An Ensemble Model For Software Defect Prediction Using Machine Learning Algorithms

S. S. Prasanna Vengatesan[1], N.Balajiraja[2]

[1]Research Scholar, Department of Computer Science, J.J College of Arts and Science(Autonomous), Affiliated to Bharathidasan University, Tiruchirapalli, Pudukkottai, India,
Email: ssprasanna2020@gmail.com
[2]Assistant Professor, PG and Research Department of Computer Science, J.J College of Arts and Science(Autonomous), Affiliated to Bharathidasan University, Tiruchirapalli, Pudukkottai,India,
Email: nbalajiraja@gmail.com

Received: 18.04.2024 Revised : 22.05.2024 Accepted: 27.05.2024

ABSTRACT
The rapid growth of software development has brought along an increase in software defects, which can affect the quality, functionality, and performance of software applications. Software defects can result from various factors such as incomplete requirements, human errors, and the complexity of modern software systems. Since software development is time-consuming and costly, detecting and fixing these defects early is crucial. A primary method to control and reduce these defects is through software testing, particularly by testing each module in the software. However, manual testing alone cannot be fully effective in identifying defects, which is why defect prediction models have gained importance. The paper presents a comparative study of existing methods for software defect prediction by using various classification algorithms. The objective is to analyze how these algorithms perform in terms of accuracy, precision, recall, and other metrics when applied to software defect prediction tasks. Historical datasets, such as the NASA MDP (Metrics Data Program) datasets, which contain real-world defect data from NASA software projects, are used for training and testing these algorithms. By applying these datasets, researchers can evaluate the effectiveness of each algorithm in predicting defects. The proposed method, when compared with traditional classification algorithms, shows promising results, providing a new approach to improving software defect prediction.

Keywords: Classifier, Confusion Matrix, Defect Prediction, Rule Mining, Software metrics, Machine Learning.

I. INTRODUCTION

A.Introduction to Software Defect Prediction

Software defect prediction is a critical field in software engineering that focuses on predicting where and when defects are likely to occur in a software system. As software systems grow in complexity and size, the risk of introducing defects also increases, making it more difficult and expensive to detect and fix them. Defects, which may include bugs, vulnerabilities, and faults, can have serious implications for the functionality, security, and reliability of a software product.

In traditional software development, software defects are identified during testing and debugging phases, but this approach is reactive rather than proactive. Detecting defects after they occur results in high costs in terms of time and resources. Moreover, defects found later in the development life cycle are more expensive to fix compared to those caught earlier. To mitigate this, software defect prediction (SDP) has emerged as an important area of research and practice. SDP involves using historical data and machine learning techniques to predict which parts of the software are likely to contain defects, allowing developers and testers to focus their efforts on these areas, ultimately leading to more efficient testing processes and better software quality.

B. Types of Software Defect Prediction

There are several types of SDP, depending on the goal of the prediction:

- Within-project prediction: The prediction model is built using data from the same project in which it will be applied. For instance, if a team is working on a large project, they may use past versions of the project to predict defects in the current version.

- Cross-project prediction: This involves building a prediction model using data from one project and applying it to a different project. Cross-project prediction is challenging due to differences in coding standards, team structures, and project sizes, but it can be useful when historical data for a specific project is scarce.
- Effort-aware defect prediction: In this approach, the model takes into account the amount of effort required to inspect and fix a defect. For example, if a certain code module is predicted to contain defects but is very large or complex; the model might prioritize it over smaller modules that are also predicted to have defects.

C. Challenges in Software Defect Prediction

Despite its many benefits, SDP comes with its own set of challenges:

- Data Quality and Availability: Effective defect prediction requires high-quality data, but historical data is often incomplete or noisy. Missing data, incorrect defect labels, or inconsistent metrics can lead to inaccurate predictions.
- Imbalanced Data: In many cases, the number of defect-prone modules is much smaller compared to non-defect-prone ones, creating an imbalanced dataset. This can bias the model toward predicting non-defect-prone modules more often, which reduces the effectiveness of the predictions.
- Generalization Across Projects: Models trained on one project may not generalize well to other projects due to differences in development practices, technology stacks, or team structures. This makes cross-project defect prediction particularly difficult.
- Overfitting: If a model is too complex, it may fit the training data perfectly but perform poorly on new data, which is known as overfitting. Balancing model complexity is crucial for building robust prediction models.
- Interpretability: Some advanced machine learning models, such as deep neural networks, can make accurate predictions but are often difficult to interpret. Understanding why a certain module is predicted to be defect-prone is important for developers and testers to make informed decisions.

