracy and robustness have made them a popular choice for SDP. Several studies have demonstrated the superior performance of ensemble methods over traditional ML techniques in SDP [15]. For instance, RF has been shown to outperform individual DTs, SVM, and Naïve Bayes in terms of both accuracy and stability. Similarly, Boosting techniques have been found to achieve higher accuracy than traditional models by focusing on the difficult-to-classify instances [16].

Both traditional ML techniques and ensemble methods have been widely applied to SDP, each with its strengths and limitations. Traditional models such as Naïve Bayes, DTs, and SVM are simple and easy to interpret, but they can suffer from overfitting, imbalanced datasets, and poor generalization across datasets. Ensemble techniques, on the other hand, address these issues by combining multiple models to create a more robust and accurate predictor. While ensemble methods can be computationally expensive and difficult to interpret, their superior performance in terms of predictive accuracy and robustness makes them an attractive option for SDP tasks. As the field of ML continues to evolve, further research into improving the efficiency and interpretability of ensemble methods is likely to lead to even more effective solutions for SDP [17].

2. Literature Review

ML methods play a crucial role in identifying the intricate architecture of PCOS disease, focusing on the evaluation of patient diagnosis reports. Various techniques, such as GS, RF, Logistic Regression (LR), GB, KN, ABC, and DT, are employed for the exploratory analysis in this field. Researchers have extensively employed a range of models, as discussed in this segment, to enhance our understanding of PCOS.

Recent studies on SDP using ML have focused on a variety of techniques to improve accuracy and model performance, though several have encountered limitations that hindered their ability to achieve accuracy rates above 94%. These studies offer critical insights into the challenges faced when applying different ML algorithms, feature selection methods, and ensemble techniques to SDP tasks. Despite their diverse methodologies, the accuracy of these models remained below the 94% mark due to factors such as data preprocessing, imbalanced datasets, overfitting, and insufficient feature extraction, among other issues.

Ronchieri et al. [18] adopted unsupervised ML methods using unlabeled datasets, introducing two models—CLAMI and CLAMI+—which operate independently of metric thresholds and require minimal expert intervention. Their findings indicated that Bagging, Boosted LR, and Adaptive Boost techniques yielded the highest average accuracy across datasets, showing potential for practical implementation in defect localization [18].

Similarly, Assim et al. [19] presented an SDP model on the Weka 3.8.3 platform using eight ML algorithms, drawing on historical software data. Their evaluation revealed that ensemble models consistently performed well, with the SMOreg algorithm achieving the best results, while the ANN model underperformed. Their comparative analysis suggested that integrating multiple classifiers enhances defect detection accuracy [19].

Qiao et al. [20] emphasized the importance of data preprocessing—specifically log transformation and normalization—before feeding it into a neural network-based prediction model. Applied to two benchmark datasets, their approach significantly reduced mean square error by over 14% and improved the squared correlation coefficient by more than 8%, demonstrating the model's robustness and accuracy in defect forecasting [20].

In a novel approach, Chen et al. [21] visualized software as images for direct input into convolutional neural networks (CNNs), leveraging self-attention and cross-project learning. Their image-based method eliminated the need for feature extraction tools. Experimental results from ten open-source projects

confirmed the model's effectiveness in defect detection, validating deep learning's capabilities in this domain [21].

Hasanpour et al. [22] introduced deep learning methods—Stacked Sparse Auto-Encoder (SSAE) and Deep Belief Network—to manage the challenge of imbalanced and small-sized NASA datasets. SSAE was noted for its superior generalization ability, although performance varied across datasets due to data limitations. SSAE outperformed traditional techniques in several test cases [22].

Jin [23] addressed class imbalance by applying cost-sensitive learning to the Large Margin Distribution Machine (LDM), creating the CS-ILDM model. Evaluated on five datasets, the model demonstrated a strong performance in mitigating misprediction costs while maintaining high accuracy.

Khan et al. [24] assessed various ML models on healthcare datasets using metrics such as mean absolute error, recall, and accuracy. RF achieved the highest performance with an average accuracy of 88.32% and a rank value of 2.96, closely followed by SVM and Creedal DT, affirming RF's predictive superiority [24].

Zhang et al. [25] compared LR, SVM, and back propagation neural network models for SDP. Among these, SVM provided the highest prediction accuracy based on combined values of Precision, Recall, and F-measure, despite challenges in parameter optimization and model stability [25].

Shi et al. [26] introduced a unique representation of source code using Multi-Perspective Tree Embedding, which extracted nodes from abstract syntax trees (ASTs) across three distinct views. Their unsupervised embedding method improved the model's ability to detect defects by encoding diverse structural and semantic information, offering enhanced expressiveness and adaptability to different software projects [26].

Rahim et al. [27] proposed a three-phase SDP framework involving data preprocessing, feature extraction, and ML-based classification. Using Naive Bayes (NB) and linear regression, their approach reached a remarkable prediction accuracy of 98.7% with the NB classifier, highlighting the importance of efficient preprocessing and feature selection [27].

Lin and Lu [28] developed a framework called Feature Learning via Dual Sequences, which generated semantic features from ASTs using bi-directional long short-term memory networks. By combining semantic and structural attributes, the method showed superior performance across eight open-source Java projects, outperforming several existing SDP models in terms of predictive accuracy [28].

Chen et al. [29] employed neighbor cleaning and ensemble under-sampling strategies to develop a robust model tested on nine highly imbalanced datasets. Their ensemble approach demonstrated enhanced accuracy compared to standard classifiers and preprocessing techniques [29].

Hammouri et al. [30] applied NB, DT, and ANN models across three real-world debugging datasets, concluding that DT provided superior prediction accuracy.

Additionally, Matloob et al. [31] conducted a systematic literature review on ensemble learning methods for SDP and found that ensemble strategies consistently outperformed single classifiers. The study highlighted the importance of classifier diversity and feature selection to maximize model performance [31]. Table 1 shows the comparative analysis with previous work.

3. Research Methodology

A training-and-testing-based supervised learning approach was applied in this work. The training dataset, from which the algorithm discovered patterns and correlations, was used to build the classification model. The trained model was then used to categorize new occurrences or predict outcomes on the testing dataset. We have created a number of classifiers, including the RF, LR, GB, KN, XGB, and DT techniques.

3.1. RF

For problems involving regression and classification, an ensemble learning technique called a RF is commonly employed. It works by constructing several DTs during training and generating the average prediction (regression) or grouping (classification) of every single tree. The "forest" is constructed using a process known as bagging (Bootstrap Aggregating), in which a DT is taught on each of the dataset's many subsets that are produced via replacement. This method averages out variances and biases from individual trees, which helps to decrease overfitting and increase the model's generalization. The usefulness of RFs in producing precise forecasts without requiring substantial parameter adjustment is well-known, as is their robustness and capacity to handle big datasets with increased dimensionality [32].

Table 1
Comparative analysis with previous work

SL No	Author name	Used algorithm	Best accuracy
1	Ronchieri et al. [18]	CLAMI and CLAMI+	98.87%
2	Assim et al. [19]	MPT-Embedding	80%
3	Qiao et al. [20]	Machine Learning	84%
4	Chen et al. [21]	PROM, LSTM, CNN	64.2%
5	Hasanpour et al. [22]	SSAE, DBN	99%
6	Jin [23]	CSL, LDM, CS-ILDM	80.9%
7	Khan et al. [24]	RF, SVM, CDT	88.32%
8	Zhang et al. [25]	LR, SVM, BPNN	85.8%
9	Shi et al. [26]	MPT, ASTs	63.57%
10	Rahim et al. [27]	NB, Linear Regression	98.7%
11	Lin and Lu [28]	FLDS, BiLSTM	68.4%
12	Chen et al. [29]	Ensemble Techniques	74.9%
13	Hammouri et al. [30]	NB, DT, ANN	97.1%
14	Matloob et al. [31]	Ensemble learning	98%

3.2. DT

Examining the attribute that the base of the tree node represents is the first step in categorizing an instance. Next, we follow the branch of the structure that corresponds to the value of that characteristic. With just two number classes needed, the DT methodology is among the most widely used and effective prediction techniques. Each inner node of a DT, a data structure with an ordered structure where each node in the leaf hierarchy indicates a distinct class, represents an attribute test. On the basis of DTs, a tree structure known as DT is frequently employed. This approach can handle classification and regression issues. The tree starts at the root node and uses the "splitting" method to select the "Best Features" or "Best Attributes" from the pool of possible qualities. Entropy examines a dataset's consistency, whereas data collection gauges how quickly changes in an attribute's volatility occur [33].

3.3. XGB classifier

The XGBoost (Extreme GB) classifier is a powerful and efficient implementation of the GB framework, specifically designed for supervised learning tasks. It excels in predictive performance by combining an ensemble of weak prediction models, typically DTs, to create a robust predictive model. XGBoost offers several key advantages, including handling missing values, incorporating regularization to prevent overfitting, and leveraging advanced tree learning algorithms that optimize speed and performance. It is a well-liked option for many applications because of its scalability and versatility, from structured/tabular data to more complex domains, offering superior accuracy and efficiency in comparison to many traditional ML algorithms [34].

