Journal of Theoretical and Applied Information Technology
31st July 2024. Vol.102. No. 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPTIMIZED ENSEMBLE LEARNING FOR SOFTWARE DEFECT PREDIC-TION WITH HYPERPARAMETER TUNING

GETACHEW MEKURIA HABTEMARIAM[1], SUDHIR KUMAR MOHAPATRA[2,*], HUSSIEN WORKU SEID[3]

[1]Addis Ababa Science and Technology University Addis Ababa, Addis Ababa, Ethiopia
[2,*]Sri Sri University, Cuttack, Odisha, India
[3]Addis Ababa Science and Technology University Addis Ababa, Addis Ababa, Ethiopia

E-mail: [1]getachewmekuria19@gmail.com, [2]sudhir.mohapatra@srisriuniversity.edu.in, [3]hussien.seid@aastu.edu.et
*Corresponding Author

ABSTRACT

The Software plays a crucial role in human life, making it essential for system developers to have reliable and accurate software. The discovery of faults during software development is becoming increasingly important to minimize costs and delivery time. As the application of software in business increases then, the soundness of software becomes more important. Several logical models have been proposed to evaluate software system reliability and predict software trustworthiness but the existing reliability model may be efficient towards solving a specific type of problem but incapable of solving other classes of software problems. Therefore, a novel and universal model is needed for fair prediction and error classification of all types of software reliability prediction problems. The study proposed ensemble models for software reliability prediction, which have an advantage over existing statistical and machine learning models. The proposed model is a binary model that can be used for error classification with automatic hyperparameter selection for flexibility. A total of 21 static metrics of the NASA dataset are taken as independent variables for the classification model and bagging, voting and stacking techniques are applied for classification. The performance of the models was evaluated using accuracy, precision; recall and f1-score and the model achieved 89.1% classification accuracy. The proposed ensemble model was also compared with existing models using a benchmark dataset for their performance. The results of the statistical comparison of the proposed model show better performance as compared to other existing models.

Keywords: *Software reliability, Software reliability prediction, Software reliability classification, Ensemble model, Machine learning, Hyperparameter*

1. INTRODUCTION

Software engineering is a disciplined and systematic approach to software development. It encompasses the entire lifecycle, from understanding user needs, requirement gathering, and design to development, testing, dep-loyment, and maintenance. [1] Software has become an indispensable part of any modern, complex system. Many industries, such as telecommunications, automotive, aviation, e-commerce, entertainment, and critical sectors like healthcare and nuclear power plants. [2] All these applications need precise and reliable software in order to provide the required services and performances. Software failure in such applications can lead to high costs and, in some cases, irreparable or catastrophic damages. Recognizing this danger, software developers have prioritized implementing rigorous quality checks throughout the development lifecycle.

In software engineering, software quality assurance and testing (SQAT) is a crucial part of the software de-velopment process, ensuring that the software meets the specified requirements and delivers high-quality soft-ware [3]. High-quality software's heart lies in reliability, which denotes the probability of failure-free software operation for a specific period of time in a specified environment [4]. Reliable software exhibits minimal fail-ures and delivers accurate results. The critical need to evaluate software reliability underscores its status as a cornerstone of software quality. By quantifying software errors, bugs, faults, and defects that can lead to fail-ures, software reliability allows us to measure and improve the overall robustness of a program.

In the quest for dependable software, software reliability prediction (SRP) emerges as a powerful tool [5]. It empowers development teams and stakeholders to make informed decisions. Stakeholders can better allocate resources, prioritize testing efforts, and determine release dates. By identifying potential problems early, devel-opers can address them proactively, saving time, and resources, and avoiding potential disasters, ultimately leading to the delivery of high-quality and reliable software. SRP relies on various models that analyze data like code complexity and historical faults to estimate future failures. The accuracy of these predictions hinges on data quality and choosing the right prediction techniques [6].

Several different techniques have been studied to predict the reliability of software modules, aiming to im-prove software quality and reduce software development costs. Traditionally, SRP has utilized various probabilistic methods, but a significant gap persists: the need for a robust and adaptable predictive framework capable of navigating the complexities of modern software systems [7]. This gap presents an opportunity for innovative solutions to emerge, with a wide spectrum of machine learning approaches, including Decision Trees, Naïve Bayes (NB), Radial Basis Function, Support Vector Machine (SVM), K-Nearest Neighbor, Multi-layer Perceptron, and Random Forest (RF), being explored to anticipate errors in software modules and ultimately enhance software quality while reducing testing costs [8].

While these techniques have been employed in the past to predict software reliability, the necessity for a robust and adaptable predictive framework persists, capable of addressing the complexities of modern software. By integrating diverse modeling approaches and refining parameter selection methods, a research gap continues to exist. Further exploration of adaptable and effective strategies for SRP is imperative to ensure better adapta-tion to evolving software landscapes and to yield results that are both more accurate and reliable.

Henceforth, to bridge these gaps the authors have introduced a novel approach employing ensemble learning and a random hyperparameter selection algorithm. Ensemble learning aggregates predictions from multiple models, thereby creating a more robust and potentially more accurate forecast of software reliability. Additionally, the random hyperparameter selection algorithm defines a range of possible values for each hyperparameter within the models and then randomly selects a combination. This randomness facilitates the exploration of a broader range of possibilities and helps to potentially avoid being trapped in local optima within a single model configuration. Through the fusion of ensemble learning with random hyperparameter selection, the study achieves more accurate and reliable SRPs.