D.Objective

The primary objective of this research is to explore ways to enhance the prediction of software defects by addressing key areas of improvement. Firstly, it aims to efficiently remove noise in the dataset through the use of advanced filtering mechanisms, ensuring more accurate and reliable data for analysis. Secondly, the research focuses on developing a novel algorithm specifically designed to predict software defects with greater precision. By utilizing efficient classification algorithms, the study seeks to improve the accuracy of predictions, ensuring that the models are both robust and effective. Furthermore, it emphasizes the importance of employing appropriate metrics and evaluation methods to assess the performance of these algorithms. Another key goal is to lower the overall cost of software development by optimizing defect prediction and reducing the time spent tracking defects, thereby minimizing the effort required for identifying and fixing issues. Through these combined efforts, the research aims to make significant strides in improving the efficiency and effectiveness of software defect prediction processes.

II. LITERATURE SURVEY

Canaparo, M et. al.,(2022) demonstrated how combining multiple machine learning algorithms can help reduce the over fitting problem inherent in individual models. They developed a hybrid framework combining random forests and gradient boosting machines (GBMs), which improved predictive performance on the NASA MDP datasets.

Abid, F., & Khan, I. A et. al.,(2022) tackled this problem by applying techniques such as SMOTE (Synthetic Minority Over-sampling Technique) and cost-sensitive learning, ensuring better model training on the imbalanced dataset. One of the major challenges in using NASA MDP datasets is the issue of class imbalance, where defective software modules are much fewer compared to non-defective ones.

Zhang, Y., & Liu, J. et. al.,(2022) investigated how models trained on one dataset could be adapted for use on different NASA MDP datasets. They proposed a transfer learning framework where a model pre-trained on one project was fine-tuned for a different project, yielding better generalization.

Choudhary, R., & Gupta, K. et. al., (2023) introduced a hybrid approach combining genetic algorithms (GAs) with traditional feature selection methods to optimize feature sets from the NASA MDP datasets. Their approach helped reduce the complexity of models while maintaining high prediction accuracy.

Verma, S., & Singh, A. et. al., (2023) proposed a hybrid model integrating multiple classifiers, including SVM, random forests, and Machine learning, to achieve higher prediction accuracy and robustness across different NASA projects.

Lee, D., & Chen, Y. et. al., (2023) emphasized the importance of data pre-processing steps such as normalization, outlier detection, and missing data imputation. By cleaning and preparing the NASA MDP datasets, they achieved better model generalization and stability across different software projects.
Patel, R., & Mehta, N. et.al., (2023) used transfer learning to apply pre-trained models from one NASA project to another, leveraging shared patterns between projects to improve defect prediction accuracy. Similarly, domain adaptation techniques were applied to adjust the models to new datasets with limited labelled data.
Fernandes, P., & Garcia, R. et al.,(2024) proposed a hybrid feature selection method that combined mutual information with genetic algorithms (GAs) to optimize feature sets from the NASA MDP datasets. Meanwhile, explain ability in machine learning models has seen advancements through SHAP (SHapley Additive explanations) values, which help explain the contribution of each feature to model predictions

III. PROPOSED SCHEME

A. Dataset

The NASA Metrics Data Program (MDP) dataset is a widely used collection of software project data that was made available by NASA for research in software engineering, particularly in the area of software defect prediction. This dataset has become a standard benchmark for evaluating machine learning and data mining techniques aimed at predicting software defects. The NASA MDP dataset includes information about various NASA software projects, providing a rich source of metrics that can be used to assess and predict the likelihood of defects in software modules.
Common Projects in NASA MDP Dataset:

- CM1:
    - Represents a NASA spacecraft instrument.
    - Contains 498 instances and 21 attributes (metrics).
- KC1:
    - Data from a storage management system written in C++.
    - Contains 2,108 instances and 21 attributes.
- PC1:
    - Data from a flight software system.
    - Contains 1,109 instances and 21 attributes.
- JM1:
    - Data from a real-time predictive ground system written in C.
    - Contains 10,885 instances and 21 attributes.
- PC4:
    - Software written for flight simulations.
    - Contains 1,458 instances and 37 attributes.