3.4. LR

Modeling the likelihood of a binary result using one or more predictor variables, LR is a method of statistics used for binary classification. Typically coded as 0 or 1, LR predicts the chance that an instance belongs to a certain category, as opposed to linear regression's prediction of a continuous result. This is accomplished by converting the linear sum of the given input variables into an amount between 0 and 1 using the logistic function, commonly referred to as the sigmoid function. The model provides coefficients that indicate the influence of each predictor and calculates the likelihood that the dependent event will occur [35, 36]. In order to predict the existence of diseases, consumer behavior, and other things, LR is often utilized in the social sciences, medical, and ML domains.

3.5. GB

For regression and classification applications, GB is a potent ML approach that creates an ensemble of weak learners, usually DTs. In order to fix the mistakes produced by earlier models, iteratively adding new models while concentrating on the data points that are most difficult to forecast correctly is how it operates. Every new model is trained to gradually minimize the total prediction error by reducing the residual errors of the aggregate ensemble. Because GB uses a sequential technique, it may produce very precise prediction models. This makes it very useful for managing complicated datasets and identifying minute trends. To avoid overfitting and maximize performance, nevertheless, meticulous hyperparameter adjustment is necessary [37].

3.6. K-Nearest

A straightforward, non-parametric, and user-friendly technique for classification and regression problems in ML is the KNN classifier. It functions by finding the k instances in the training dataset that are the closest to a given input (neighbors) and then predicting the output based on the average value (for regression) or majority class (for classification) of these neighbors. Distance measures like Euclidean distance are commonly used to quantify similarity. Even though k-NN is straightforward, it can be effective, particularly for short datasets or in situations where the decision border is extremely irregular. It is, however, sensitive to the choice of k and the distance measure and computationally costly for big datasets. For data to perform better, feature selection and data scaling must be done correctly.

3.7. Ensemble learning algorithms

Combining many ML models to increase overall prediction accuracy and resilience is known as ensemble learning [38]. Ensemble learning methods like Boosting and Bagging offer superior predictive performance in SDP by aggregating the strengths of multiple base models. These approaches reduce variance (Bagging) and bias (Boosting), leading to more robust and generalized models. In the context of SDP, where software quality and reliability are paramount, these techniques outperform traditional models by handling noisy, imbalanced, and high-dimensional data more effectively. They also improve recall and precision, which are critical in identifying actual defect-prone modules without too many false positives or negatives.

3.7.1. Bagging classifier

Bootstrap Aggregating, or Bagging, is a strategy for ensemble learning that aims to increase the precision and resilience of ML models. It entails using several subsets of the training data produced by bootstrap sampling—random sampling with replacement—to train numerous base models, usually DTs. Every model undergoes separate training, and for classification tasks, the predictions are aggregated by a majority vote, and for regression tasks, through averaging. Bagging, as opposed to using a single model, produces more consistent and dependable performance by combining the predictions of many models, which lowers variance and helps prevent overfitting. One of the most well-known uses of this technique is RF, which expands bagging by adding more randomization to the characteristics chosen for each split in the trees [39].

3.7.2. Boosting classifier

An effective ML method that builds a strong overall model by combining several weak classifiers is called a boosting ensemble classifier. Boosting is based on the sequential training of classifiers, where each new model learns from the mistakes made by the prior ones. This is usually accomplished by giving misclassified cases larger weights, which incentivizes the subsequent classifier in the series to fix the errors. AdaBoost and GB are two popular boosting methods that iteratively modify the weights of the classifiers and training data. Boosting improves the final classifier's accuracy and resilience by combining the predictions of all the models, frequently beating the performance

of individual models, especially in situations with complicated patterns and noisy data [40].

3.7.3. Stacking classifier

An effective ML method that mixes many base models to increase prediction accuracy is the stacking ensemble classifier. This approach involves training several learning algorithms on the same dataset so they can each generate unique predictions. The final prediction is then produced by a second-level meta-model using these predictions as input attributes. By leveraging the strengths and compensating for the weaknesses of the individual base models, stacking often results in superior performance compared to any single model. This approach effectively reduces overfitting and bias, making it particularly useful in complex tasks where no single model performs optimally across all scenarios [41].

3.7.4. Voting classifier

A ML model called a voting ensemble classifier aggregates the predictions of several different classifiers to increase overall resilience and accuracy. The way the ensemble functions is by using a voting mechanism, which can be either soft or hard voting, to aggregate the predictions of its component models. Hard voting selects the class with the majority of votes as the final forecast, with each classifier voting for a particular class label. During the soft voting process, the class with the greatest average probability is selected by averaging the projected probabilities of each class from all classifiers. This approach leverages the strengths of diverse models, reducing the likelihood of errors and increasing the performance, especially in complex and noisy datasets. By pooling the decisions of several models, a voting ensemble can achieve better generalization compared to any individual model alone [42–44].

3.8. Flow chart

The research utilizes four datasets sourced from Kaggle. First, the datasets we selected were put into use. By locating and updating any inaccurate or missing numbers, we guaranteed the integrity of the data. Next, we employed a range of algorithms and assessed their performance. It employs a variety of algorithm models, including GB, KNN, XGB Classifier (XGB), RF, LR, and DT, to predict software defect risks. To enhance prediction accuracy, the study also incorporates ensemble models such as Bagging, Boosting, Stacking, and Voting. We resorted to ensemble algorithms, which include bagging, boosting, stacking, and voting, in order to maximize our forecast accuracy. This enabled us to get the most out of the integrated algorithms and produce thoroughly examined outcomes. To find out how well the models used in this phase predicted software defects, they were put through an outcome analysis. The study applied Grid Search and Cross-Validation methods to fine-tune the hyperparameters of the classifiers. Parameters such as the number of estimators, learning rate, and maximum depth in boosting models were optimized. This tuning significantly influenced the final results by ensuring each model performed at its best possible configuration, thus enhancing accuracy, minimizing overfitting, and improving generalization across datasets. The model, depicted in Figure 1, provided insights into the most successful methods used in our study and summarized our research path.

3.9. Data collection and preprocessing

The datasets used in this study—PC1, JM1, KC1, and CM1—were selected from the NASA MDP (Metrics Data Program) repository. These datasets are widely recognized benchmarks in defect prediction research due to their detailed software metric attributes and real-world relevance. The variation in dataset size, the ratio of defective to non-defective instances, and the distribution of feature values significantly impacted model accuracy. For example, JM1 is larger and more complex, often leading to different prediction dynamics compared to smaller datasets like KC1 or CM1. The data were stored in GitHub nearly prepared for utilization. Dataset CM1 has 498 rows, 22 columns, dataset JM1 contains 10880 columns and 22 rows, dataset KC1 contains 2109 columns and 22 rows and dataset PC1 contains 1109 columns and 22 rows [45]. The column "Defect" is responsible for categorizing SDP prevalence. Each attribute in the dataset played a vital role in SDP, where target was classified into two groups. The datasets were then divided into two subsets: one for testing (20%) and the other for training (80%) [46]. The target column has been balanced using ADASYN. Figure 2 shows the count plot of CM1 dataset, Figure 3 for JM1 dataset, Figure 4 for KC1 dataset, and Figure 5 for PC1 dataset.