This research study makes the following contributions:

- Proposed a novel ensemble learning approach with random hyperparameter selection: This paper introduces a new method for software reliability prediction that combines the strengths of ensemble learning and random hyperparameter selection. By leveraging multiple models with a variety of configurations, the study resulted in more robust, accurate, and potentially generalizable predictions compared to previous state-of-the-art approaches.
- Implemented a prototype of the proposed model: In a Python programming language and conducted an empirical analysis to evaluate its effectiveness and efficiency. Through rigorous experimentation, the study demonstrated the capability of accurately predicting software reliability while considering various factors and complexities inherent in modern software systems.
- Potential for improved prediction accuracy: The combination of ensemble learning and random hyperparameter selection has the potential to improve the accuracy of software reliability predictions compared to existing methods. This can lead to more informed decision-making during the software development process.

The subsequent sections of this study are structured as follows. Section 2 encompasses an extensive literature review. Section 3 concisely outlines the materials and methods proposed methodology. The exploration of experimental results, along with a comparative analysis of proposed approach against other methodologies, is detailed in Section 4. Section 5 encapsulates the conclusion and recommendation of the study.

2. LITERATURE REVIEW

The field of software reliability prediction is continually evolving, with researchers exploring innovative methodologies to improve the accuracy

and efficiency of forecasting software defects. While there's no univer-sal solution, numerous studies in the literature contribute to this ongoing investigation [7, 8]. This review sets the stage for further investigation by discussing the existing methodologies and approaches, highlighting their strengths and limitations. Subsequently, the following section lists related works that complement and extend the understanding of SRP techniques, paving the way for proposed improvements and novel approach.

Rath et al. [9] proposed a hybrid software reliability prediction model that combines feature selection with a Support Vector Classifier (SVM), achieving successful validation on the standard NASA Metrics Data Pro-gram datasets. They achieved improved performance metrics compared to the existing model. Wu et al. [10] proposed a Hybrid Multi-layer Heterogeneous Particle Swarm Optimization Algorithm (HMHPSO) to optimize a GRU neural network for software reliability prediction, addressing the limitations of low accuracy and weak generalization in current models.

Dong et al. [11] investigated ensemble learning for software defect prediction, demonstrating its superiority over seven individual machine learning and deep learning algorithms on four public datasets from NASA and PROMISE. Their ensemble approach achieved a high AUC of 0.99, G-Mean of 0.96, and an average F1-score of 0.97, outperforming all single models and offering a good balance between performance and runtime for time-sensitive scenarios. Moeini et al. [12] proposed a combination of machine learning and approximation Bayesian inference to address limitations in both parametric and non-parametric software reliability prediction models. They evaluated the effectiveness of their approach using three real-world software failure datasets with varying sizes.

Rath et al. [13] proposed a novel approach for software reliability prediction using an Extreme Learning Machine (ELM) combined with feature selection. This approach addresses the limitations of existing models in cross-system defect prediction, where past data from one project is used to predict defects in a new project. Their experiments on NASA Metrics Data Program datasets demonstrated that the proposed model achieved superior prediction accuracy compared to traditional ELM defect prediction models. Yadav et al. [14] proposed software reliability prediction utilizing a dense neural network implemented using deep learning. The experiment was performed on twelve datasets from different sources. Their results are evaluated using four standard performance metrics which are accuracy, precision, recall, and f1-score. However, their study is limited to small datasets and can be extended with large industrial datasets to achieve better results.

Jabeen et al. [15] proposed an improved software reliability prediction model by using a high-precision er-ror iterative analysis method based on residual errors. The residual error values between actual and estimated values are used iteratively to improve the fitting of historical failure data. The artificial neural network was also used to estimate signs, which are associated with residual errors. Qiaoa et al. [16] proposed a deep learning-based approach to predict the number of defects in software modules. They compared the proposed techniques with state-of-the-art approaches like SVR, FSVR, and DTR on two well-known datasets. Their approach per-formed better in reducing the mean square error (varying between 3% and 13%) and improving the squared correlation coefficient (varying between 2% and 27%).

Manjula et al. [17] proposed a deep neural network-based hybrid approach for software defect prediction using software metrics. Their approach employs a genetic algorithm (GA) for feature optimization and a deep neural network (DNN) for classification. The proposed approach was implemented on benchmark datasets that were obtained from the PROMISE repository. The performance was compared with existing classification schemes such as Naïve Bayes, SVM, Decision Tree, and KNN in terms of classification accuracy, sensitivity, specificity, precision, and recall. The experimental analysis showed that their approach improved performance.

Kim et al. [18] studied a prediction and comparative analysis of the software reliability models using the nonhomogeneous Poisson-process model (NHPP) and deep learning. Their approach relied on data rather than mathematical and statistical assumptions. A software reliability model based on recurrent neural networks (RNN), long short-term memory (LSTM), and gated recurrent units (GRU), which are the most basic deep and recurrent neural networks that have been applied to time-series data characteristics, was used. It was constructed by including the hidden

layer of the neural network, and using two datasets, it was observed that their model showed better estimation and predictive power[19,20,21].

