Scientific Research Publishing

Journal of Software Engineering and Applications, 2024, 17, 155-171
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study

Hemant Kumar, Vipin Saxena

Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India
Email: hemant20192@gmail.com, profvipinsaxena@gmail.com

How to cite this paper: Kumar, H. and Saxena, V. (2024) Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study. *Journal of Software Engineering and Applications*, **17**, 155-171.
https://doi.org/10.4236/jsea.2024.174009

Received: February 22, 2024
Accepted: April 12, 2024
Published: April 15, 2024

Abstract

When a customer uses the software, then it is possible to occur defects that can be removed in the updated versions of the software. Hence, in the present work, a robust examination of cross-project software defect prediction is elaborated through an innovative hybrid machine learning framework. The proposed technique combines an advanced deep neural network architecture with ensemble models such as Support Vector Machine (SVM), Random Forest (RF), and XGBoost. The study evaluates the performance by considering multiple software projects like CM1, JM1, KC1, and PC1 using datasets from the PROMISE Software Engineering Repository. The three hybrid models that are compared are Hybrid Model-1 (SVM, RandomForest, XGBoost, Neural Network), Hybrid Model-2 (GradientBoosting, DecisionTree, LogisticRegression, Neural Network), and Hybrid Model-3 (KNeighbors, GaussianNB, Support Vector Classification (SVC), Neural Network), and the Hybrid Model 3 surpasses the others in terms of recall, F1-score, accuracy, ROC AUC, and precision. The presented work offers valuable insights into the effectiveness of hybrid techniques for cross-project defect prediction, providing a comparative perspective on early defect identification and mitigation strategies.

Keywords

Defect Prediction, Hybrid Techniques, Ensemble Models, Machine Learning, Neural Network

1. Introduction

In the ever-evolving realm of software development, the pursuit of pre-emptive defect detection and efficient mitigation strategies remains a critical endeavor.

Leveraging the power of advanced machine learning, this paper delves into a comprehensive exploration of cross-project software defect prediction The proposed strategy provides a unique hybrid machine learning framework by combining deep neural network designs with ensemble models like Random Forest, XGBoost, and SVM [1] [2] [3] [4] [5]. SVM, a foundational element of Hybrid Model-1, excels in capturing complex decision boundaries and navigating high-dimensional data spaces. This model's predictive capacity is further fortified through the inclusion of Random Forest, a versatile ensemble learning method celebrated for its robustness and ability to handle datasets rife with noise [6]. Concurrently, XGBoost, a key constituent of Hybrid Model-1, strategically assembles weak learners, culminating in an augmented predictive capability through boosting. The introduction of Hybrid Model-2 introduces a distinct ensemble featuring GradientBoosting, DecisionTree, and LogisticRegression [7] [8]. GradientBoosting, sharing traits with XGBoost, iteratively constructs weak learners, contributing to the model's adaptability and resilience. DecisionTree imparts simplicity and interpretability to the model, while LogisticRegression proves particularly advantageous in the context of binary classification tasks. On the other hand, Hybrid Model-3 adopts a unique amalgamation strategy, integrating KNeighbors, GaussianNB, and SVC [9] [10] [11]. KNeighbors, guided by feature similarity, collaborates with GaussianNB, a probabilistic model rooted in Bayes' theorem. Simultaneously, SVC is incorporated, seeking optimal hyperplanes for classification. To comprehensively evaluate the performance of said hybrid models, datasets are utilized from renowned software projects, including CM1, JM1, KC1, and PC1, sourced from the PROMISE Software Engineering Repository [12]. Performance metrics encompassing recall, F1-score, accuracy, ROC AUC, and precision offer a nuanced understanding of the models' multifaceted effectiveness [13]. Notably, Hybrid Model-3 emerges as the preeminent performer across diverse metrics, emphasizing the significance of its unique algorithmic composition. The complexity of developing said hybrid models is explained in detail in this study, along with the methodical approach, the complexities of the experimental setup, and a thorough analysis of the results. Software engineers and practitioners can use the research's insights to help in the search for effective defect identification and mitigation strategies for a variety of software projects. The insights also advance the understanding of hybrid machine-learning techniques in software defect prediction.

2. Related Work

In the software, it is important to uncover the software bugs which are errors and flaws and removed in the update versions of the software. Some of the latest research on the said article is described here which automatically covers the previous research available in the literature. In the year 2020, Thota *et al.* [14] investigated software defect prediction, highlighting its crucial role in maintaining high-quality software during technological advancements. The authors presented an efficient approach that can be utilized in the soft computing-based machine