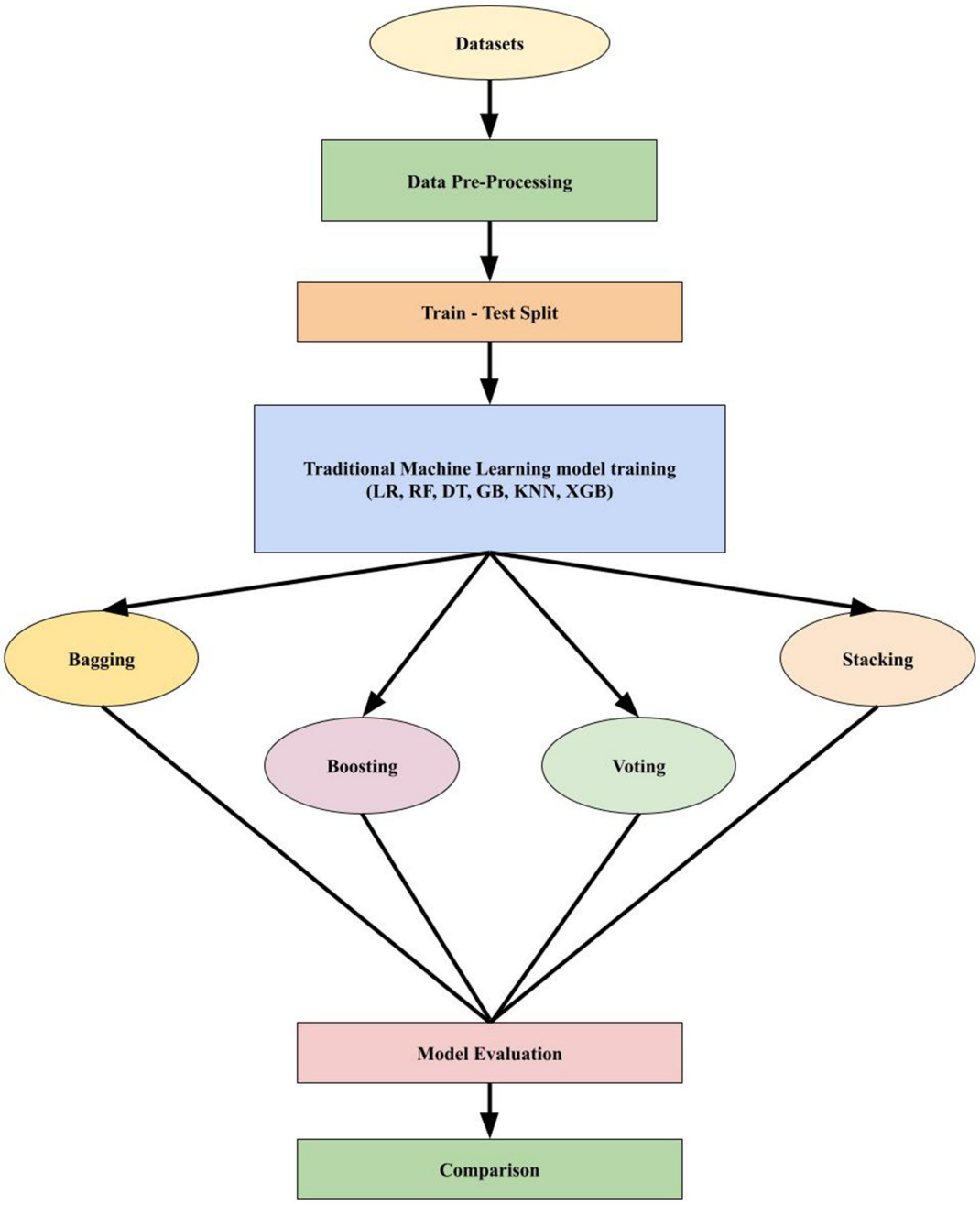
4. Experimental Result

In order to determine the effectiveness of the suggested model aimed at SDP using the assigned datasets, the evaluation of current models was crucial in this stage of the project. After the selected datasets were first implemented, a thorough analysis was conducted to find and correct any missing or incorrect data points and guarantee the dataset's integrity. After that, a wide variety of ML methods were used, and their effectiveness was carefully examined. A thorough evaluation of the suggested algorithms' predictive skills was carried out using confusion matrices, which comprised important metrics including accuracy, precision, recall, and F-1 score. A comparison examination was also made possible by the same inspection that was given to conventional algorithms. The assessment also looked at how various ensemble methods, such as bagging, boosting, stacking, and voting, may be used to combine the advantages of several models to improve prediction accuracy. Six different classical classifiers were used, and the results, which were carefully evaluated, made it easier to determine which methods were best for predicting software defects. This thorough assessment procedure was essential for determining the suggested model's performance and optimizing its forecast accuracy for real-world use. Table 2 shows the comparative analysis of traditional algorithms.

Table 2 summarizes the evaluation results of traditional ML algorithms for SDP across four datasets: PC1, JM1, KC1, and CM1. These datasets are commonly used in software engineering research to benchmark defect prediction models. The algorithms assessed include LR, RF, DT, GB, KNN, and Extreme GB (XGB). Various performance metrics such as accuracy, precision, Sensitivity (recall), and *F*1-score were used to evaluate the algorithms' effectiveness in predicting software defects.

For the PC1 dataset, KNN delivered the best accuracy of 87.08%, with a high *F*1-score of 87.61%, which suggests a strong balance between precision and recall. RF followed closely with an accuracy of 86.36%, with a slightly lower recall but higher

Figure 1
Methodology of software defect prediction



precision compared to KNN, indicating that it identified more true positives but missed a few more true defects. The GB algorithm performed well with 83.97% accuracy and an impressive recall of 91.41%, but its *F*1-score of 84.87% implies that its precision was somewhat lacking compared to other models. LR, by contrast, showed a lower performance at 59.33% accuracy, which underscores its limitations when applied to the PC1 dataset.

In the JM1 dataset, KNN emerged as the top performer, with an accuracy of 76.49% and an *F*1-score of 77.48%, indicating robust performance in identifying software defects while maintaining a balance between precision and recall. In contrast, the RF and DT models performed significantly worse, with accuracies of 55.6% and 53.06%, respectively. The GB and XGB algorithms were among the least effective, with accuracies around 51%, suggesting that ensemble methods did not generalize well on the JM1 dataset. The low precision scores for GB and XGB also reflect a high number of false positives, meaning they incorrectly predicted many non-defective cases as defective.

In the KC1 dataset, KNN was the most effective model again, achieving 81.45% accuracy and an *F*1-score of 82.14%. This demonstrates KNN's capability in predicting software defects with a high balance between precision and recall. XGB followed with 70.85% accuracy, but it had a lower *F*1-score of 73.81%, indicating slightly weaker performance in balancing precision and recall. RF underperformed in this dataset, with an accuracy of 63.87% and lower recall, which signals that it struggled to generalize across different instances of software defects. GB performed particularly poorly, with an accuracy of just 53.41%, a

Figure 2
Count plot before and after using ADASYN (CM1 dataset)

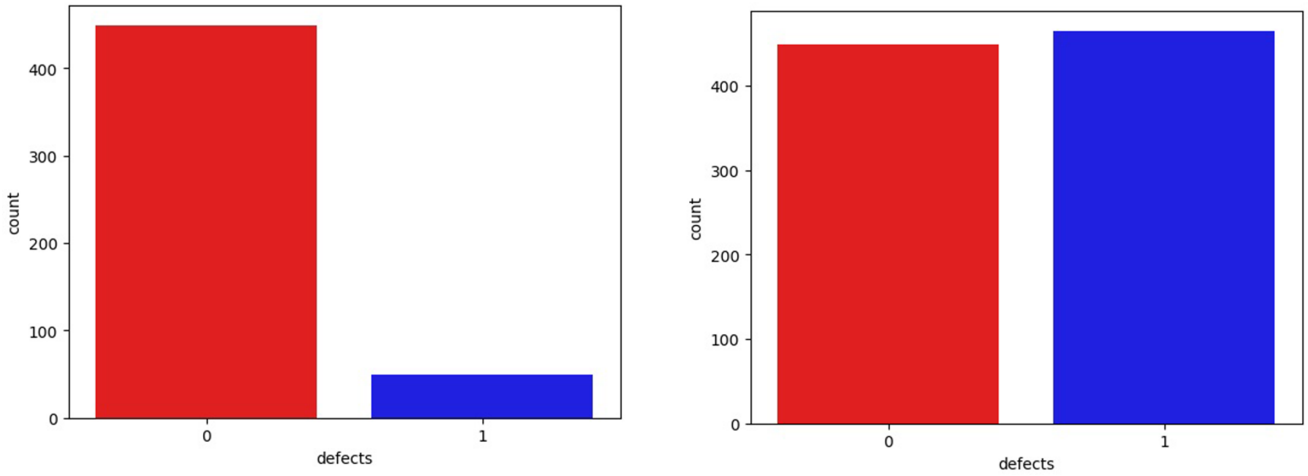


Figure 3
Count plot before and after using ADASYN (JM1 dataset)

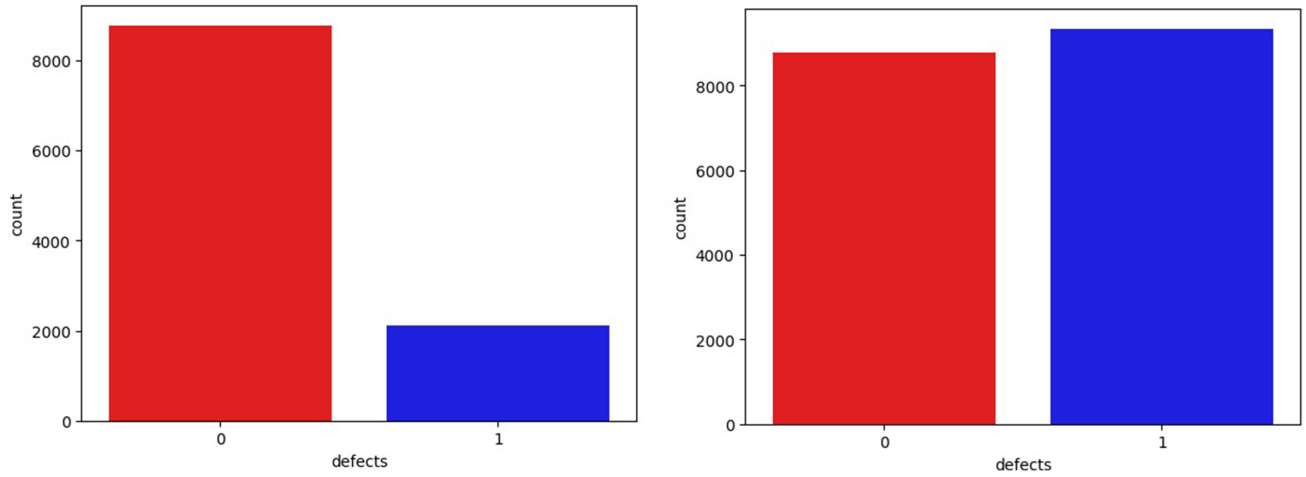


Figure 4
Count plot before and after using ADASYN (KC1 dataset)