In general, the reviewed literature underscores the diverse methodologies employed in SRP, ranging from traditional machine learning to advanced deep learning algorithms. While significant progress has been made in enhancing prediction accuracy and generalization across diverse datasets, challenges persist in adapting these models to the complexities of modern software landscapes. The need for a robust and adaptable predictive framework capable of addressing evolving software environments remains paramount. By integrating diverse modeling approaches and refining parameter selection methods, researchers continue to strive towards more accurate and reliable SRP models. Moreover, as software systems evolve in complexity and dynamics, there is a growing demand for methodologies that can dynamically adapt to changing conditions and requirements. To address these challenges and bridge existing gaps, this study proposed a novel approach leveraging ensemble learning and a random hyperparameter selection algorithm, offering a promising avenue for enhancing SRP models and addressing ongoing challenges in the field. Through empirical validation and comparative analysis, the study contributes to the ongoing advancement of SRP methodologies, paving the way for more reliable and efficient software development practices.

Table 1: Tabular summary of related works.

Authors	**Proposed Model /Approach**	**Datasets used**	**Key Findings / Results**
Rath et al [**9**]	Hybrid software reliability prediction model combining feature selection with SVM	NASA Metrics Data Program datasets	Improved performance metrics compared to existing models
Wu et al. [**10**]	Hybrid Multi-layer Heterogeneous Particle Swarm Optimization Algorithm (HMHPSO) to optimize GRU neural network for reliability prediction	5 public datasets from NASA and Bell Lab	Addressed low accuracy and weak generalization in current models
Dong et al [**11**].	Investigated ensemble learning for software defect prediction	4 public datasets from NASA and PROMISE	Ensemble approach outperformed individual machine learning and deep learning algorithms
Moeini et al [**12**]	Combination of machine learning and approximation Bayesian inference for software reliability prediction	3 real-world software failure datasets with varying sizes	Effectiveness evaluated using real-world datasets
Rath et al [**13**]	Novel approach using Extreme Learning Machine (ELM) combined with feature selection for software reliability prediction	NASA Metrics Data Program datasets	Superior prediction accuracy compared to traditional ELM defect prediction models
Yadav et al. [**14**]	Software reliability prediction using a dense neural network implemented with deep learning	12 datasets from different sources	Performance was evaluated using accuracy, precision, recall, and F1-score. Limited to small datasets, suggests extension for better results

Jabeen et al. [**15**]	Improved reliability prediction model using high-precision error iterative analysis method based on residual errors	2 well-known datasets	Used residual error values iteratively to improve the fitting of historical failure data
Qiaoa et al [**16**]	Deep learning based approach for predicting number of defects in software modules	2 well-known datasets	Outperforme d state-of-the-art approaches like SVR, FSVR, and DTR in terms of mean square error reduction and squared correlation coefficient improvement
Manjula et al [**17**]	Deep neural network-based hybrid approach for software defect prediction using genetic algorithm and DNN	Benchmar k datasets from the PROMISE repository	Improved performance compared to existing classification schemes such as Naïve Bayes, SVM, Decision Tree, and KNN
Kim et al. [**18**]	Prediction and comparative analysis of software reliability models using NHPP and deep learning	2 datasets	Deep learning model showed better estimation and predictive power compared to NHPP model

3. METHODOLOGY

This section outlines section of this study presents the proposed hyperparameterized ensemble model design, dataset characteristics, and techniques, including data pre-processing, feature selection, and proposed classification approach for software reliability prediction.

3.1 The Model Design

The proposed model design encompasses an ensemble learning approach specifically tailored for software reliability prediction. This methodology leverages multiple distinct models, such as AdaBoost, Gradient Boosting, and Random Forest, to enhance overall effectiveness and generalization capacity while mitigating overfitting risks. The classification model operates within the realm of binary classification, aiming to categorize errors present within the software. It analyzes a spectrum of input factors linked to the software's characteristics to discern the presence of flaws. By amalgamating insights from diverse models, ensemble techniques elevate classification accuracy while mitigating biases and errors. Hyperparameter selection for the models is accom-plished through a random search technique, optimizing model performance, are not inherently discovered during training. Manual adjustment of hyperparameters can be laborious and inefficient. To optimize model performance, automatic hyperparameter search techniques such as Grid Search, Random Search, or Bayesian Optimization are employed to effectively explore the hyperparameter space. These methodologies are essential for enhancing classification accuracy, scheduling maintenance, and minimizing the impact of system malfunctions.

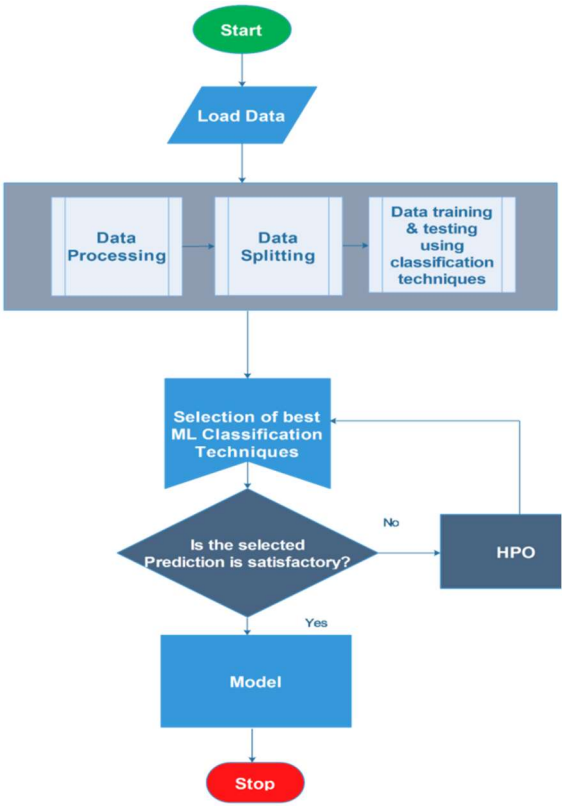

Figure 1: The flow chart of proposed model design

3.2 Dataset

In our study of two proposed models, we employed two openly available data sources. The first model utilized the PROMISE Software Engineering Repository's CM1 NASA dataset, which incorporates data from the NASA Metrics Data Program and the C-based NASA spacecraft instrument [**22**], detailed in Table 2.