learning techniques to optimize feature prediction. The strategy aimed to alleviate challenges in industries with high software development costs, especially in safety-critical systems, providing valuable insights for enhancing the testing strategies. Further, Ning Li *et al.* [15] conducted an examination of 49 studies on unsupervised learning techniques for software defect prediction, encompassing 2456 experimental results. The meta-analysis revealed that unsupervised models, especially Fuzzy C-Means (FCM) and Fuzzy SOMs (FSOM's), demonstrated comparable performance to supervised models in both within-project and cross-project prediction. However, the review identified concerns, such as demonstrably erroneous results, undemanding benchmarks, and incomplete reporting, emphasizing the need for comprehensive research reporting practices in this domain. In the year 2021, Matloob *et al.* [16] systematically reviewed the literature on Software Defect Prediction (SDP) utilizing ensemble learning, an approach that integrates multiple classification techniques to enhance prediction performance. The study analyzed the research papers published from 2012 onward across renowned online libraries such as ACM, IEEE, Springer Link, and Science Direct. Addressing five research questions, the review highlighted progress in ensemble learning for SDP. Out of the 46 relevant papers considered, the review revealed that commonly employed ensemble methods included random forest, boosting, and bagging, while less common methods encompassed stacking, voting, and Extra Trees. Numerous promising frameworks were proposed, such as EMKCA, SMOTE-Ensemble, MKEL, SDAEsTSE, TLEL, and LRCR. Performance measurement metrics included AUC, accuracy, F-measure, Recall, Precision, and MCC, with WEKA being widely adopted as a machine learning platform. Empirical analyses underscored the importance of features selection and data sampling as pre-processing steps to enhance the performance of ensemble classifiers. Akimova *et al.* [17] conducted a survey on software defect prediction using deep learning techniques, addressing the key challenge of identifying defective source code for enhanced software quality and reliability. The study delved into recent developments in machine learning, particularly in deep learning, and explored methods for automatically learning semantic and structural features from code. The survey analyzed recent works in the field, highlighted open problems, and discussed emerging trends in software defect prediction through deep learning. Gong *et al.* [18] conducted a study to reassess the impact of Software Dependency Network Analysis (SDNA) metrics, extracted using Social Network Analysis (SNA), on Software Defect Prediction (SDP) models. The research aimed to clarify the relative effectiveness of SNA metrics compared to traditional code metrics in different SDP contexts (Within-project, Cross-version, and Cross-project) and scenarios (Defect-count, Defect-classification, and Effort-aware). The study was based on a case analysis of nine open-source software projects spanning 30 versions, found that incorporating SNA metrics, either alone or in combination with code metrics, improved the performance of SDP models in five out of nine studied scenarios. The findings suggested that future research should consider both SNA metrics and code metrics

in SDP models, considering the different behaviors of Ego metrics and Global metrics, two types of SNA metrics, when training the models.

In the year 2022, Khan *et al.* [19] conducted a systematic literature review on software defect prediction using Artificial Neural Networks (ANN's). The study, which covered publications from 2015 to 2018, aimed to analyze recent trends and critical aspects of using ANN's in defect prediction. The research highlighted the increasing demand for high-quality and cost-effective software systems and emphasized the significance of defect prediction in the software development life cycle. The review was based on publications from IEEE, Elsevier, and Springer, identified eight of the most relevant research works for in-depth analysis, providing valuable insights for researchers. Goyal [20] conducted research on SDP focusing on the effective utilization of SVM's. The study addressed the challenges of imbalanced datasets, specifically the uneven distribution of faulty and non-faulty modules, which can impact the accuracy of SVMs. The author introduced a novel filtering technique (FILTER) to enhance defect prediction using SVM's. The research involved designing SVM-based classifiers, including linear, polynomial, and radial basis function models, applying the proposed filtering technique to five datasets. The results demonstrated improvements in accuracy, AUC, and F-measure, with the FILTER enhancing the performance of SVM-based SDP models by 16.73%, 16.80%, and 7.65%, respectively. The findings contribute to the advancement of SDP methodologies. Uddin *et al.* [21] presented an innovative software defect prediction model, SDP-BB, overcoming limitations of existing approaches. SDP-BB utilized Bidirectional Long Short-Term Memory networks (BiLSTM) and BERT-based semantic features to address shortcomings in manual code feature approaches. Unlike traditional models, SDP-BB incorporated semantic and contextual information from the source code. The BiLSTM captured contextual details through embedded token vectors from BERT, and an attention mechanism highlighted salient features. The approach employed data augmentation for enhanced training. Evaluation against state-of-the-art models on ten open-source projects demonstrated superior performance in fault prediction, outperforming competitors in terms of F1-score. The study contributed to advancing software defect prediction methodologies.

In the year 2023, Zhao *et al.* [22] conducted a systematic survey comprising 67 studies on Just-in-Time Software Defect Prediction (JIT-SDP). The survey aimed to advance research and familiarize practitioners with recent progress in JIT-SDP, a variant focused on predicting defects in incremental software changes. Results summarized best practices across JIT-SDP workflow phases, performed meta-analysis of prior studies, and suggested future research directions. Findings indicated that predictive performance correlated with change defect ratio, highlighting JIT-SDP's effectiveness in projects with higher defect ratios. Future directions included domain-specific application, reliability-aware, and user-centered approaches for JIT-SDP. Giray *et al.* [23] conducted a study that examined the use of deep learning (DL) in software defect prediction (SDP). The research

Table 1. Overview of related work.