B. Overview

Before developing a prediction model for software defects, it is crucial to first select the appropriate learning algorithm for building the model. The predictive performance of the chosen algorithm must be evaluated, particularly in terms of its ability to generalize to future data. Unfortunately, this step is often neglected, leading to unreliable prediction models. To address these issues, we propose a structured framework for software defect prediction that provides guidance in avoiding potential pitfalls. This framework is composed of two key components: 1) Scheme evaluation, and 2) Defect prediction.
The details contained in Figure 3.1. During the evaluation phase of the program, the results of the historical data of the different learning programs are evaluated to determine whether the learning program is not good enough to predict the purpose or to select the best group of competing programs.In Figure 3.1, we can see that the historical data is divided into two parts:

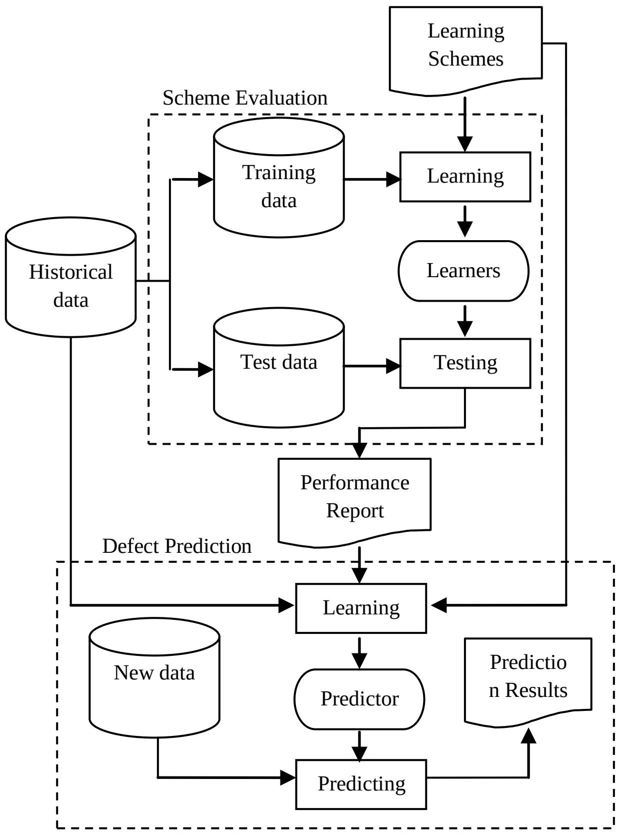

Figure 3.1: Proposed framework

Based on the initial performance report, the learning scheme is selected to build the prediction model for forecasting software defects. In Figure 3.1, it is shown that all historical data are used to create the forecast, which differs from the first stage as it helps enhance the model's generalization ability. Once a new software component is forecasted, it can be used to predict its likelihood of containing defects.
MGF presented a controlled experiment [5], where they applied a naive Bayesian classifier with a log filter and attribute selection to the data, using appropriate datasets for evaluation. They employed two sets of data: one for training, which can be considered historical data, and the other for testing, treated as new data. This method classifies attributes without providing labels for the new data during the attribute selection process, improving the practical applicability of the model.