Table 1: CM1 NASA Dataset information

Title	Language	Source code	Modules	Features	Defective	Defect Free	Defect Rate
CM1	C	NASA Spacecraft Instrument	498	22	49	449	9.83%

3.2.1 NASA Dataset

The NASA dataset is a labeled dataset containing information about software modules, focusing on their characteristics and defect presence. Each entry represents a software module, consisting of 22 columns or features. These features offer a comprehensive overview of the software's characteristics, enabling classification models in predicting whether a module is likely to have reported defects. The dataset consists of 21 static metrics or columns, providing insights into various aspects of software complexity and size. Column 22 (Label) contains Boolean values, indicating the presence or absence of reported defects in the software module. A value of True indicates the presence of one or more reported defects (positive class), while False indicates the absence of reported defects (negative class) as illustrated in Table 3. As a result, this dataset is used for classification tasks, where the goal is to predict whether a software module is likely to have defects based on its static metrics. Machine learning models can be trained on this dataset to classify modules as either defective or defect-free, helping software developers identify potential problem areas and prioritize testing and debugging efforts.

Table 3: NASA dataset 22 columns or features

Categories of metrics	Dataset Information	Attribute
4 McCabe metrics	01	loc: McCabe's line count of code
	02	v(g): McCabe cyclomatic complexity
	03	ev(g): McCabe essential complexity
	04	iv(g): McCabe design complexity
12 Base and derived Halstead metrics	05	N: Halstead total operators + operands
	06	V: Halstead volume
	07	L: Halstead program length
	08	D: Halstead difficulty
	09	I: Halstead intelligence
	10	E: Halstead effort to write program
	11	B: Delivered bugs
	12	T: Halstead's time estimator
	13	IOCode: Halstead's line count
	14	IOComment: Halstead' count of lines of comments
	15	IOBlank: Halstead's count of blank lines
	16	IOCodeAndComment: line of code and comments
2 operators, 2 operands, a branch-count and a goal field metrics	17	Uniq_Op: unique operators
	18	Uniq_Opnd: unique operands
	19	Total_Op: total operators
	20	Total_Opnd: total operands
	21	branchCount: of the flow graph
	22	D: module has defects or not

3.3 Techniques

3.3.1 Dataset Preprocessing

The benchmarked NASA dataset has been preprocessed to ensure its integrity for subsequent analysis, addressing missing values, outliers, and noise. As a result, no further preprocessing steps are necessary before utilizing this dataset for classification tasks. Visual inspection of individual feature distributions is commonly performed using boxplots to identify outliers, as demonstrated in

Journal of Theoretical and Applied Information Technology
31st July 2024. Vol.102. No. 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 2. These outliers, however, are contextual or conditional rather than genuine anomalies in the dataset. Specifically, the outliers observed in the "Lines of Code (LOC)" feature may reflect the natural variability in software size across different projects due to various contextual factors. Understanding this data distribution is crucial for accurate data analysis, as it provides insights into data properties and trends for informed decision-making and forecasting.

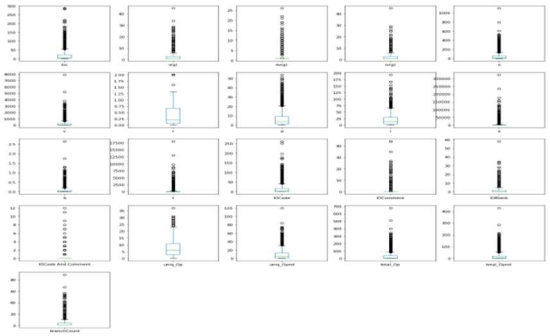

Figure 1: Box plot of the NASA datasets showing outliers

3.3.2 Feature Selection

In the context of feature selection, heatmaps serve as a valuable tool for visualizing the correlations among variables within a dataset. These visualizations depict correlation coefficients between pairs of variables, with varying colors indicating the strength and direction of correlation. In our study, we present the heatmap of features from the NASA dataset in Figure 3.

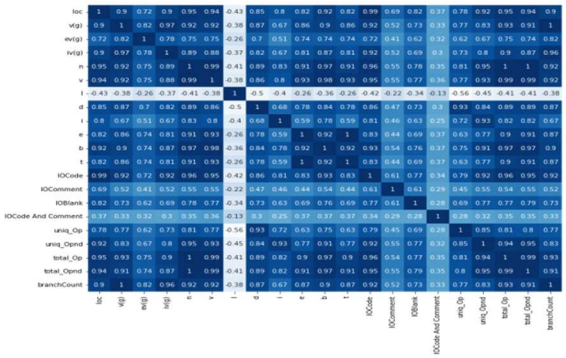

Figure 2: Heatmap of the features of the NASA dataset

Notably, the diagonal of the heatmap represents the correlation of each variable with itself, with a correlation value of 1 indicating a perfect positive correlation. This signifies a flawless linear relationship between the variable and itself. Moreover, feature importance graphs offer a succinct and intuitive means of understanding intricate machine learning models. They help in comprehending which features hold the most influential in prediction-making, thereby enhancing model interpretability and trust. Our analysis of the NASA dataset, as depicted in Figure 4, highlights LOC (Lines of Code) as possessing the highest F-score at 29.0, while the Halstead Time Estimator (t) registers the lowest at 6.0, underscoring their respective degrees of influence.