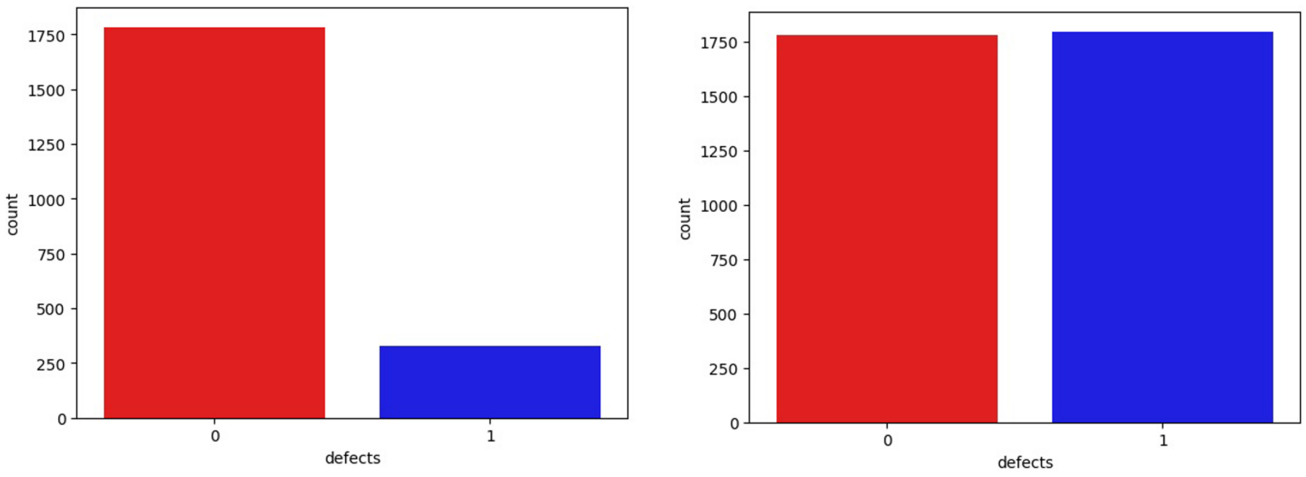


Figure 5
Count plot before and after using ADASYN (PC1 dataset)

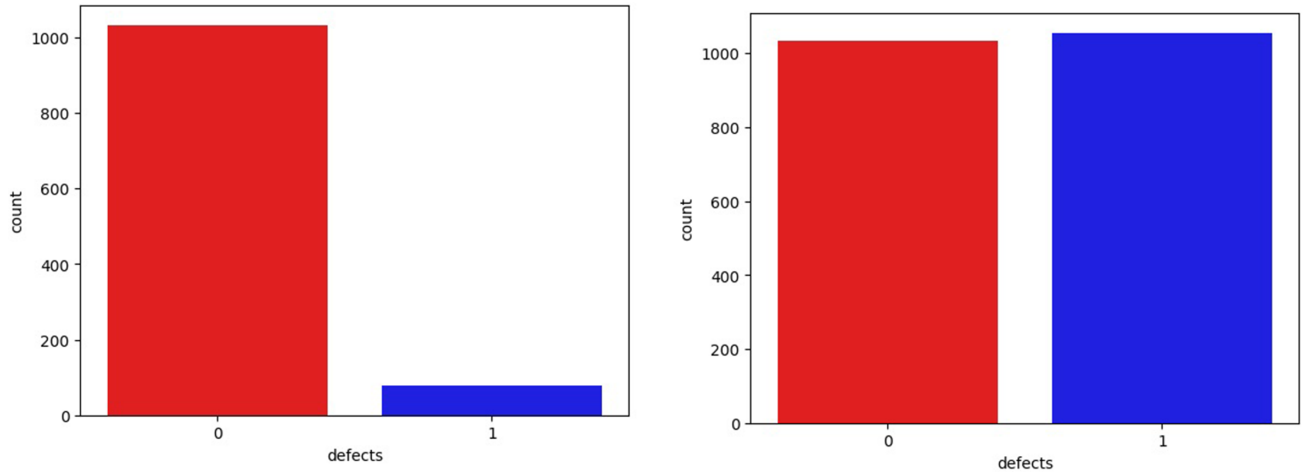


Table 2
Comparative analysis of traditional algorithms

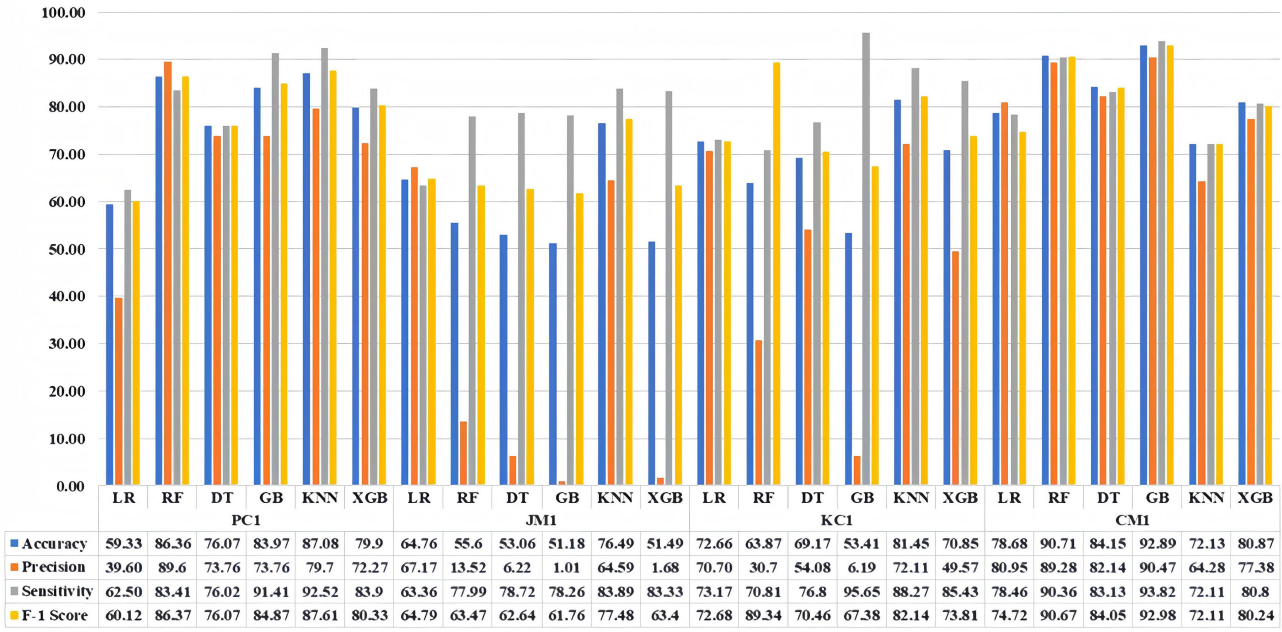
Dataset	Algorithm	Accuracy	Precision	Sensitivity	F-1 score
PC1	LR	59.33	39.60	62.50	60.1
	RF	86.36	89.6	83.41	86.3
	DT	76.07	73.76	76.02	76.0
	GB	83.97	73.76	91.41	84.8
	KNN	87.08	79.7	92.52	87.6
	XGB	79.9	72.27	83.9	80.3
JM1	LR	64.76	67.17	63.36	64.7
	RF	55.6	13.52	77.99	63.4
	DT	53.06	6.22	78.72	62.6
	GB	51.18	1.01	78.26	61.7
	KNN	76.49	64.59	83.89	77.4
	XGB	51.49	1.68	83.33	63.4
KC1	LR	72.66	70.70	73.17	72.6
	RF	63.87	30.7	70.81	89.3
	DT	69.17	54.08	76.8	70.4
	GB	53.41	6.19	95.65	67.3
	KNN	81.45	72.11	88.27	82.1
	XGB	70.85	49.57	85.43	73.8
CM1	LR	78.68	80.95	78.46	74.7
	RF	90.71	89.28	90.36	90.6
	DT	84.15	82.14	83.13	84.0
	GB	92.89	90.47	93.82	92.9
	KNN	72.13	64.28	72.11	72.1
	XGB	80.87	77.38	80.8	80.2

precision of 6.19%, and a recall of 95.65%, indicating that although it found almost all of the true defects, its precision was alarmingly low, resulting in a very high number of false positives.

For the CM1 dataset, GB was the standout performer with an accuracy of 92.89%, and it maintained high scores across all metrics, with a precision of 90.47%, recall of 93.82%, and an *F*1-score of 92.98%. This reflects its ability to accurately predict software defects while minimizing false positives. RF also achieved strong performance, with 90.71% accuracy and a balanced *F*1-score of 90.67%, indicating high effectiveness. In contrast, KNN struggled with this dataset, achieving only 72.13% accuracy, which may be attributed to its sensitivity to the choice of neighbors or the distribution of the data in the CM1 dataset. LR performed reasonably well with 78.68% accuracy, but its *F*1-score of 74.72% highlights some imbalance in precision and recall.