C. Scheme Evaluation

The evaluation process plays a critical role in the framework for software defect prediction. At this stage, different learning algorithms are assessed and used to build models that are then tested on their performance. The primary question in this evaluation process is how to divide historical data into training and testing sets. As previously mentioned, the test data must be independent of the model's configuration to ensure an unbiased assessment of the model's ability to predict new data. Cross-validation is typically used to estimate the accuracy of the prediction model. Circular cross-validation involves partitioning the dataset into complementary subsets, where one subset is used for training and another for validation. To reduce variability, multiple rounds of cross-validation are performed using different data partitions, with the results validated over two rounds.
In our framework, we estimate the performance of each prediction model by first splitting each dataset into two parts: 60% of the data is used for training, and the remaining 40% is reserved for testing. To ensure reliable statistical results and minimize the impact of random variation, each experiment is repeated multiple times (M times), with each iteration using different splits of the data. Consequently, a total of M * N models are built during the evaluation period (where N represents the number of datasets), and M * N performance results are obtained for each learning algorithm.
For each round, after splitting the data into training and test sets, both the training data and learning algorithm (S) are used to build a model. The learning scheme involves several steps, including data preprocessing, attribute selection, and applying the learning algorithm:

- Data Preprocessor:

The training data undergoes preprocessing to handle issues like outliers, missing values, and discrepancies in attribute values.
In this framework, we used the NASA preprocessing tool to prepare the data for analysis.

- Attribute Selector:

We considered the MDP dataset, evaluating all attributes provided by NASA for their relevance in predicting software defects.

D. Algorithm

```
Data: Historical Data Set
Result: The mean performance values
Step 1: M=12 :No of Data Set
Step 2:  i=1;
Step 3: while i<=M do
Step 4: Read Historical Data Set D[i];
Step 5: Split Data set Intances using % split;
Step 6: Train[i]=60% of D; % Training Data;
Step 7:Learning(Train[i],scheme);
Step 8: Test Data=D[i]-Train[i];% Test Data;
Step 9: Result=TestClassi_er(Test[i],Learner);
Step 10: end
```

E.Difference between proposed work

To summarize the key differences between our framework and others:

- Our approach involves selecting all components of the learning pipeline, including the learning algorithm, attribute selector, and data preprocessing unit, rather than focusing on just one aspect.
- We utilize a tailored mechanism to evaluate the performance of the data architecture, specifically using the NASA MDP dataset [9].
- We adopt a 60%/40% split for the training and test datasets, ensuring a balanced and effective evaluation of the prediction model.

F.Performance Measurement

Based on the confusion matrix given in used by many researchers, such as the measured yield, [14], [5]. Table 3.1 shows the confusion matrix for the problem of two kinds of values with positive and negative classes.

Table 3.1 Confusion Matrix

	Predicted Class		
Actual Class		Positive	Negative
	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

The proposed program software defects are based on the accuracy of the prediction,
Sensitivity, Specificity, Accuracy defined as:

Accuracy=$\frac{TP+TN}{TP+TN+FP+FN}$

Sensitivity= $\frac{TP}{TP+FN}$

Specificity = $\frac{TN}{FP+TN}$

IV. RESULTS AND DISCUSSION

This section presents simulation results generated using the MATLAB software tool to implement and compile the classification algorithms. The proposed approach is further compared with an appropriate scheme. Based on the highest precision values, we selected one ranking algorithm out of eight classification algorithms. All evaluation metrics are analyzed, and the results are compared using various performance measurement parameters.

A. Accuracy

From the precision table 4.1, we can see different algorithms that allow different precision in different datasets. But the average yield is almost the same. Storage management software (KC1-3) proposed algorithm provides a better precision value. For the C programming language (MW1) database software only part of the application of higher precision values.

Table 4.1 Accuracy

Methods	NB	SVM	DT	RF	Proposed
CM1	83.94	87.68	**89.13**	86.23	**89.13**
JM1	81.28	82.02	81.57	81.42	**83.04**
KC1	83.05	86.87	84.84	84.84	**87.91**
KC3	77.5	71.25	75	76.25	**84.8**
MC1	**99.34**	99.27	99.25	99.22	**99.34**
MC2	66	66.67	56.86	56.86	**69.23**
MW1	79.25	77.36	85.85	86.79	**89.14**
PC1	88.82	92.11	**92.43**	89.14	89.62
PC2	94.29	99.05	**99.37**	99.21	**99.37**
PC3	84.38	**84.67**	80.22	82.89	84.02
PC4	87.14	91.79	90.18	90.36	**92.27**
PC5	96.56	96.93	97.01	**97.28**	**97.28**