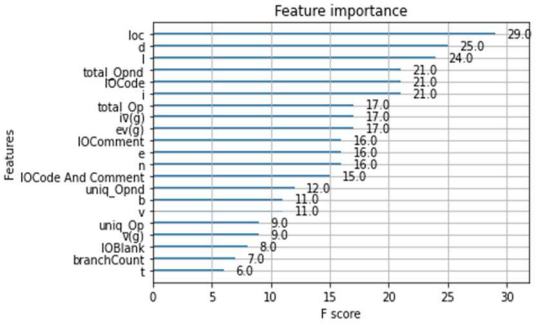

Figure 3: Features importance of NASA Dataset

3.3.3 Classification

In the context of classification problem, the objective is to categorizing software modules based on parameters like cyclomatic complexity, significance complexity, blueprint complexity, and line count, to determine whether they will likely have one or more reported defects. To address this classification task, we employed a range of models including the Bagging Classifier, Random Forest Classifier, Extra Trees Classifier, AdaBoost Classifier, Gradient Boosting Classifier, Voting Classifier, XGBoost Classifier, MLP Classifier, and SVM Classifier, as outlined in Figure 5.

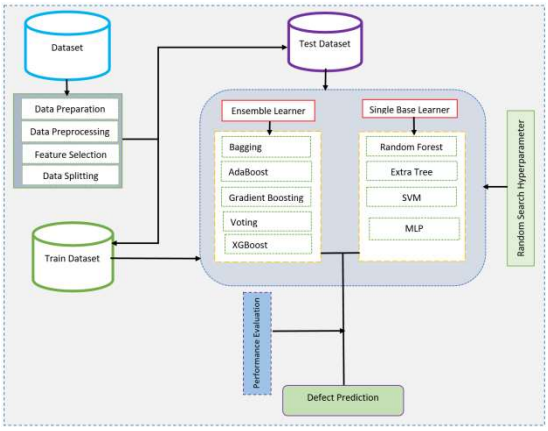

Figure 4: The proposed Ensemble Model for classification

Figure 5 outlines our model selection process. We utilized the NASA failure dataset, comprising 22 features with 21 independent variables and 1 dependent variable. Before training the models, thorough preprocessing was conducted to handle missing values, outliers, and noise effectively. With a dataset containing 2109 data points, we adopted a standard train-test split of 70:30 for model evaluation.

Table 4: Classifiers along with their parameters and model details

Classifiers	Model parameter setting
BaggingClassifier	Base Estimator: DecisionTree Classifier. Number of Estimators: 1500. Random State: 42.
RandomForestClassifier	Number of Estimators: 1000. Random State: 42.
ExtraTreesClassifier	Number of Estimators: 1000. Maximum Features: 7. Random State: 42.
AdaBoostClassifier	Number of Estimators: 30.
GradientBoostingClassifier	Number of Estimators: 100 Random State: 42.
VotingClassifier:	Estimators: 30. LogisticRegression with 'liblinear' solver. DecisionTreeClassifier. SVC with 'scale' gamma.
XGBClassifier (XGBoost):	Objective: Binary Logistic. Colsample by Tree: 0.3; Learning Rate: 0.1. Max Depth: 5; Alpha: 10. Number of Estimators: 50.
MLPClassifier (Multi-layer Perceptron):	Hidden Layer Sizes: (22, 22, 22). Activation Function: ReLU. Max Iterations: 1000
SVC (Support Vector Classifier):	Gamma: 'scale'

These classifiers are designed to address the classification task using the provided dataset, each employing different techniques and strategies to capture the underlying patterns and relationships within the data. We utilize 10-fold cross-validation to evaluate the performance of the machine learning models. This methodology involves dividing the dataset into 10 subsets. The model undergoes training and evaluation ten times, with each subset serving as the validation set once while utilizing the remaining subsets for training. Finally, the efficacy of our proposed model was rigorously assessed using a comprehensive set of software reliability metrics. These metrics, including precision, accuracy, recall, f1-score, support, performance error measures, and a confusion matrix, provided a holistic evaluation of the model's performance. Through this thorough analysis, we gained valuable insights into the effectiveness and robustness of our approach in addressing the challenges of software reliability. Moving forward, these findings serve as a foundation for further refinement and optimization, ultimately enhancing the reliability and efficiency of software systems.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

The experimental setup for our study involved the implementation of all models in Python, utilizing both a local workstation and the cloud-based Google Colab platform equipped with GPU support. The local platform, a Windows laptop, featured an Intel(R) Core(TM) i5-2410 M processor, 6 GB of primary storage, and 1 TB of secondary storage. The Colab platform provided additional computational power through GPU acceleration. The local platform utilized the Visual Studio Code editor, while Jupiter Notebook was exclusively utilized in the Colab environment. All models were compiled and executed using Python 3 compilers. The implementation of the models heavily relied on the Pandas, Keras, and Sci-kit learn libraries, alongside TensorFlow and the matplotlib library for data visualization in Python. This setup ensured a robust and efficient environment for conducting our machine learning classification experiments.