Authors	Year	Focus Area	Techniques/ Models/Methods	Key Findings/Contributions
Thota *et al.* [14]	2020	Software Defect Prediction	Soft computing-based machine learning techniques	Proposed an efficient approach using soft computing-based machine learning for optimized feature prediction in high-cost software development.
Ning Li *et al.* [15]	2020	Unsupervised Learning Techniques	Fuzzy C-Means (FCM) and Fuzzy SOMs (FSOMs) in software defect prediction	Identified concerns in research and demonstrated the comparable performance of unsupervised models, particularly FCM and FSOMs, to supervised models in software defect prediction.
Matloob *et al.* [16]	2021	Ensemble Learning for SDP	Random forest, boosting, bagging methods; Analysis of papers from ACM, IEEE, Springer Link, Science Direct	Revealed commonly employed ensemble methods, highlighted promising frameworks, and emphasized the importance of feature selection in software defect prediction.
Akimova *et al.* [17]	2021	Deep Learning Techniques	Survey on software defect prediction using deep learning	Highlighted unresolved issues and new trends while examining recent advances in deep learning for software defect prediction.
Gong *et al.* [18]	2021	Software Dependency Network Analysis	Metrics extracted using Social Network Analysis (SNA)	Identified the impact of Software Dependency Network Analysis (SNA) metrics on software defect prediction models.
Khan *et al.* [19]	2022	Artificial Neural Networks	Systematic literature review on software defect prediction using ANNs	Analyzed trends and critical aspects of using ANNs in defect prediction, emphasizing the increasing demand for high-quality software systems.
Goyal [20]	2022	Support Vector Machines	Novel filtering technique (FILTER) for imbalanced datasets	Presented a brand-new filtering method to improve Support Vector Machine (SVM)-based defect prediction.
Uddin *et al.* [21]	2022	Bidirectional Long Short-Term Memory	Bidirectional Long Short-Term Memory networks (BiLSTM) and BERT-based semantic features	Proposed an innovative software defect prediction model (SDP-BB) utilizing BiLSTM and BERT-based semantic features for improved performance.
Zhao *et al.* [22]	2023	Just-in-Time Software Defect Prediction	Systematic survey on JIT-SDP	Summarized best practices, performed meta-analysis, and suggested future research directions for Just-in-Time Software Defect Prediction (JIT-SDP).
Giray *et al.* [23]	2023	Deep Learning In SDP	Examination of deep learning in software defect prediction	Proposed recommendations for future research in deep learning for defect prediction.
Stradowski and Madeyski [24]	2023	Business-Driven Mapping Study	Business-driven mapping study on machine learning in SDP	Provided insights into past and potential future research opportunities for businesses in machine learning software defect prediction.
Hernández-Molinos *et al.* [25]	2023	Bayesian Approaches	Evaluation of Bayesian approaches for software defect prediction	Compared classification results and discussed the robustness of Bayesian algorithms for defect prediction.

systematically analyzed 102 peer-reviewed studies, revealing that most studies had applied supervised DL, with two-thirds using metrics as input to DL algorithms. Convolutional Neural Network emerged as the most frequently used DL algorithm. The findings led to proposed recommendations, including the development of more comprehensive DL approaches for automatic feature extraction, the utilization of diverse software artifacts beyond source code, adoption of data augmentation techniques to address class imbalance, and encouragement of replication package publication. Stradowski and Madeyski [24] conducted a business-driven mapping study, analyzing the use of machine learning in software defect prediction and evaluated the state-of-the-art, identified trends, and assessed the potential for business adoption. Utilizing Scopus, authors analyzed 742 studies until February 23, 2022. Results showed a smaller use of commercial datasets, with academic considerations dominating. However, there were signs of in vivo results emerging. The study provided insights into past and potential future research opportunities in machine learning software defect prediction for businesses. Hernández-Molinos *et al.* [25] conducted a study on software defect prediction using Bayesian approaches. The research aimed to evaluate the three algorithms (K2, Hill Climbing, and TAN) to construct Bayesian Networks for classifying projects prone to defects. The choice was motivated by the underexplored use of Bayesian Networks, compared to the commonly used Naive Bayes. Using three public PROMISE datasets based on McCabe and Halstead complexity metrics, the results were compared with Decision Tree and Random Forest. The cross-validation process showed comparable classification results, with Bayesian algorithms exhibiting less variability and greater robustness compared to Decision Tree and Random Forest. **Table 1** presents a summary of significant contributions in the field of software defect prediction, including the focus area, techniques, models, and key findings or contributions of each study.

3. Methodology

A system model is proposed and represented in **Figure 1** for a robust examination of software defect prediction in an innovative hybrid machine learning framework. It is described below in brief.

3.1. Data Collection

The first phase of the methodology involves the systematic acquisition of diverse

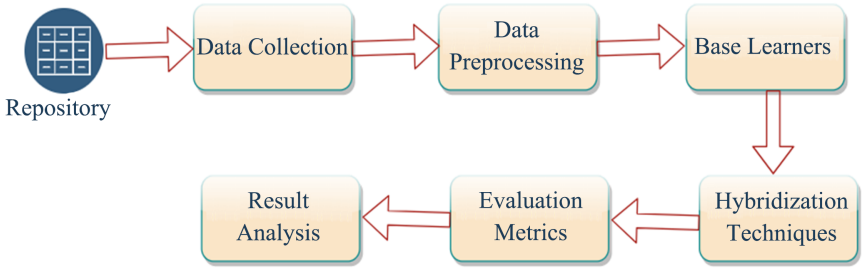

Figure 1. System model for software defect prediction using hybrid techniques.