Across all datasets, the KNN algorithm demonstrated consistent performance, particularly excelling in the PC1 and KC1 datasets with accuracies of 87.08% and 81.45%, respectively. However, its performance dipped in the CM1 dataset. RF and GB performed well in certain datasets (CM1), but their effectiveness varied significantly across others (JM1, KC1). XGB showed moderate results, with its best performance in the KC1 dataset (70.85%

Figure 6
Comparative analysis of traditional algorithm



	LR	RF	DT	GB	KNN	XGB	LR	RF	DT	GB	KNN	XGB	LR	RF	DT	GB	KNN	XGB	LR	RF	DT	GB	KNN	XGB
	PC1						JM1						KC1						CM1					
Accuracy	59.33	86.36	76.07	83.97	87.08	79.9	64.76	55.6	53.06	51.18	76.49	51.49	72.66	63.87	69.17	53.41	81.45	70.85	78.68	90.71	84.15	92.89	72.13	80.87
Precision	39.60	89.6	73.76	73.76	79.7	72.27	67.17	13.52	6.22	1.01	64.59	1.68	70.70	30.7	54.08	6.19	72.11	49.57	80.95	89.28	82.14	90.47	64.28	77.38
Sensitivity	62.50	83.41	76.02	91.41	92.52	83.9	63.36	77.99	78.72	78.26	83.89	83.33	73.17	70.81	76.8	95.65	88.27	85.43	78.46	90.36	83.13	93.82	72.11	80.8
F-1 Score	60.12	86.37	76.07	84.87	87.61	80.33	64.79	63.47	62.64	61.76	77.48	63.4	72.68	89.34	70.46	67.38	82.14	73.81	74.72	90.67	84.05	92.98	72.11	80.24

accuracy). LR consistently underperformed across datasets, which aligns with its known limitations in handling complex, non-linear relationships often present in SDP problems.

In terms of metrics, precision and Recall varied widely depending on the dataset and algorithm, with ensemble methods like GB excelling in recall but sometimes suffering in precision. The imbalance between precision and recall in many models highlights the challenge of handling imbalanced datasets, which is a common issue in SDP tasks. Figure 6 shows the comparative analysis of traditional algorithms.

Table 3 shows the comparative analysis of bagging ensemble algorithms. The evaluation results for the ensemble bagging techniques, specifically RF, DT, GB, and KNN, were compared across four datasets (PC1, JM1, KC1, and CM1) for SDP. These results highlight the performance of each algorithm in terms of accuracy, precision, sensitivity (recall), and *F*1-score, offering insights into their effectiveness in handling defect prediction tasks.

In the PC1 dataset, KNN performed the best with an accuracy of 86.84% and a high *F*1-score of 86.72%, showcasing its balanced performance across both precision and recall. This suggests that KNN had a strong ability to classify both defective and non-defective instances effectively. RF and DT followed closely with accuracies of 82.75% and 83.01% respectively, maintaining competitive *F*1-scores. GB showed slightly lower performance with an accuracy of 80.62%, and XGB underperformed with an accuracy of 74.16%. This suggests that while bagging-based models like RF and DT performed well, boosting-based models like XGB struggled more on this dataset.

In the JM1 dataset, KNN once again led the performance with 72.95% accuracy, 74.15% precision, and an *F*1-score of 72.5%. The ability of KNN to generalize well on this dataset is notable, as it balanced precision and recall better than other models. However, RF, DT, GB, and XGB showed a significant drop in performance, with accuracies ranging from 50.91% (GB) to 54.44% (RF). This drop in performance may be attributed to the complexity of the JM1 dataset, where ensemble techniques struggled to handle the distribution of defects effectively. Particularly, GB and XGB had low *F*1-scores, signaling a higher number of false positives and lower model stability.

For the KC1 dataset, KNN delivered the highest accuracy at 80.89% with a solid *F*1-score of 80.73%, reaffirming its strength in defect prediction. XGB performed competitively with 70.99% accuracy and a *F*1-score of 69.85%, while RF followed closely with 70.85% accuracy. Both algorithms demonstrated an ability to detect defects, but the marginally better precision of KNN allowed it to outperform other models. DT and GB lagged behind, with significantly lower accuracies of 57.46% and 51.04% respectively, indicating that these algorithms struggled with the complexity of the KC1 dataset. GB in particular showed a very low *F*1-score, which reflects poor classification balance.

The CM1 dataset saw the best overall performance from ensemble models. DT slightly outperformed the other models with an accuracy of 89.61% and an *F*1-score of 89.55%, followed by GB and RF, both showing strong accuracies at 89.07% and 87.97% respectively. The high *F*1-scores across all three algorithms—DT (89.55%), GB (88.89%), and RF (87.84%)—indicate that these models effectively balanced precision and recall, making them reliable predictors of software defects in this dataset. KNN also showed commendable results with an accuracy of 84.15% but underperformed slightly compared to bagging-based models. XGB again lagged behind with 72.67% accuracy, which demonstrates the challenges that boosting-based models faced with this particular dataset.

Across all datasets, KNN emerged as the top performer in most cases, particularly in the PC1 and KC1 datasets, where it demonstrated the highest accuracy and balanced metrics. Its ability to leverage neighbor-based classification seemed to generalize well to software defect data, especially in cases where

Table 3
Comparative analysis of bagging ensemble algorithms

Dataset	Algorithm	Accuracy	Precision	Sensitivity	F-1 score
PC1	LR	74.4	74.72	74.57	74.3
	RF	82.75	83.74	82.48	82.5
	DT	83.01	83.38	82.82	82.8
	GB	80.62	82.67	80.19	80.1
	KNN	86.84	87.46	86.62	86.7
	XGB	74.16	75.48	73.74	73.5
JM1	LR	63.13	63.18	63.16	63.1
	RF	54.44	65.22	53.72	43.3
	DT	52.12	69.4	51.32	36.9
	GB	50.91	75.43	50.08	33.8
	KNN	72.95	74.15	72.76	72.5
	XGB	51.04	60.47	50.22	34.4
KC1	LR	68.47	68.8	68.53	68.3
	RF	70.85	75.49	70.63	69.3
	DT	57.46	70.57	57.06	48.8
	GB	51.04	67.05	50.56	35.0
	KNN	80.89	81.71	80.81	80.7
	XGB	70.99	74.27	70.8	69.8
CM1	LR	78.14	78.16	78.35	78.1
	RF	87.97	88.06	87.71	87.8
	DT	89.61	89.52	89.59	89.5
	GB	89.07	89.51	88.63	88.8
	KNN	84.15	85.98	83.18	83.5
	XGB	72.67	72.63	72.13	72.2

other models struggled. DT, RF, and GB also performed effectively in several datasets, especially in CM1, where they showed strong overall performance. GB and XGB, although powerful boosting-based techniques, did not consistently outperform bagging methods like RF or DT across these datasets, particularly showing weakness in JM1 and KC1 datasets.

These results underscore that while ensemble techniques are powerful, their performance in SDP is highly dataset-dependent. For instance, models like DT and RF generally perform well in datasets with more clear defect patterns (e.g., CM1), but struggle with noisier or more complex datasets (e.g., JM1). On the other hand, KNN's simplicity and ability to adjust to local data distributions make it more consistent across varying datasets. Furthermore, models like GB and XGB, which are known for handling non-linear relationships, might require further tuning or additional data preprocessing to perform better in defect prediction tasks. Overall, bagging-based techniques tend to offer more stable and reliable results for SDP across these datasets. Figure 7 shows the comparative analysis of bagging ensemble algorithms.

Table 4 shows the comparative analysis of boosting ensemble algorithms. The evaluation results for boosting algorithms in SDP reveal significant differences in performance across four datasets (PC1, JM1, KC1, and CM1) when compared to LR, RF, DT, GB, and XGBoost (XGB). Each algorithm's accuracy, precision, sensitivity, and *F*1-score are important metrics to analyze their performance in identifying software defects.

In the PC1 dataset, RF outperformed the other boosting algorithms with an accuracy of 86.36% and a high *F*1-score of 86.33%, demonstrating strong predictive power with consistent precision and recall values. GB performed similarly well with 83.25% accuracy and an *F*1-score of 83.08%, suggesting that boosting techniques can handle defect prediction effectively. However, XGB performed very poorly with an accuracy of 51.67% and an *F*1-score of 34.06%, indicating that it struggled with this dataset, potentially due to overfitting or the nature of the dataset not being well-suited for XGBoost's complexity. LR also performed decently with an accuracy of 76.31%, but it could not match the ensemble methods.

In the JM1 dataset, RF once again led the results with an accuracy of 72.01% and an *F*1-score of 71.92%, showcasing its strong classification abilities in this dataset. GB, on the other hand, underperformed significantly with an accuracy of 52.09% and a low *F*1-score of 36.82%, indicating that it struggled with the JM1 dataset, which may be more complex or noisy. XGB was the worst-performing algorithm in this dataset, with 50.82% accuracy and an *F*1-score of 33.69%, suggesting that it faced difficulties similar to those in the PC1 dataset. LR performed slightly better than boosting methods like XGBoost and GB, achieving 64.29% accuracy, but overall, the results indicate that boosting techniques are not as effective in this dataset.

In the KC1 dataset, none of the boosting algorithms showed impressive performance. RF managed to achieve 65.13% accuracy and an *F*1-score of 61.68%, which was the highest among the boosting models. GB followed with 53.41% accuracy, but its *F*1-score of 40.4% reflects its struggle with maintaining a balance between precision and recall. XGB again performed poorly with an accuracy of 50.48% and an *F*1-score of 33.54%, indicating that it was not effective in this dataset. LR performed better than the boosting models with 71.82% accuracy, suggesting that linear methods may be more suited for KC1's data patterns than complex boosting techniques.