Journal of Theoretical and Applied Information Technology
31st July 2024. Vol.102. No. 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.2 Experimental Results

In our experimental results, the first proposed model focuses on employing classification techniques to identify software errors. Utilizing Bagging, Voting, Stacking, and XGBoost methods within the Python programming language, we address software reliability classification tasks with a CM1 dataset comprising 21 static features. Effective model performance depends on the training set's size and representativeness, enabling accurate predictions for unseen data instances. Concurrently, the validation set aids in refining the model by analyzing its performance during training, thereby facilitating performance evaluation and modification. By assessing the model's generalizability on the validation set, potential issues like overfitting are identified, crucial for ensuring robust functionality in real-world scenarios. Figure 6 illustrates the training and validation accuracy of our dataset, providing insights into the model's performance.

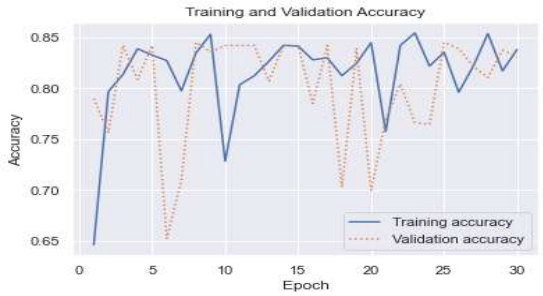

Figure 5: Training and validation accuracy comparison of the dataset

In the proposed classification approach, nine distinct base classifiers are employed to classify errors. Alongside, confusion matrices, maximum, minimum, and mean values, along with the final accuracy outputs, are printed, offering clear insights into each model's classification performance. Figures 7 and 8 visualize the accuracy of each model, with Random Forest, Bagging, and Extra Tree achieving the highest classification accuracies during training. Conversely, XGBoost attains the highest classification accuracy during testing.

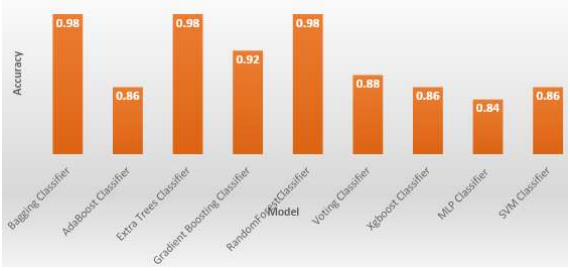

Figure 6: Training accuracy of the different proposed classifiers

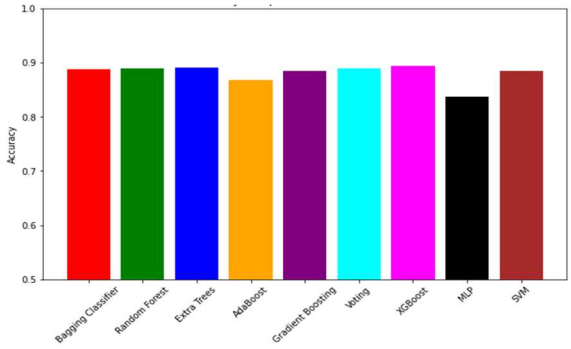

Figure 7: Testing Accuracy of the different proposed classifiers

The following confusion matrix provides a summary of prediction results on a classification problem. The number of correct and incorrect predictions is summarized with count values and broken down by each class. This visual representation, depicted in Figure 9, provides a breakdown of prediction results and highlights how our classification model encounters confusion during its predictions. By analyzing the matrix, we gain insights into the types of errors made by the classifier, enabling us to refine and improve its performance.

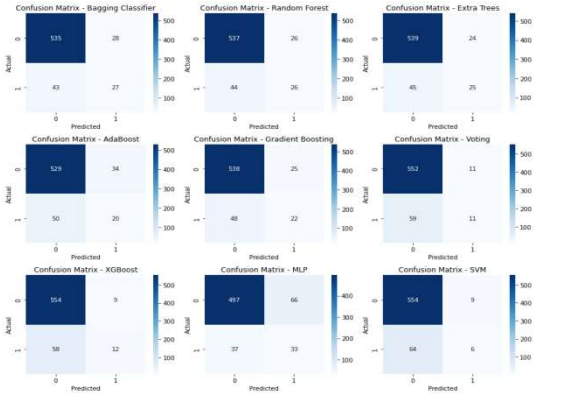

Figure 8: Confusion matrix visualization of different machine learning models

In machine learning, evaluating model performance is essential to understand how effectively a model performs its designated task. The following tables present the performance evaluation of nine base classifiers, assessing metrics such as accuracy, precision, recall, f-score, and support. Notably, the XGBoost, Extra Trees, and Voting classifiers demonstrate strong performance, while the MLP classifier shows less favorable results.