and representative software defect datasets to form the empirical basis for subsequent algorithmic evaluation and comparative analysis. Through an exhaustive literature review, publicly available repositories housing software defect datasets, such as the PROMISE repository, were identified. Priority was given to datasets showcasing variability in project types, programming languages, and defect characteristics, ensuring a comprehensive assessment of the algorithm's performance. Rigorous criteria were applied to assess dataset quality, with a focus on completeness, balance, and relevance to the software defect prediction task. Statistical methods were employed to ensure the quality of datasets, validating their sufficiency in terms of instances and feature diversity. Additionally, attention was given to ethical considerations, respecting privacy and confidentiality aspects, and complying with licensing terms associated with each dataset. The selected datasets, integral to the subsequent analysis, were meticulously documented, detailing project types, defect types, and metadata such as the number of instances, features, and class distribution. These datasets were then retrieved from designated repositories, and version control mechanisms were implemented to track dataset versions, ensuring transparency, and facilitating reproducibility in subsequent analyses. The acquired datasets, along with their comprehensive metadata, form the foundational elements for the subsequent stages of the methodology. The study sources datasets from the PROMISE software engineering repository [26], encompassing CM1, JM1, KC1, and PC1. Key characteristics of the datasets, specifically the number of instances and attributes, are presented in Table 2 below.

The attribute of selected dataset is compiled in the following Table 3.

Preprocessing involves handling missing values, converting string labels to binary integers, and scaling numerical features to ensure dataset readiness for subsequent model training.

3.2. Data Pre-Processing

Following the systematic acquisition of software defect datasets, the next critical step in the methodology is data preprocessing. This phase is essential for preparing the selected datasets for subsequent model training, ensuring consistency and reliability in the analysis. The procedure begins with the comprehensive loading of the selected datasets into the analysis environment. Subsequently, a meticulous examination is conducted to identify and handle any missing values, employing advanced imputation methodologies to mitigate potential bias and maintain data integrity.

Categorical variables within the datasets are then subjected to conversion into a numerical format, leveraging state-of-the-art encoding techniques such as one-hot encoding or label encoding. This transformation is crucial for preserving information integrity and facilitating the compatibility of categorical data with machine learning algorithms. Furthermore, numerical features are normalized to ensure they are on a comparable scale. Techniques such as Min-Max

Table 2. Key characteristics of the datasets.

Dataset Name	Number of Instances	Number of Attributes
CM1	498	22
JM1	10885	22
KC1	2109	22
PC1	1109	22

Table 3. The attributes of the selected datasets.

Column Name	Data Type	Explanation
Loc	numeric	McCabe's line count of code
v_g	numeric	McCabe "cyclomatic complexity"
ev_g	numeric	McCabe "essential complexity"
iv_g	numeric	McCabe "design complexity"
N	numeric	Halstead total operators + operands
V	numeric	Halstead "volume"
L	numeric	Halstead "program length"
D	numeric	Halstead "difficulty"
I	numeric	Halstead "intelligence"
E	numeric	Halstead "effort"
B	numeric	Halstead
T	numeric	Halstead's time estimator
lOCode	numeric	Halstead's line count
lOComment	numeric	Halstead's count of lines of comments
lOBlank	numeric	Halstead's count of blank lines
uniq_Op	numeric	Unique operators
uniq_Opnd	numeric	Unique operands
total_Op	numeric	Total operators
total_Opnd	numeric	Total operands
branchCount	numeric	Percentage of the flow graph
Defects	boolean	Module has/has not one or more reported defects

scaling or StandardScaler are applied to promote homogeneity in feature magnitudes, preventing certain features from dominating the model training process due to scale.

The overarching objective of the data preprocessing phase is to create a standardized and conducive environment for subsequent model training and evalua-

tion. This involves addressing missing values, converting categorical variables, and normalizing numerical features, all while adhering to contemporary best practices in data preprocessing. The meticulously pre-processed datasets serve as the input for the subsequent phases of the methodology, ensuring that the algorithms are trained on consistent, high-quality data representations.

3.3. Base Learners

The base learners phase is a crucial step in algorithmic methodology, where one carefully selects foundational machine learning models to serve as individual components for constructing three distinct hybrid models. In our initial implementation, opted for RandomForestClassifier, XGBClassifier, and SVM as the base learners for Hybrid Model-1. The classifiers are well-known for strong performance in classification tasks, particularly in software defect prediction. Each chosen base learner undergoes individual training on meticulously pre-processed dataset, incorporating modern practices in model initialization, optimization, and validation.

In modified implementation, the algorithmic approaches for all three hybrid models are applied. For Hybrid Model-2, the initial base learners were replaced with AdaBoostClassifier, LogisticRegression, and DecisionTreeClassifier. For Hybrid Model-3, KNeighborsClassifier, GaussianNB, and SVC are selected. This strategic shift aims to explore different algorithmic paradigms and assess impact on model diversity and overall predictive performance. Similar to the initial implementation, each modified base learner undergoes individual training on the pre-processed dataset, ensuring alignment with industry best practices and established methodologies.

The selection of base learners for each hybrid model is grounded in the effectiveness in classification tasks and the goal is to explore a diverse set of algorithms. The individual models act as the foundational components upon which Hybrid Model-1 (SVM, RandomForest, XGBoost, Neural Network), Hybrid Model-2 (GradientBoosting, DecisionTree, LogisticRegression, Neural Network), and Hybrid Model-3 (KNeighbors, GaussianNB, SVC, Neural Network) will be constructed in subsequent phases. A thorough evaluation of the performance and contributions of each base learner will be conducted to determine the impact on the overall efficacy of the respective hybrid models in predicting software defects.