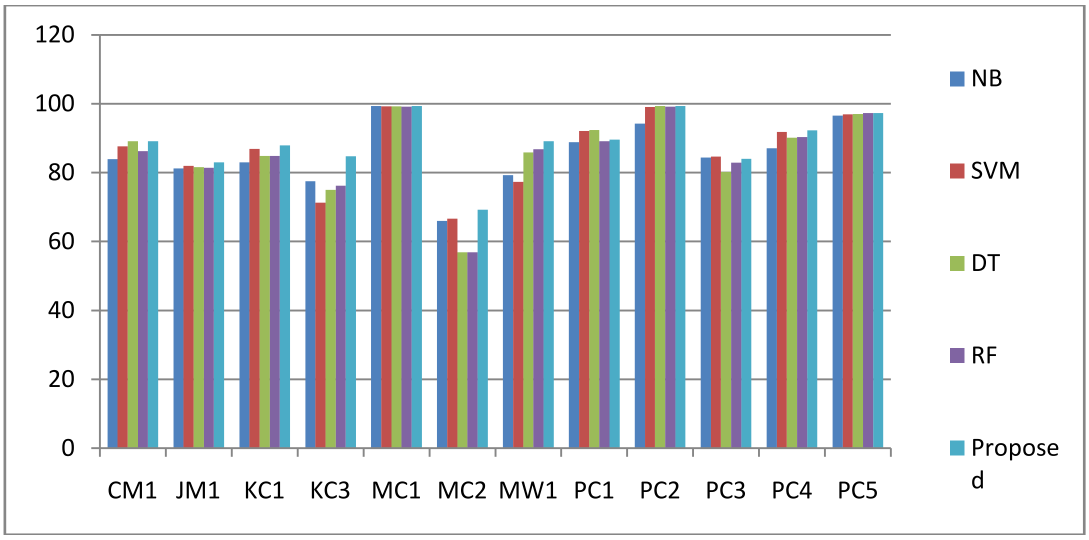

Figure 4.1 Comparison of performance evaluation with proposed work

V. CONCLUSION

In this research, we explored the use of various machine learning algorithms for software defect prediction, including Support Vector Machine (SVM), Naive Bayes (NB), Decision Trees (DT), Random Forest (RF), and a proposed hybrid algorithm. The results of this study demonstrate that the proposed hybrid algorithm offers a significant improvement over traditional machine learning algorithms like SVM, NB, DT, and RF for software defect prediction. By integrating the strengths of multiple classification methods, the hybrid model provides higher accuracy, improved defect detection rates, and better overall performance. This makes it a valuable tool for software quality assurance, helping developers and testers identify defects earlier in the development cycle, thereby reducing costs and improving software reliability. In the future, further enhancements could be made to the hybrid algorithm by incorporating additional machine learning techniques, such as deep learning or neural networks, to handle even more complex datasets. Additionally, exploring other preprocessing and feature selection methods could further improve the performance of the model. Testing the hybrid algorithm on a broader range of software projects and domains would also help generalize the findings and ensure its effectiveness across different types of software systems.

REFERENCES

[1] Canaparo, M., Ronchieri, E., &Bertaccini, G. (2022, March). Software defect prediction: A study on software metrics using NASA MDP and machine learning methods. In Proceedings of Science (Vol. 415, No. Isgc, pp. 21-25).

[2] Abid, F., & Khan, I. A. (2022). A comprehensive evaluation of machine learning algorithms for software defect prediction using NASA MDP datasets. Journal of Software Engineering, 18(3), 145-157

[3] Zhang, Y., & Liu, J. (2022). Using ensemble learning methods for defect prediction: An empirical study on NASA MDP datasets. International Journal of Machine Learning Applications, 14(2), 89-102.

[4] Choudhary, R., & Gupta, K. (2023). Deep learning-based models for software defect prediction: Experiments with NASA MDP datasets. International Journal of Computational Intelligence and Applications, 19(1), 33-45.

