www.ijprems.com
editor@ijprems.com

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)
(Int Peer Reviewed Journal)
Vol. 05, Issue 03, March 2025, pp : 796-801

e-ISSN : 2583-1062
Impact Factor : 7.001

INTELLIGENT SOFTWARE DEFECT PREDICTION USING MACHINE LEARNING CLASSIFICATION ALGORITHMS

Manuj Joshi[1]

[1]Assistant Professor, FCI, Sir Padampat Singhania University, Udaipur, India.

manujjoshi@gmail.com

DOI: https://www.doi.org/10.58257/IJPREMS39007

ABSTRACT

Software defect prediction (SDP) is a critical task in software engineering, aimed at improving software quality by identifying defective modules before deployment. Conventional defect detection techniques are frequently laborious and prone to human error. Automated defect prediction models have drawn a lot of attention as a result of improvements in Machine Learning (ML) classification algorithms. These models increase the effectiveness of defect management by using software metrics and historical defect data to categorise software modules as either defective or non-defective.In this study, we compare different classification models, such as Lazy-IBK, Lazy-K Star, SMO, Decision Stump, J48, and Naïve Bayes, and measure the effect of ML classification algorithms on software defect prediction. Key performance metrics like accuracy, Kappa statistic, mean absolute error (MAE), root mean squared error (RMSE), relative absolute error (RAE), and root relative squared error (RRSE) are used in the study to assess these algorithms. The study develops and tests two hypotheses in order to validate the results: ($H_0$1) that ML classification algorithms have no discernible effect on defect prediction, and ($H_0$2) that there are no discernible differences between different classification models. The experimental findings show differences in error rates and predictive accuracy, emphasising the superiority of some models over others in the classification of software defects. Insights into the top-performing machine learning classifiers for defect prediction are provided by the research findings, which benefit software engineering by empowering developers to choose effective models for defect detection. To further increase prediction accuracy and dependability, future studies can investigate the combination of deep learning methods and hybrid models.

Keywords: Machine Learning, Software Defect Prediction, Deep Learning

1. INTRODUCTION

The Software defect prediction (SDP) plays a crucial role in enhancing software quality and reducing maintenance costs by identifying potential defective modules early in the software development lifecycle. Manual code reviews, rule-based methods, and static analysis are the mainstays of traditional defect prediction approaches, which are frequently laborious, prone to errors, and ineffective for large-scale software systems. The increasing complexity of modern software demands intelligent and automated methods to detect defects effectively.As machine learning (ML) has advanced, classification algorithms have become increasingly effective tools for anticipating software flaws. ML-based models leverage historical defect data and software metrics to learn patterns associated with defective and non-defective code segments. New software components can then be categorised by these models according to how likely they are to have flaws. In software defect prediction, well-known classification algorithms like Decision Trees, Random Forest, Support Vector Machines (SVM), Naïve Bayes, K-Nearest Neighbours (KNN), and Deep Learning models have shown encouraging results.

Several variables, such as feature selection, model optimisation, and training data quality, affect how well defect prediction models work. Enhancing prediction accuracy requires the use of feature engineering techniques, such as static code attributes (e.g., lines of code, cyclomatic complexity, code churn) and process metrics (e.g., historical defect density, change frequency). In order to improve the generalisability and resilience of defect prediction models, hybrid and ensemble learning approaches have also been investigated. This study aims to explore and compare different machine learning classification algorithms for intelligent software defect prediction. This study offers insights into the best methods for defect detection by assessing their performance using open-source repositories and standard software defect datasets like NASA PROMISE. By facilitating proactive defect management techniques, the findings aid in the creation of software systems that are more dependable and effective.

2. LITERATURE REVIEW

Albattah, W., & Alzahrani, M. (2024) Software bug prediction helps identify defects early in the development process, reducing costs and improving reliability. Using a dataset of 60 software metrics from five public bug datasets, this study assesses eight machine learning and deep learning models. Cohesion, coupling, complexity, documentation inheritance, and size metrics are among the important quality indicators that are used for classification. Accuracy, weighted F1 score, binary F1 score, and macro F1 score are used to gauge performance. With an accuracy of 0.87, the results demonstrate

that the LSTM deep learning model performs better than the others. Software maintainability research trends and challenges were summarised in a systematic literature review on the topic by Malhotra and Chug (2016). The study identifies the primary determinants of maintainability, such as software metrics, design patterns, and code complexity. The authors talk about different statistical and machine learning models that are used to predict maintainability and propose that new methods like search-based optimisation and deep learning could improve maintainability evaluation. This study offers a solid starting point for future research on automated software quality assessment. The issue of self-admitted technical debt (SATD) in open-source software projects was investigated by Huang et al. (2018). The authors created models that automatically detect situations in which developers acknowledge technical debt in code comments by utilising text mining techniques. According to their findings, SATD can have a major effect on the maintainability and quality of software. Additionally, by identifying latent software issues before they become more serious, natural language processing (NLP) techniques can improve software quality prediction models.

A data-efficient learning method for forecasting software performance in configurable systems was presented by Guo et al. (2018). By introducing methods that reduce the need for comprehensive performance measurements, the authors increased the efficiency of software optimisation. They applied machine learning models to predict system behavior with limited training data, emphasizing the importance of feature selection and sampling strategies. In addition to offering insights into creating more flexible machine learning models for software defect prediction, the study advances the field of performance-aware software engineering. A thorough overview of search-based model-driven engineering (SBMDE), which incorporates search-based optimisation methods into software modelling and development, is provided in this paper. The authors examine several metaheuristic algorithms for software model optimisation, including simulated annealing and genetic algorithms. In addition to highlighting how artificial intelligence (AI) can improve software design choices, the paper makes the case that machine learning could improve software defect prediction and maintainability even more.