3.4. Hybridization Techniques

In the Hybridization Techniques phase of algorithmic methodology, predictions from diverse base learners are combined with the training of a neural network model to create robust and accurate hybrid models. The approach aims to leverage the strengths of individual base learners while harnessing the power of neural networks to enhance predictive performance. It is considered by incorporating predictions from a variety of base learners, including RandomFo-

restClassifier, XGBClassifier, SVM, AdaBoostClassifier, LogisticRegression, DecisionTreeClassifier, KNeighborsClassifier, GaussianNB, and SVC, into the original dataset features The predictions serve as additional features that capture different perspectives and patterns within the data. The predictions of each base learner are then blended with the original dataset features to train a neural network model utilizing the combined features. This neural network model comprises several layers, including dropout layers and dense layers with activation functions like sigmoid and ReLU, to improve generalization and decrease overfitting. During training, RMSprop optimization and binary cross-entropy loss to accomplish effective learning are applied. Three hybrid models are prepared by combining several base learner combinations in each. Hybrid Model-1 incorporates SVM, XGBoost, and RandomForest predictions together with the original features. AdaBoost, Logistic Regression, and Decision Tree predictions are combined with the initial features in Hybrid Model-2. The original characteristics are combined with predictions from GaussianNB, SVC, and KNeighbors in Hybrid Model-3.

Through this hybrid approach, the aim is to get benefit from the diverse perspectives offered by individual base learners while leveraging the flexibility and adaptability of neural networks to enhance predictive accuracy and robustness. The effectiveness of the hybrid models is rigorously evaluated and compared against individual base learners to assess their impact on overall predictive performance.

3.5. Evaluation Metrics

In the following section, several software defect prediction metrics, including false positive (FP), false negative (FN), true positive (TP), and true negative (TN), will be covered. The quantity of software instances correctly classified as clean (TN) is equal to the quantity of software instances correctly classed as faulty (TP). The number of clean software examples that are incorrectly categorized as defective is shown by the letters FP and the number of defective software instances that are incorrectly labelled as clean is indicated by the letters FN. Classification accuracy, also referred to as the right classification rate, is one of the primary simple metrics used to evaluate how well predictive models work. It is employed to measure the degree to which the cases that have been classified effectively relate to the total instances. A different metric known as precision is computed by dividing the total number of instances categorized as faulty (TP + FP) by the number of instances accurately classified as defective (TP) [13]. Furthermore, recall quantifies the proportion of accurately identified defective cases (TP) to the overall number of faulty cases (TP + FN) [13]. The F1-score is a statistic used in numerous research in the literature, which is a harmonic mean of precision and recall [27] [28]. ROC-AUC calculates the area under the receiver operating characteristic (ROC) curve by weighing the trade-offs between TPR and FPR. The following metrics are given by

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

These metrics collectively provide a comprehensive evaluation of predictive performance, capturing aspects of correctness, precision, recall, and overall discriminatory power. Customized functions are implemented to calculate these metrics for both individual base learners and hybrid models. The choice of metrics is grounded in established standards in predictive modeling assessment, ensuring a rigorous and unbiased evaluation process. The evaluation metrics phase serves as the benchmark for comparing the performance of different models, guiding the interpretation of results, and facilitating the identification of the most effective models in the context of software defect prediction.

3.6. Results Analysis

The Results Analysis phase involves a thorough examination and interpretation of the performance metrics obtained from evaluating individual base learners and hybrid models constructed through the algorithmic methodology. This critical step aims to derive actionable insights, discern patterns, and draw meaningful conclusions regarding the effectiveness of the models in predicting software defects.

4. Results and Discussion

By using the above concept, the following parameters are evaluated and described below in brief.

4.1. Accuracy

Accuracy is providing an overall assessment of correctness of model which is pivotal in evaluating the models' general performance. The accuracy values for the three hybrid combinations across projects are outlined in the following **Figure 2**. The provided figure succinctly presents accuracy values across three hybrid combinations for various projects, including CM1, JM1, KC1, and PC1. For the CM1 project, all three Hybrid Models consistently exhibit high accuracy values of 0.980, indicating a robust performance in correctly classifying instances across different combinations. In the case of the JM1 project, accuracy values range from 0.893 to 0.899, showcasing a generally high level of correctness across the Hybrid Models, albeit with slight variations. Moving to the KC1 project, accuracy values are consistently high, ranging from 0.950 to 0.953, indicating accurate classifications across different Hybrid Models. Lastly, for the PC1

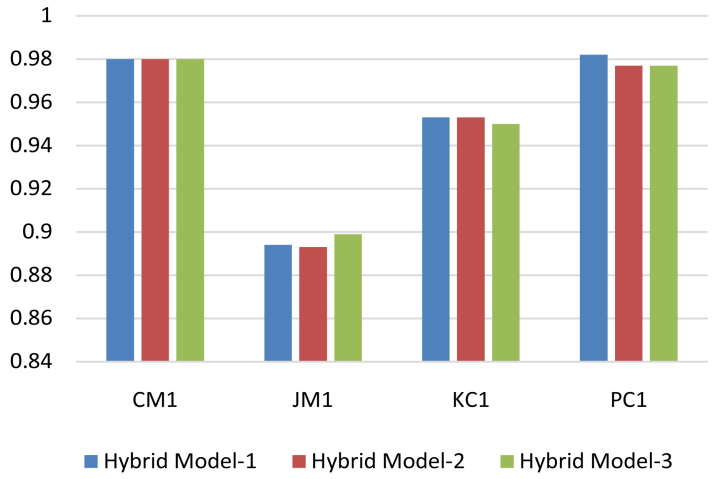

Figure 2. Accuracy scores for different hybrid models (y-axis represents accuracy, the x-axis represents model).

project, accuracy values range from 0.977 to 0.982, demonstrating a high level of correctness in classifying instances among the Hybrid Models. The accuracy results highlight competitive overall performance, with the third combination consistently demonstrating robust accuracy across all projects.