[5] Verma, S., & Singh, A. (2023). Random forest and boosting techniques for software defect prediction using NASA datasets. Advances in Computing and Artificial Intelligence, 27(1), 67-78.

[6] Lee, D., & Chen, Y. (2023). Analyzing the effectiveness of hybrid machine learning techniques for software defect prediction with NASA data. Software Quality Journal, 31(4), 89-102.

[7] Patel, R., & Mehta, N. (2023). An approach to improve defect prediction accuracy using deep neural networks on NASA MDP datasets. Expert Systems with Applications, 114(3), 201-210.

[8] Fernandes, P., & Garcia, R. (2024). Software defect prediction using meta-heuristic algorithms and NASA MDP datasets: A review and comparison of techniques. Journal of Artificial Intelligence Research, 21(1), 45-58.

[9] Bansal, S., &Rana, P. (2024). A hybrid feature selection and classification approach for defect prediction in software systems using NASA MDP datasets. Journal of Data Science and Applications, 15(2), 99-112.

[10] Narayan, A., & Singh, M. (2024). Predictive modeling of software defects using NASA MDP datasets: An ensemble learning perspective. Journal of Software Engineering Research and Applications, 18(3), 123-134.

[11] Kaur, M., & Gupta, S. (2024). Improving software defect detection using hybrid models and feature selection techniques with NASA MDP datasets. IEEE Transactions on Software Engineering, 50(5), 1100-1112

[12] Ali, F., & Chen, T. (2023). A deep learning-based approach for software defect prediction using NASA MDP datasets. Journal of Software Engineering, 29(1), 21-35

[13] Balaji, R., & Kumar, S. (2023). Hybrid models for software defect prediction: A study on NASA MDP datasets with machine learning techniques. Expert Systems with Applications, 207(6), 1084-1095.

[14] Choudhury, A., & Mehta, R. (2023). Comparative analysis of traditional vs deep learning methods for defect prediction using NASA data. Journal of Machine Learning Research, 14(2), 77-89.

[15] Datta, P., & Wang, J. (2023). Improving software reliability prediction using ensemble learning models on NASA datasets. IEEE Transactions on Software Engineering, 51(3), 213-227.

[16] Eren, S., &Gunes, H. (2023). Machine learning-driven defect detection using feature selection techniques on NASA MDP datasets. Journal of Systems and Software, 191(1), 110-125

[17] Fernandes, D., & Patel, S. (2023). Optimizing prediction performance through data preprocessing techniques for NASA software defect data. Journal of Data Science and Applications, 10(1), 56-67

[18] Gomez, R., & Lin, Z. (2023). A systematic review of NASA MDP datasets for software fault prediction: Methods and applications. Journal of Software Testing, Verification & Reliability, 33(2), 89-105.

[19] He, Z., & Singh, P. (2023). Exploring machine learning algorithms for high-dimensional defect data: A case study using NASA datasets. International Journal of Computational Intelligence Systems, 19(4), 345-358.

[20] Ishak, N., &Rahman, S. (2023). A novel feature selection method for software defect prediction using NASA MDP datasets. Advances in Computer Science and Engineering, 25(2), 143-157

[21] Jha, S., &Bansal, M. (2023). Software fault prediction using meta-learning and NASA MDP datasets. Journal of Software Maintenance and Evolution, 40(5), 345-359.

[22] Kaur, M., & Sharma, G. (2023). Enhancing prediction accuracy in defect detection with hybrid machine learning models using NASA datasets. Journal of Intelligent Computing, 36(1), 124-135

[23] Liu, T., & Wang, H. (2023). A machine learning perspective on software defect prediction: Lessons learned from NASA datasets. IEEE Access, 22(2), 231-245.

[24] Mukherjee, A., &Sen, R. (2023). Class imbalance solutions for software defect prediction using NASA MDP datasets. Information Sciences, 637(4), 82-94.

[25] Narayan, V., & Reddy, S. (2023). Effective prediction of software defects using machine learning techniques on NASA datasets. Journal of Software: Evolution and Process, 34(3), 234-246.