Table 5: Experimental results of Base Classifier Models with various evaluation metrics

Base Classifier	Accuracy	Precision	Recall	F1-score	Support
Bagging	0.8878	0.77	0.35	0.45	89
Random Forest	0.8894	0.79	0.30	0.43	89
Extra Trees	0.8909	0.81	0.34	0.43	89
AdaBoost	0.8672	0.65	0.18	0.27	89
Gradient Boosting	0.8846	0.75	0.29	0.36	89
Voting	0.8894	0.79	0.17	0.26	89
XGBoost	0.8941	0.83	0.12	0.21	89
MLP	0.8372	0.55	00	00	89
SVM	0.8846	0.75	0.17	0.26	89

The evaluation results presented in Table 5 provide a comprehensive assessment of various classification models, including Bagging, Random Forest, Extra Trees, AdaBoost, Gradient Boosting, Voting, XGBoost, MLP, and SVM, are presented. Each model's performance is assessed across multiple metrics: Accuracy measures overall correctness, with XGBoost achieving the highest accuracy of 0.8941, indicating it correctly classified approximately 89.41% of instances. Precision evaluates the model's ability to avoid false positives, with XGBoost exhibiting the highest precision of 0.83. Recall assesses the model's capability to capture all positive instances, where Bagging achieved the highest recall of 0.35, effectively capturing 35% of positive instances. The F1-score, reflecting a balance between precision and recall, shows XGBoost leading with a score of 0.21. Support: column indicates the number of instances in the test dataset that belong to each class. Overall, XGBoost emerges as the top performer, among the evaluated models based on accuracy, precision, recall, and F1-score, making it a promising choice for the classification task at hand.

Table 6: Accuracy comparison with existing model [23]

o.	xisting ML Model	uthor	ataset	ccuracy
	B	lbal, A.	ASA	2.65
	VM-FS	lumtaz, .	ASA	1.79
	B-FS	lumtaz, .	ASA	5.55
	F-FS	lumtaz, .	ASA	5.20
	VM-AdaBoost	lsaeedi, .	ASA	9.0
	B-PCA	etiner, M.	ASA	1
	VM-PCA	etiner, M.	ASA	3
	F-PCA	etiner, M.	ASA	3
	ur Proposed Ensemble lodel			
	xtra Trees		ASA	9.09
	daBoost		ASA	5.72
	radient Boosting		ASA	8.46
	oting		ASA	8.94
	GBoost		ASA	9.41
	andom Forest		ASA	8.94

The comparison Table 6 displays a significant improvement in accuracy with the proposed models compared to the existing machine-learning models. In the existing models, the highest accuracy was achieved by Naive Bayes with Feature Selection (NB-FS) at 85.55%, closely followed by Random Forest with Feature Selection (RF-FS) at 85.20%. Meanwhile, the Support Vector Machine with AdaBoost (SVM-AdaBoost) had the lowest accuracy, reaching 79.0%.

On the other hand, the proposed models demonstrate higher accuracy overall. Extra Trees leads with an accuracy of 0.8909, with XGBoost slightly ahead at 0.8941. Other models, like Gradient Boosting, Voting, and Random Forest, also maintain high accuracy, ranging from 0.8672 to 0.8894. This suggests that the proposed models, especially those utilizing ensemble techniques such as Extra Trees, XGBoost, and Gradient Boosting, deliver superior performance in terms of accuracy.

These results highlight the effectiveness of ensemble-based methods in achieving higher

accuracy compared to traditional machine learning models. The substantial difference in accuracy indicates that ensemble techniques, along with sophisticated feature selection and boosting methods, offer a robust approach for achieving better predictive performance.

Comparing our results with current state-of-the-art solutions in the literature, we observe that our ensemble methods, particularly XGBoost, demonstrate competitive performance in software defect classification. Existing studies have shown the efficacy of various machine learning models, but our approach highlights the superior accuracy and precision achievable through ensemble techniques. For instance, traditional single classifiers like SVM and MLP, while effective, often fall short in recall and overall balance of performance metrics compared to our ensemble methods.

Critiquing our work against our initial goals, we aimed to enhance the accuracy and robustness of software reliability prediction models. Our outcomes align with these goals, as evidenced by the high performance metrics achieved by the ensemble models. However, our analysis was constrained by a limited dataset, which impacted the recall and F1-score, particularly for models like Bagging.

5. CONCLUSION AND FUTURE WORKS

In this study, we employed ensemble learning techniques alongside hyperparameter selection algorithms to enhance model performance for software reliability prediction. Our evaluation encompassed various well-established classification algorithms, including Bagging, Random Forest, Extra Trees, AdaBoost, Gradient Boosting, Voting, XGBoost, MLP, and SVM, each assessed across multiple performance metrics on the CM1 dataset. Notably, XGBoost emerged as the top performer, achieving the highest accuracy of 89.41% and precision of 83%. Bagging exhibited the highest recall at 35%, while XGBoost led in F1-score with 21%. These findings underscore the effectiveness of ensemble learning methodologies in enhancing software reliability prediction accuracy. By leveraging a diverse set of classification models within the ensemble framework, we achieved robust performance across different evaluation metrics.

The comparison highlights the potential of ensemble techniques, particularly XGBoost, in accurately classifying software defects and improving overall reliability prediction. Our study contributes to advancing the understanding of ensemble learning's efficacy in the context of software reliability prediction. However, our analysis was constrained by a limited dataset. Moving forward, we aim to expand the dataset size and explore various ensemble classifiers after implementing data balancing techniques. Balancing the dataset can enhance error measures and potentially yield improved results. Additionally, we plan to investigate alternative optimization approaches on larger datasets to further refine our models' performance. By addressing these limitations and conducting future research in these areas, we can enhance the robustness and applicability of our findings in software bug detection and contribute to advancing the field of machine learning in software engineering.

Competing Interests: The author declares no conflict of interest.