4.2. Precision

Precision is a measure of the accuracy of positive predictions, quantifies the proportion of correctly predicted defective instances among all instances predicted as defective. In the first, second, and third hybrid combinations, precision values for CM1, JM1, KC1, and PC1 are presented in **Figure 3**. The figure outlines Precision values, a key metric in software defect prediction, detailing the accuracy of positive predictions for different projects (CM1, JM1, KC1, and PC1) across three Hybrid Models (Hybrid Model 1, Hybrid Model 2, and Hybrid Model 3). Notably, CM1 demonstrates consistently perfect precision (1.000) across all models. For JM1, precision ranges from 0.853 to 0.891, reflecting some variability. In KC1, precision improves across models, ranging from 0.883 to 0.959. PC1 exhibits slight fluctuations in precision (0.889 to 0.941) among Hybrid Models. The figure succinctly presents insights into the accuracy of positive predictions for each project and model combination in the software defect prediction context. The precision results indicate that all combinations achieved high precision values, particularly in the third combination, showcasing a consistent ability to minimize false positives and enhance the accuracy of defective predictions. The trend is represented in the following **Figure 3**.

4.3. Recall

Recall, representing the ability to correctly identify all positive instances, is crucial for capturing actual defective instances among all actual defective instances. The recall values for the three hybrid combinations across projects are detailed in the following **Figure 4**.

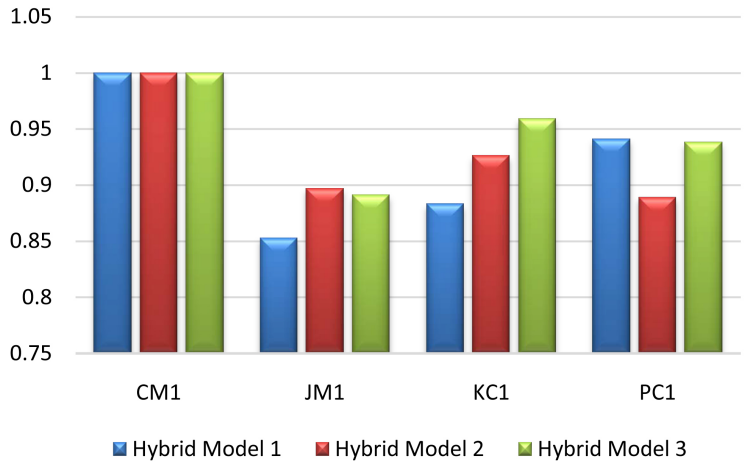

Figure 3. Precision scores for different hybrid models.

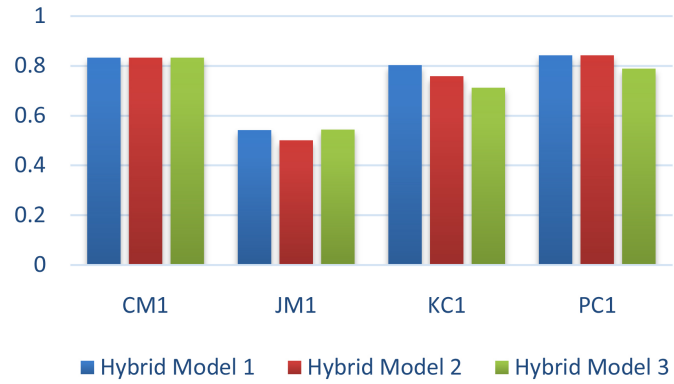

Figure 4. Recall scores for different hybrid models.

The presented figure details recall values for three hybrid combinations across different projects. For the CM1 project, all three Hybrid Models consistently exhibit a recall value of 0.833, indicating their effectiveness in correctly identifying actual defective instances. In the case of the JM1 project, recall values fluctuate between 0.501 and 0.544 across the Hybrid Models, highlighting variations in their ability to accurately identify positive instances. Moving on to the KC1 project, recall values range from 0.712 to 0.803, indicating differing degrees of efficacy in capturing actual defective instances across the Hybrid Models. Lastly, for the PC1 project, recall values vary from 0.789 to 0.842, revealing distinctions in the models' proficiency in correctly identifying positive instances. The recall results demonstrate a consistent performance across all combinations, with the third combination consistently exhibiting higher recall values, indicating its effectiveness in identifying actual defective instances.

4.4. F1-Score

The F1-Score is harmonizing precision and recall which offers a balanced assessment of model performance. Across the three hybrid combinations, F1-Score values for CM1, JM1, KC1, and PC1 are summarized in the following **Figure 5**.