The CM1 dataset yielded the best performance overall, especially for GB, which achieved an impressive accuracy of 94.5% and an *F*1-score of 92.26%, demonstrating that GB is highly effective when the dataset structure favors non-linear relationships and complex feature interactions. RF also performed excellently with 90.71% accuracy and an *F*1-score of 90.51%, maintaining a balanced and strong classification

Figure 7
Comparative analysis of bagging classifiers

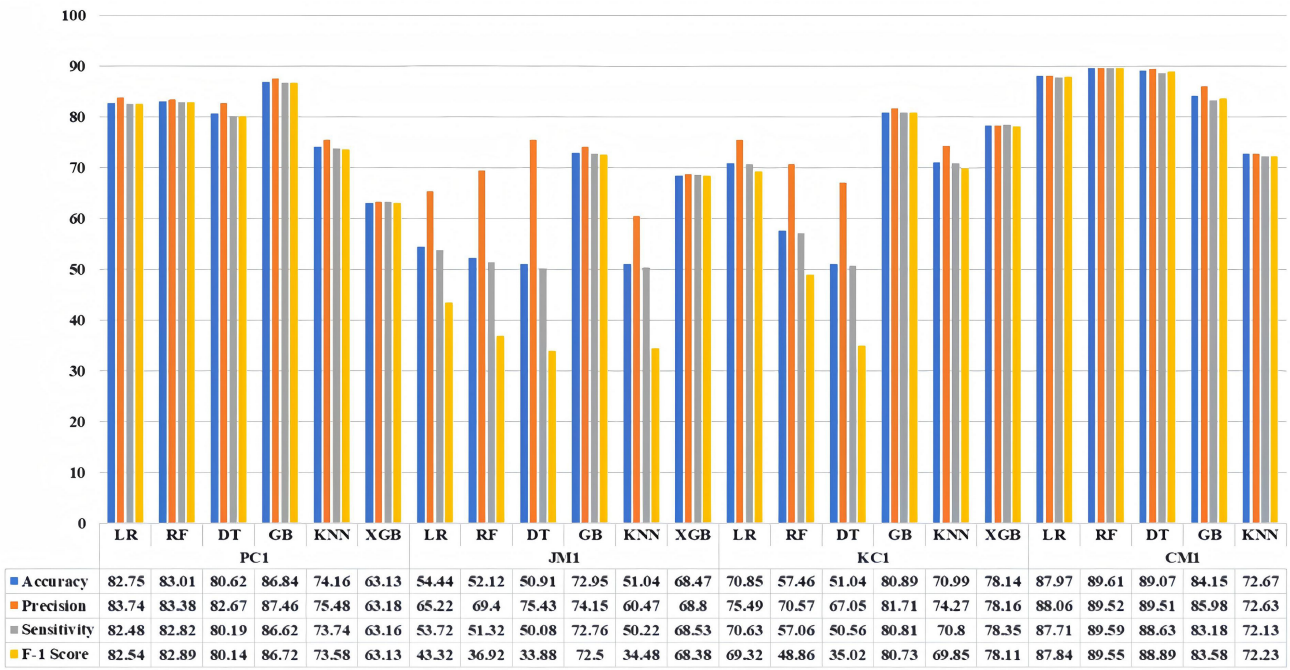


Table 4
Comparative analysis of boosting ensemble algorithms

Dataset	Algorithm	Accuracy	Precision	Sensitivity	F-1 score
PC1	LR	76.31	77.11	76.58	76.2
	RF	86.36	86.38	86.3	86.3
	DT	82.77	82.95	82.9	82.7
	GB	83.25	83.92	83.01	83.0
	XGB	51.67	25.83	50	34.0
JM1	LR	64.29	64.28	64.26	64.2
	RF	72.01	72.16	71.94	71.9
	DT	61.69	64.01	61.34	59.6
	GB	52.09	69.91	51.29	36.8
	XGB	50.82	25.41	50	33.6
KC1	LR	71.82	71.83	71.83	71.8
	RF	65.13	72.76	64.84	61.6
	DT	59.55	69.65	59.19	53.1
	GB	53.41	70.45	52.96	40.4
	XGB	50.48	25.24	50	33.4
CM1	LR	74.31	74.65	74.72	74.3
	RF	90.71	91.58	90.15	90.5
	DT	86.33	86.27	86.2	86.2
	GB	94.5	92.5	92.11	92.2
	XGB	54.09	27.04	50	35.1

performance. DT followed with a commendable accuracy of 86.33%. However, XGB lagged again with 54.09% accuracy and a poor *F*1-score of 35.1%, indicating that it was outperformed by all other boosting methods and even linear models like LR (which achieved 74.31% accuracy).

The results across all datasets suggest that ensemble boosting techniques like GB and RF are generally effective for SDP, but their success varies significantly depending on the dataset. RF proved to be the most stable performer across different datasets, consistently yielding high accuracy, precision, and *F*1-scores, especially in the PC1, JM1, and CM1 datasets. GB showed strong potential, particularly in the CM1 dataset, where it outperformed other methods, but struggled in more complex or less structured datasets like JM1.

XGB, surprisingly, performed poorly across all datasets, particularly with its accuracy hovering around 50% and *F*1-scores

below 40%. This is unusual given XGBoost's reputation for strong performance in many classification tasks, and it may indicate that the algorithm's default hyperparameters were not optimal for these specific datasets or that XGBoost was more prone to overfitting or underfitting in these cases. Figure 8 shows the comparative analysis of boosting ensemble algorithms.

Table 5 shows the comparative analysis of stacking and voting ensemble algorithms. The results of stacking and voting ensemble algorithms in SDP across four datasets—PC1, JM1, KC1, and CM1—indicate varying performance levels.

In the PC1 dataset, stacking performed slightly better than voting, achieving an accuracy of 86.36% compared to voting's 85.88%. Stacking also had superior *F*1-scores (86.21%) and precision (87.17%), demonstrating its ability to combine models effectively and yield higher predictive accuracy.

In the JM1 dataset, stacking again outperformed voting, with an accuracy of 70.47% compared to voting's 53.17%. The precision and *F*1-scores also highlight stacking's superior performance: 76.63% precision and 68.39% F1 compared to voting's 39.24% F1. The notable gap between the two methods suggests that stacking is more suited to this dataset's complexity.

For the KC1 dataset, voting was more effective than stacking, showing an accuracy of 75.59% versus 61.78% for stacking. Voting also had better precision (79.71%) and *F*1-scores (74.62%), indicating that combining predictions through voting provided better generalization for this dataset compared to the more complex stacking method.

In the CM1 dataset, voting slightly outperformed stacking in terms of accuracy, with 92.34% compared to stacking's 91.8%. However, stacking still delivered strong results with *F*1-scores and precision above 91%, showing that both ensemble methods performed well in this dataset. The minimal difference between the two methods suggests that both were well-suited to the CM1 dataset's characteristics. Figure 9 shows the comparative analysis of stacking and voting ensemble algorithms.

While the Gradient Boosting with Boosting (GBB) model showed exceptional accuracy in the studied datasets, its generalizability to other datasets or real-world projects depends on

Figure 8
Comparative analysis of boosting classifiers

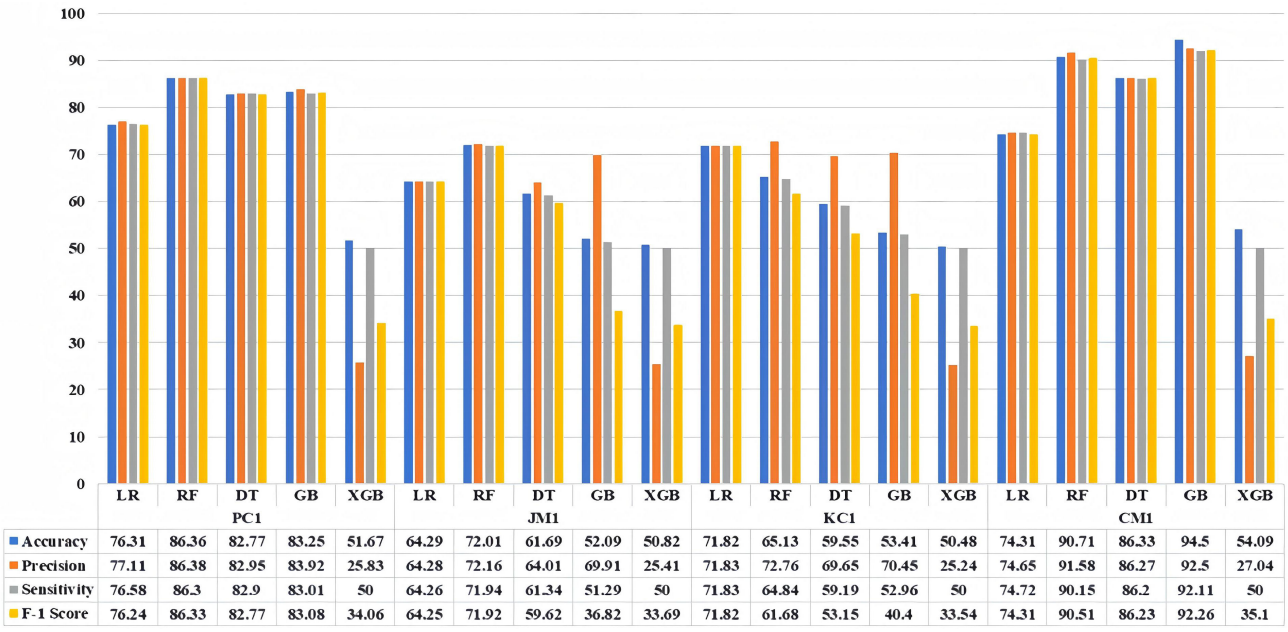
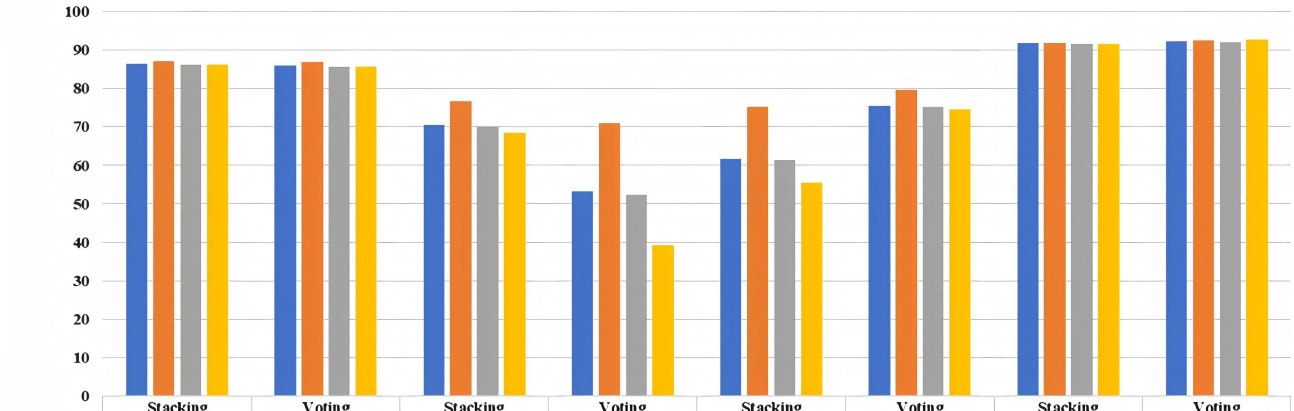