Funding Information: No funding is available

Data Availability Statement: The data is publicly available at https://github.com/sudhirkmohapatra/SRP-using-Enseble-Learning

Research Involving Human and /or Animals: No animal or human is involve in the experiment of this research.

Informed Consent: NA

REFERENCES:

[1] Pressman, R.S. (2001) Software Engineering: A Practitioner's Approach, 5th ed., McGraw Hill Publications, USA.
[2] Liu, M., Fang, S., Dong, H., & Xu, C. (2021). Review of digital twin about concepts, technologies, and industrial applications. Journal of manufacturing systems, 58, 346-361.
[3] Bhuiyan, S. A. R., Rahim, M. S., Chowdhury, A. E., & Hasan, M. H. (2018). A survey of software quality assurance and testing practices and challenges in bangladesh. International Journal of Computer Applications, 975, 8887.
[4] Standard glossary of software engineering terminology (STD-729-1991). ANSI/IEEE, 1991
[5] Lyu, M. R., & Nikora, A. (1992, July). CASRE-A computer-aided software reliability

estimation tool. In CASE'92 Proceedings (pp. 264-275).

[6] Ucar, A.; Karakose, M.; Kırımça, N. Artificial Intelligence for Predictive Maintenance Applications: Key Components, Trustworthiness, and Future Trends. Appl. Sci. 2024, 14, 898. https://doi.org/10.3390/app14020898

[7] Balaram, A., & Vasundra, S. (2022). A Review on Machine Learning Techniques to Predict the Reliability in Software Products. In Proceedings of the 2nd International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications: ICMISC 2021 (pp. 309-317). Springer Singapore.

[8] Habtemariam, G. M., Mohapatra, S. K., Seid, H. W., & Mishra, D. B. (2022). A Systematic Literature Review of Predicting Software Reliability Using Machine Learning Techniques. Optimization of Automated Software Testing Using Meta-Heuristic Techniques, 77-90.

[9] Rath S. K., M. Sahu, S. P. Das and S. K. Mohapatra, "Hybrid Software Reliability Prediction Model Using Feature Selection and Support Vector Classifier," 2022 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2022, pp. 1-4, doi: 10.1109/ESCI53509.2022.9758339

[10] R Wu, M., Lin, J., Shi, S., Ren, L., Wang, Z. (2020). Hybrid Optimization-Based GRU Neural Network for Software Reliability Prediction. https://doi.org/10.1007/978-981-15-7984-4_27

[11] Dong, X., Liang, Y., Miyamoto, S., & Yamaguchi, S. (2023). Ensemble learning based software defect prediction. Journal of Engineering Research, 11(4), 377–391. https://doi.org/10.1016/j.jer.2023.10.038

[12] Moeini, A., oveisi, shahrzad, farsi, mohammad ali, & Mirzaei, S. (2022). Software reliability prediction: A machine learning and approximation bayesian inference approach. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.4264288

[13] Rath, S.K., Sahu, M., Das, S.P., Pradhan, J. (2022). An Improved Software Reliability Prediction Model by Using Feature Selection and Extreme Learning Machine. https://doi.org/10.1007/978-3-031-11713-8_23

[14] S. Yadav and Balkishan, "Software reliability prediction by using Deep Learning Technique," International Journal of Advanced Computer Science and Applications, vol. 13, no. 3, 2022. doi:10.14569/ijacsa.2022.0130381

[15] G. Jabeen, P. Luo, and W. Afzal, "An improved software reliability prediction model by using high precision error iterative analysis method," Software Testing, Verification and Reliability, vol. 29, no. 6–7, Sep. 2019. doi:10.1002/stvr.1710

[16] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep Learning based software defect prediction," Neurocomputing, vol. 385, pp. 100–110, Apr. 2020. doi:10.1016/j.neucom.2019.11.067

[17] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," Cluster Computing, vol. 22, no. S4, pp. 9847–9863, Jan. 2018. doi:10.1007/s10586-018-1696-z

[18] Y. S. Kim, K. Y. Song, and I. H. Chang, "Prediction and comparative analysis of software reliability models based on NHPP and deep learning," Applied Sciences, vol. 13, no. 11, p. 6730, May 2023. doi:10.3390/app13116730

[19] Habtemariam, G.M., Mohapatra, S.K., Seid, H.W. (2022). Prediction of Software Reliability Using Particle Swarm Optimization. In: Panda, M., et al. Innovations in Intelligent Computing and Communication. ICIICC 2022. Communications in Computer and Information Science, vol 1737. Springer, Cham. https://doi.org/10.1007/978-3-031-23233-6_11

[20] Mohapatra, S.K., Mishra, A.K. & Prasad, S. Intelligent Local Search for Test Case Minimization. J. Inst. Eng. India Ser. B 101, 585–595 (2020). https://doi.org/10.1007/s40031-020-00480-7

[21] Deneke, Aliazar, Beakal Gizachew Assefa, and Sudhir Kumar Mohapatra. "Test suite minimization using particle swarm optimization." Materials Today: Proceedings 60 (2022): 229-233.

[22] Promise Software Engineering Repository. Available online: http://promise.site.uottawa.ca/SERepository/datasets-page.htm (accessed on 20 October 2021).

[23] Khalid, A., Badshah, G., Ayub, N., Shiraz, M., & Ghouse, M. (2023). Software Defect Prediction Analysis Using Machine Learning Techniques. Sustainability, 15(6), 5517.