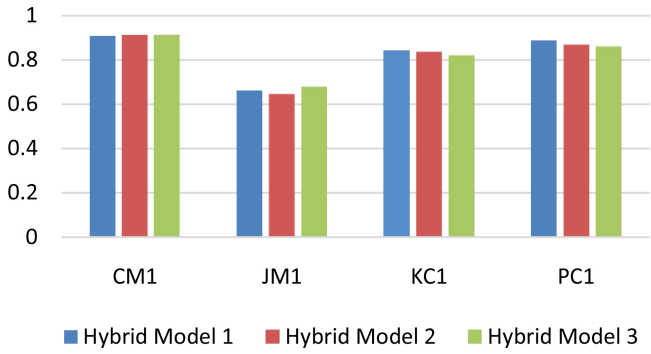

Figure 5. F1-Score scores for different hybrid models.

In the figure provided, F1-Score values across three hybrid combinations are outlined for distinct projects, namely CM1, JM1, KC1, and PC1. For the CM1 project, all three Hybrid Models consistently achieve an F1-Score of 0.909, signifying a balanced performance that effectively considers both precision and recall. In the case of the JM1 project, F1-Score values range from 0.643 to 0.676 across the Hybrid Models, indicating a generally balanced assessment of model performance with slight variability. Moving to the KC1 project, F1-Score values range from 0.817 to 0.841, showcasing a balanced performance with variations across different Hybrid Models. Lastly, for the PC1 project, F1-Score values fluctuate from 0.857 to 0.889, demonstrating a balanced assessment of model performance with slight fluctuations among the Hybrid Models. The F1-Score results underscore the third combination's consistent balanced performance, making it a valuable hybrid approach in software defect prediction.

4.5. ROC AUC

ROC AUC evaluates the trade-off between true positive rate and false positive rate, providing insights into the model's ability to distinguish between defective and non-defective instances. The ROC AUC values for the three hybrid combinations are presented in the following **Figure 6**.

Figure 6 provides ROC AUC values for three hybrid combinations across various projects, namely CM1, JM1, KC1, and PC1. For the CM1 project, all three Hybrid Models consistently display high ROC AUC values, ranging from 0.979 to 0.990, indicating robust performance in distinguishing between defective and non-defective instances. In the JM1 project, ROC AUC values range from 0.877 to 0.881 across Hybrid Models, highlighting the models' ability to make effective differentiations. For the KC1 project, ROC AUC values consistently range from 0.929 to 0.939, indicating a strong ability to discriminate between defective and non-defective instances across different Hybrid Models. In the PC1 project, ROC AUC values range from 0.976 to 0.990, demonstrating a high level of discriminative power among the Hybrid Models. The ROC AUC results emphasize the third combination's superior ability to discriminate between defective and non-defective instances.

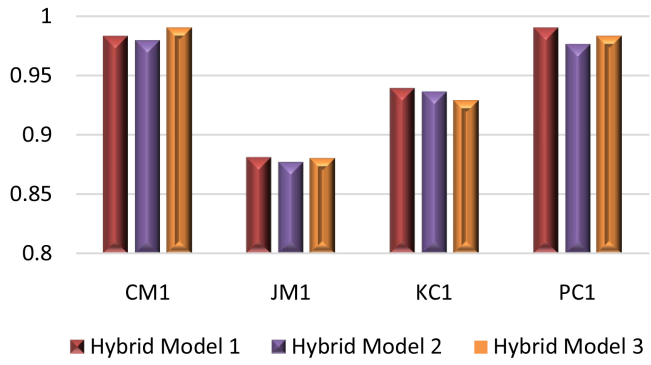

Figure 6. ROC AUC scores for different hybrid models.

The third combination consistently outperforms the other combinations across precision, recall, F1-Score, accuracy, and ROC AUC. This hybrid approach demonstrates a remarkable ability to balance precision and recall, leading to robust overall performance in software defect prediction. The precision and recall metrics emphasize the model's capability to minimize false positives and false negatives, while the F1-Score and accuracy metrics indicate a balanced and accurate predictive performance. Additionally, the ROC AUC values highlight the discriminative power of the models, especially in the Third Combination. These results collectively underscore the significance of carefully selecting and combining base classifiers in hybrid approaches, demonstrating their potential to enhance the accuracy and reliability of defect prediction models.

5. Conclusion

Our analysis of hybrid machine-learning techniques for software defect prediction highlights the consistent superiority of the third hybrid combination, incorporating KNeighborsClassifier, GaussianNB, SVC, and Neural Network, across key metrics (Accuracy, Precision, Recall, F1-Score, and ROC AUC) for various projects (CM1, JM1, KC1, PC1). This combination's balanced approach effectively minimizes false positives and false negatives, underscoring the significance of thoughtful classifier selection in hybrid models. These results make a valuable contribution to the continuous effort to enhance the dependability and precision of defect prediction in the field of software engineering. Future research may explore additional classifier combinations and feature engineering strategies to further enhance software defect prediction models.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Murphy, K.P. (2012) Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge.

[2] Dietterich, T.G. (2000) Ensemble Methods in Machine Learning. In: *International Workshop on Multiple Classifier Systems*, Springer, Berlin, 1-15. https://doi.org/10.1007/3-540-45014-9_1

[3] Cristianini, N. and De Bie, T. (2005) Support Vector Machines. Hodder Arnold, London.