Table 5
Comparative analysis of stacking and voting ensemble algorithms

Dataset	Algorithm	Accuracy	Precision	Sensitivity	F-1 score
PC1	Stacking	86.36	87.17	86.11	86.2
	Voting	85.88	86.79	85.62	85.7
JM1	Stacking	70.47	76.63	70.06	68.3
	Voting	53.17	70.89	52.39	39.2
KC1	Stacking	61.78	75.29	61.42	55.5
	Voting	75.59	79.71	75.4	74.6
CM1	Stacking	91.8	91.88	91.61	91.7
	Voting	92.34	92.5	92.11	92.6

Figure 9
Comparative analysis of staking and voting classifiers



	Stacking	Voting	Stacking	Voting	Stacking	Voting	Stacking	Voting
	PC1		JM1		KC1		CM1	
Accuracy	86.36	85.88	70.47	53.17	61.78	75.59	91.8	92.34
Precision	87.17	86.79	76.63	70.89	75.29	79.71	91.88	92.5
Sensitivity	86.11	85.62	70.06	52.39	61.42	75.4	91.61	92.11
F-1 Score	86.21	85.71	68.39	39.24	55.52	74.62	91.72	92.66

various factors. These include the similarity of data distributions, the relevance of selected features, and the prevalence of defects. Nonetheless, the robust performance of GBB across multiple datasets in this study suggests it has strong potential for broader applicability, provided appropriate tuning and preprocessing are applied.

5. Conclusion

The comparison of various ML models, including traditional, bagging, boosting, stacking, and voting ensemble techniques, for SDP shows that no single method universally outperforms the others across all datasets. Traditional methods like RF and KNN performed consistently well, especially in datasets like PC1 and CM1. However, ensemble techniques like stacking and voting offered improved performance in some cases, particularly for complex datasets like JM1 and KC1, where stacking outperformed voting in accuracy and precision. Ensemble techniques, due to their ability to integrate multiple models, provided a balanced approach to handling SDP, showing resilience in datasets with different characteristics. This reinforces the importance of selecting the right algorithm based on dataset attributes, as more sophisticated techniques such as stacking could be advantageous when the dataset presents more complex patterns.

One of the primary challenges in using ensemble models for SDP is the risk of overfitting, especially when dealing with small or imbalanced datasets. Boosting methods, while powerful, may fit noise in the data if not properly regularized. Additionally, ensemble models can be computationally intensive, requiring more resources and time for training and inference, which may limit their scalability in large, real-time systems without proper optimization.

In future work, further improvements in SDP could be achieved by exploring hybrid ensemble techniques that combine the strengths of stacking, voting, and other ensemble methods. Incorporating advanced feature engineering, such as deep learning-based feature extraction, could also enhance model performance. Additionally, investigating deep learning methods like CNNs and recurrent neural networks (RNNs) may yield new insights, particularly for larger datasets. Moreover, cross-dataset validation and automated hyperparameter tuning through optimization algorithms can help ensure model robustness and generalizability across various software projects.

Recommendations

To further improve SDP, future research should explore hybrid ensemble techniques that integrate stacking, voting, and other ensemble methods to enhance predictive performance. Advanced feature engineering, including deep learning-based feature extraction, could provide better representations of software defect patterns. Investigating deep learning models like CNNs and RNNs may offer new insights, especially for larger datasets. Additionally, cross-dataset validation should be conducted to assess model generalizability across diverse software projects. Automated hyperparameter tuning using optimization algorithms can further refine model performance, ensuring robustness and adaptability to varying dataset characteristics.

Acknowledgement

The authors sincerely acknowledge the support and resources provided by their respective institutions, which facilitated the successful completion of this research. They also extend their gratitude to the developers and maintainers of the publicly available datasets used in this study, as well as the open-source ML communities for providing valuable tools and frameworks. Additionally, the authors appreciate the constructive feedback from peers and reviewers, which contributed to refining the study's methodology and findings.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

The data that support the findings of this study are openly available in Software Defect Prediction Datasets at https://github.com/SinghJasmeet585/Software-Defect-Prediction/tree/master/Data/data.

Author Contribution Statement

S. M. Hasan Kabir: Methodology, Software, Formal analysis, Data curation, Writing – original draft, Writing – review & editing. **Md. Tanim Rahman:** Software, Validation, Data curation, writing – original draft, Writing – review & editing. **Aunik Hasan Mridul:** Conceptualization, Software, Investigation, Resources, Data curation, Writing – review & editing, Visualization, Supervision, Project administration.

References

[1] Chengxiao, Y., Owais, J., Ahmed, A., Khan, M. O., Xiaoyang, Z., & Hanif Tunio, M. (2023). Software defect prediction using machine learning-a systematic literature review. In *2023 20th International Computer Conference on Wavelet Active Media Technology and Information Processing,* 1–9. https://doi.org/10.1109/ICCWAMTIP60502.2023.10387043

[2] Sekaran, K., & Puspha Annabel, L. S. (2023). A deep learning based model for defect prediction in intra-project software. In *2023 7th International Conference on Trends in Electronics and Informatics*, 1148–1155. https://doi.org/10.1109/ICOEI56765.2023.10126014

[3] Gautam, S., Khunteta, A., & Ghosh, D. (2023). A review on software defect prediction using machine learning. In *Proceedings of the 4th International Conference on Information Management & Machine Intelligence*, 81. https://doi.org/10.1145/3590837.3590918

[4] Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2011). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276–1304. https://doi.org/10.1109/TSE.2011.103

[5] Mohammadi, M., di Nucci, D., & Tamburri, D. A. (2023). Bayesian meta-analysis of software defect prediction with machine learning. *IEEE Transactions on Industrial Cyber-Physical Systems*, *1*, 147–156. https://doi.org/10.1109/TICPS.2023.3306723

[6] Chen, H., Jing, X. Y., Li, Z., Wu, D., Peng, Y., & Huang, Z. (2021). An empirical study on heterogeneous defect prediction approaches. *IEEE Transactions on Software Engineering*, *47*(12), 2803–2822. https://doi.org/10.1109/TSE.2020.2968520

[7] Guo, Z., Liu, S., Liu, X., Lai, W., Ma, M., Zhang, X., . . . , & Zhou, Y. (2023). Code-line-level bugginess identification: How far have we come, and how far have we yet to go? *ACM Transactions on Software Engineering and Methodology*, *32*(4), 102. https://doi.org/10.1145/3582572

[8] Jiarpakdee, J., Tantithamthavorn, C., & Hassan, A. E. (2021). The impact of correlated metrics on the interpretation of defect models. *IEEE Transactions on Software Engineering*, *47*(2), 320–331. https://doi.org/10.1109/TSE.2019.2891758

[9] Giray, G., Bennin, K. E., Köksal, Ö., Babur, Ö., & Tekinerdogan, B. (2023). On the use of deep learning in software defect prediction. *Journal of Systems and Software*, *195*, 111537. https://doi.org/10.1016/j.jss.2022.111537

[10] Jiarpakdee, J., Tantithamthavorn, C. K., Dam, H. K., & Grundy, J. (2022). An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering*, *48*(1), 166–185. https://doi.org/10.1109/TSE.2020.2982385

[11] Li, Z., Jing, X. Y., Zhu, X., Zhang, H., Xu, B., & Ying, S. (2019). Heterogeneous defect prediction with two-stage ensemble learning. *Automated Software Engineering*, *26*(3), 599–651. https://doi.org/10.1007/s10515-019-00259-1

[12] Li, Z., Jing, X. Y., Zhu, X., Zhang, H., Xu, B., & Ying, S. (2019). On the multiple sources and privacy preservation issues for heterogeneous defect prediction. *IEEE Transactions on Software Engineering*, *45*(4), 391–411. https://doi.org/10.1109/tse.2017.2780222

[13] Karnavel, K., & Dillibabu, R. (2014). Development and application of new quality model for software projects. *The Scientific World Journal*, *2014*(1), 491246. https://doi.org/10.1155/2014/491246

[14] Alsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Al-Radaideh, A., Aljarah, I., & Alshamaileh, Y. (2020). Softwaredefect prediction using heterogeneous ensemble classification based on segmented patterns. *Applied Sciences*, *10*(5), 1745. https://doi.org/10.3390/app10051745

[15] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2019). The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, *45*(7), 683–711. https://doi.org/10.1109/TSE.2018.2794977

[16] Saheed, Y. K., Longe, O., Baba, U. A., Rakshit, S., & Vajjhala, N. R. (2021). An ensemble learning approach for software defect prediction in developing quality software product. In *Advances in Computing and Data Sciences: 5th International Conference,* 317–326. https://doi.org/10.1007/978-3-030-81462-5_29

[17] Ge, J., Liu, J., & Liu, W. (2018). Comparative study on defect prediction algorithms of supervised learning software based on imbalanced classification data sets. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing,* 399–406. https://doi.org/10.1109/SNPD.2018.8441143

[18] Ronchieri, E., Canaparo, M., Belgiovine, M., & Salomoni, D. (2019). Software defect prediction on unlabelled dataset with machine learning techniques. In *2019 IEEE Nuclear Science Symposium and Medical Imaging Conference,* 1–2. https://doi.org/10.1109/NSS/MIC42101.2019.9059737

[19] Assim, M., Obeidat, Q., & Hammad, M. (2020). Software defects prediction using machine learning algorithms. In *2020 International Conference on Data Analytics for Business and Industry: Way towards a Sustainable Economy,* 1–6. https://doi.org/10.1109/ICDABI51230.2020.9325677

[20] Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning based software defect prediction. *Neurocomputing*, *385*, 100–110. https://doi.org/10.1016/j.neucom.2019.11.067

[21] Chen, J., Hu, K., Yu, Y., Chen, Z., Xuan, Q., Liu, Y., & Filkov, V. (2020). Software visualization and deep transfer learning for effective software defect prediction. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering,* 578–589. https://doi.org/10.1145/3377811.3380389

[22] Hasanpour, A., Farzi, P., Tehrani, A., & Akbari, R. (2020). Software defect prediction based on deep learning models:

Performance study. *arXiv Preprint: 2004.02589*. https://doi.org/10.48550/arXiv.2004.02589

[23] Jin, C. (2021). Software defect prediction model based on distance metric learning. *Soft Computing*, *25*(1), 447–461. https://doi.org/10.1007/s00500-020-05159-1

[24] Khan, B., Naseem, R., Shah, M. A., Wakil, K., Khan, A., Uddin, M. I., & Mahmoud, M. (2021). Software defect prediction for healthcare big data: An empirical evaluation of machine learning techniques. *Journal of Healthcare Engineering*, *2021*(1), 8899263. https://doi.org/10.1155/2021/8899263

[25] Zhang, W. H., He, R. Y., Wu, L. J., Jian, Y., & Han, X. Y. (2021). Comparison of software defect prediction models based on machine learning. *IOP Conference Series: Materials Science and Engineering, 1043*(3), 032074. https://doi.org/10.1088/1757-899x/1043/3/032074

[26] Shi, K., Lu, Y., Liu, G., Wei, Z., & Chang, J. (2021). MPT-embedding: An unsupervised representation learning of code for software defect prediction. *Journal of Software: Evolution and Process*, *33*(4), e2330. https://doi.org/10.1002/smr.2330

[27] Rahim, A., Hayat, Z., Abbas, M., Rahim, A., & Rahim, M. A. (2021). Software defect prediction with Naïve Bayes classifier. In *2021 International Bhurban Conference on Applied Sciences and Technologies,* 293–297. https://doi.org/10.1109/IBCAST51254.2021.9393250

[28] Lin, J., & Lu, L. (2021). Semantic feature learning via dual sequences for defect prediction. *IEEE Access*, *9*, 13112–13124. https://doi.org/10.1109/ACCESS.2021.3051957

[29] Chen, L., Fang, B., Shang, Z., & Tang, Y. (2018). Tackling class overlap and imbalance problems in software defect prediction. *Software Quality Journal*, *26*(1), 97–125. https://doi.org/10.1007/s11219-016-9342-6

[30] Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. *International Journal of Advanced Computer Science and Applications, 9*(2), 78–83. https://doi.org/10.14569/IJACSA.2018.090212

[31] Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., . . . , & Soomro, T. R. (2021). Software defect prediction using ensemble learning: A systematic literature review. *IEEe Access*, *9*, 98754–98771. https://doi.org/10.1109/ACCESS.2021.3095559.

[32] Tajmen, S., Karim, A., Hasan Mridul, A., Azam, S., Ghosh, P., Dhaly, A. A., & Hossain, M. N. (2022). A machine learning based proposition for automated and methodical prediction of liver disease. In *Proceedings of the 10th International Conference on Computer and Communications Management,* 46–53. https://doi.org/10.1145/3556223.3556230.

[33] Mridul, A. H., Islam, M. J., Asif, A., Rahman, M., & Alam, M. J. (2023). A machine learning-based traditional and ensemble technique for predicting breast cancer. In *22nd International Conference on Hybrid Intelligent Systems,* 237–248. https://doi.org/10.1007/978-3-031-27409-1_21

[34] Pasha, M., & Fatima, M. (2017). Comparative analysis of meta learning algorithms for liver disease detection. *Journal of Software*, *12*(12), 923–933. https://doi.org/10.17706/jsw.12.12.923-933

[35] Chandrasekaran, M. (2021). *Logistic regression for machine learning*. Retrieved from: https://www.capitalone.com/tech/machine-learning/what-is-logistic-regression/

[36] Ghosh, P., Karim, A., Atik, S. T., Afrin, S., & Saifuzzaman, M. (2021). Expert cancer model using supervised algorithms with a LASSO selection approach. *International Journal of Electrical and Computer Engineering*, *11*(3), 2631–2639. https://doi.org/10.11591/ijece.v11i3.pp2631-2639

[37] Lemmens, A., & Croux, C. (2006). Bagging and boosting classification trees to predict churn. *Journal of Marketing Research*, *43*(2), 276–286. https://doi.org/10.1509/jmkr.43.2.276

[38] Bentéjac, C., Csörgő, A., & Martínez-Muñoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, *54*(3), 1937–1967. https://doi.org/10.1007/s10462-020-09896-5

[39] Wang, Y., Jha, S., & Chaudhuri, K. (2018). Analyzing the robustness of nearest neighbors to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning, 80*, 5133–5142. https://doi.org/10.48550/arXiv.1706.03922

[40] Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., & Vapnik, V. (1994). Boosting and other ensemble methods. *Neural Computation*, *6*(6), 1289–1301. https://doi.org/10.1162/neco.1994.6.6.1289

[41] Sharma, A., & Suryawanshi, A. (2016). A novel method for detecting spam email using KNN classification with spearman correlation as distance measure. *International Journal of Computer Applications*, *136*(6), 28–35. https://doi.org/10.5120/ijca2016908471

[42] Bakır, H. (2025). VoteDroid: A new ensemble voting classifier for malware detection based on fine-tuned deep learning models. *Multimedia Tools and Applications*, *84*(12), 10923–10944. https://doi.org/10.1007/s11042-024-19390-7

[43] Islam, R., Beeravolu, A. R., Islam, M. A. H., Karim, A., Azam, S., & Mukti, S. A. (2021). A performance based study on deep learning algorithms in the efficient prediction of heart disease. In *2021 2nd International Informatics and Software Engineering Conference,* 1–6. https://doi.org/10.1109/IISEC54230.2021.9672415

[44] Mridul, A. H. (2025). The precision skin disease classification in IoMT systems harnessing SVM and SmoothGrad for better interpretability: Skin disease classification in IoMT systems harnessing SVM and SmoothGrad. *International Journal of Digital Innovation, Insight, and Information*, *1*(01), 60–68.

[45] SinghJasmeet585. (2019). *Software defect prediction* [Data set]. GitHub. https://github.com/SinghJasmeet585/Software-Defect-Prediction/tree/master/Data/data

[46] Mridul, A. H., Ahsan, N., Alam, S. S., Afrose, S., & Sultana, Z. (2024). Polycystic ovary syndrome (PCOS) disease prediction using traditional machine learning and deep learning algorithms. *International Journal of Computer Information Systems and Industrial Management Applications*, *16*(3), 322–346.

How to Cite: Kabir, S. M.H., Rahman, M. T., & Mridul, A. H. (2025). Software Defect Prediction Using Traditional Machine Learning and Ensemble Learning Algorithms. *Smart Wearable Technology*, *1*, A9. https://doi.org/10.47852/bonviewSWT52025645