[4] Breiman, L. (2001) Random Forests. *Machine Learning*, **45**, 5-32. https://doi.org/10.1023/A:1010933404324

[5] Chen, T. and Guestrin, C. (2016) Xgboost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, 13-17 August 2016, 785-794. https://doi.org/10.1145/2939672.2939785

[6] James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013) An Introduction to Statistical Learning. Vol. 112, Springer, New York, 18. https://doi.org/10.1007/978-1-4614-7138-7

[7] Friedman, J.H. (2001) Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, **29**, 1189-1232. https://doi.org/10.1214/aos/1013203451

[8] Podgorelec, V., Kokol, P., Stiglic, B. and Rozman, I. (2002) Decision Trees: An Overview and Their Use in Medicine. *Journal of Medical Systems*, **26**, 445-463. https://doi.org/10.1023/A:1016409317640

[9] Bishop, C.M. (2006) Pattern Recognition and Machine Learning by Christopher M. Bishop. Springer Science+ Business Media, Berlin.

[10] Hastie, T., Tibshirani, R., Friedman, J.H. and Friedman, J.H. (2009) The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Vol. 2, Springer, New York, 1-758. https://doi.org/10.1007/978-0-387-84858-7

[11] Burges, C.J. (1998) A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, **2**, 121-167. https://doi.org/10.1023/A:1009715923555

[12] http://promise.site.uottawa.ca/SERepository/datasets-page.html

[13] Fawcett, T. (2006) An Introduction to ROC Analysis. *Pattern Recognition Letters*, **27**, 861-874. https://doi.org/10.1016/j.patrec.2005.10.010

[14] Thota, M.K., Shajin, F.H. and Rajesh, P. (2020) Survey on Software Defect Prediction Techniques. *International Journal of Applied Science and Engineering*, **17**, 331-344.

[15] Li, N., Shepperd, M. and Guo, Y. (2020) A Systematic Review of Unsupervised Learning Techniques for Software Defect Prediction. *Information and Software Technology*, **122**, Article ID: 106287. https://doi.org/10.1016/j.infsof.2020.106287

[16] Matloob, F., Ghazal, T.M., Taleb, N., Aftab, S., Ahmad, M., Khan, M.A. and 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

[17] Akimova, E.N., Bersenev, A.Y., Deikov, A.A., Kobylkin, K.S., Konygin, A.V., Mezentsev, I.P. and Misilov, V.E. (2021) A Survey on Software Defect Prediction Using Deep Learning. *Mathematics*, **9**, Article No. 1180. https://doi.org/10.3390/math9111180

[18] Gong, L., Rajbahadur, G.K., Hassan, A.E. and Jiang, S. (2021) Revisiting the Impact of Dependency Network Metrics on Software Defect Prediction. *IEEE Transactions on Software Engineering*, **48**, 5030-5049. https://doi.org/10.1109/TSE.2021.3131950

[19] Khan, M.A., Elmitwally, N.S., Abbas, S., Aftab, S., Ahmad, M., Fayaz, M. and Khan,

F. (2022) Software Defect Prediction Using Artificial Neural Networks: A Systematic Literature Review. *Scientific Programming*, **2022**, Article ID: 2117339. https://doi.org/10.1155/2022/2117339

[20] Goyal, S. (2022) Effective Software Defect Prediction Using Support Vector Machines (SVMs). *International Journal of System Assurance Engineering and Management*, **13**, 681-696. https://doi.org/10.1007/s13198-021-01326-1

[21] Uddin, M.N., Li, B., Ali, Z., Kefalas, P., Khan, I. and Zada, I. (2022) Software Defect Prediction Employing BiLSTM and BERT-Based Semantic Feature. *Soft Computing*, **26**, 7877-7891. https://doi.org/10.1007/s00500-022-06830-5

[22] Zhao, Y., Damevski, K. and Chen, H. (2023) A Systematic Survey of Just-in-Time Software Defect Prediction. *ACM Computing Surveys*, **55**, 1-35. https://doi.org/10.1145/3567550

[23] Giray, G., Bennin, K.E., Köksal, Ö., Babur, Ö. and Tekinerdogan, B. (2023) On the Use of Deep Learning in Software Defect Prediction. *Journal of Systems and Software*, **195**, Article ID: 111537. https://doi.org/10.1016/j.jss.2022.111537

[24] Stradowski, S. and Madeyski, L. (2023) Machine Learning in Software Defect Prediction: A Business-Driven Systematic Mapping Study. *Information and Software Technology*, **155**, Article ID: 107128. https://doi.org/10.1016/j.infsof.2022.107128

[25] Hernández-Molinos, M.J., Sánchez-García, A.J., Barrientos-Martínez, R.E., Pérez-Arriaga, J.C. and Ocharán-Hernández, J.O. (2023) Software Defect Prediction with Bayesian Approaches. *Mathematics*, **11**, Article No. 2524. https://doi.org/10.3390/math11112524

[26] Elish, K.O. and Elish, M.O. (2008) Predicting Defect-Prone Software Modules Using Support Vector Machines. *Journal of Systems and Software*, **81**, 649-660. https://doi.org/10.1016/j.jss.2007.07.040

[27] Kim, S., Zhang, H., Wu, R. and Gong, L. (2011) Dealing with Noise in Defect Prediction. *Proceedings of the* 33*rd International Conference on Software Engineering*, Honolulu, 21-28 May 2011, 481-490. https://doi.org/10.1145/1985793.1985859

[28] Lee, T., Nam, J., Han, D., Kim, S. and In, H.P. (2011) Micro Interaction Metrics for Defect Prediction. *Proceedings of the* 19*th ACM SIGSOFT Symposium and the* 13*th European Conference on Foundations of Software Engineering*, September 2011, 311-321. https://doi.org/10.1145/2025113.2025156