The use of machine learning algorithms to forecast software module fault-proneness was investigated by Gondra (2008). The study evaluates the efficacy of several classification models, such as support vector machines (SVMs), decision trees, and neural networks, in identifying software flaws. The findings show that feature selection and ensemble learning strategies increase the accuracy of fault prediction. This research is particularly relevant for developing automated defect prediction systems.

A neural attention model was presented by Iyer et al. (2016) in order to automatically produce source code summaries. To produce human-readable explanations of code functionality, the authors used deep learning techniques, specifically recurrent neural networks (RNNs) with attention mechanisms. By strengthening code comprehension and review procedures, AI-based techniques can increase software maintainability and defect prediction, according to this research, which also helps automate software documentation.

In order to forecast software maintainability in object-oriented applications, Mishra and Sharma (2015) investigated fuzzy systems based on adaptive networks. Their method creates a clear, understandable model for software quality evaluation by fusing machine learning and fuzzy logic. In assessing software maintainability, the study highlights the significance of object-oriented metrics like inheritance depth and coupling. The findings imply that hybrid machine learning techniques increase the accuracy of maintainability predictions.

Ahmed and Al-Jamimi (2013) presented a fuzzy-based machine learning model for predicting software maintainability. To create a prediction system that is easier to understand and interpret, the study combines fuzzy logic with decision trees and neural networks. The authors contend that fuzzy-based methods are better suited for predicting software maintainability because traditional statistical models are less adaptable when dealing with ambiguous and imprecise data. This study offers important insights into fuzzy logic applications in predictive modelling, despite its primary focus on the machining process. The authors talk about how machine learning and fuzzy logic can be combined to improve software quality and defect prediction. According to the study, fuzzy systems are a promising method for software engineering applications because they can effectively handle uncertainty and imprecise software metrics.

A multi-objective optimisation model was put forth by Chhabra (2017) in an effort to enhance object-oriented package structure. To improve software modularity and maintainability, the study makes use of optimisation algorithms and weighted class connections. According to the study's findings, software architecture can be optimised through search-based and machine learning approaches, which lower maintenance costs and defect rates. Three empirical studies on the use of ensemble learning techniques to predict software maintainability were carried out by Elish et al. (2015). When compared to conventional machine learning models, the study finds that bagging, boosting, and stacking ensembles greatly increase prediction accuracy and robustness. The study backs up the claim that software defect prediction models can be improved through ensemble learning.

www.ijprems.com
editor@ijprems.com
INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)
(Int Peer Reviewed Journal)
Vol. 05, Issue 03, March 2025, pp : 796-801
e-ISSN : 2583-1062
Impact Factor : 7.001

3. RESEARCH METHODOLOGY

In software defect prediction (SDP), the AR1 dataset is a generally used benchmark dataset for assessing the performance of machine learning classification algorithms. Software metrics and defect labels make up it and enable the identification of trends separating faulty from non-defective software modules. The dataset includes several attributes taken from process metrics and static code elements taken from software source code. These characteristics are fundamental markers of software quality and help to train predictive models to label software modules as either non-defective or defective.

Structural characteristics of the code are measured in the dataset by means of code complexity metrics including Lines of Code (LOC), Cyclomatic Complexity (CC), and Halstead Metrics). Process metrics like change count, defect density, and historical modification frequency—which help to show how software components have evolved over time—also are included. Target variable is the defect label—binary classification: defective or non-defective. Usually employing 10-fold cross-valuation, the dataset guarantees a fair assessment of several machine learning classifiers. This allows a strong comparison of models depending on performance criteria including True Positive (TP), False Positive (FP) rate, Precision, Recall, F-Measure, and Area Under the Curve (AUC).

Objectives:

1. To measure the impact of Machine Learning Classification Algorithms on software defect prediction.
2. Comparative analysis of various classification machine learning algorithms used for software defect prediction.

Hypotheses:

Based on the above objectives following hypotheses was being framed:

$H_0$1: There is no significant impact of Machine Learning Classification Algorithms on software defect prediction.

H_a1: There is significant impact of Machine Learning Classification Algorithms on software defect prediction.

$H_0$2: There is no significant difference between various classification machine learning algorithms used for software defect prediction.

H_a2: There is significant difference between various classification machine learning algorithms used for software defect prediction.

4. RESULTS AND DISCUSSION

4.1 Comparative Analysis of Different Machine Learning:

The Table 4.1 evaluates several machine learning classifiers for software defect prediction using 10-fold cross-valuation on the AR1 dataset. Multiple performance criteria—Kappa Statistic, Accuracy, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Relative Absolute Error (RAE), Root Relative Squared Error (RRSE)—are used to evaluate the classifiers. The chosen models show differences in classification performance, according the results.

Table 4.1: Performance Evaluation of Different Classifiers on the AR1 Dataset

Using 10-Fold Cross-Validation (Kappa Statistic, Accuracy, MAE, RMSE, RAE, RRSE)

Classifier	Kappa Statistic	Correctly Classified Instances (Accuracy)	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error (%)	Root Relative Squared Error (%)
Lazy-IBK	0.3042	110 (90.90 %)	0.0964	0.29	66.76	113.74
Lazy-K Star	0.0293	104 (85.95%)	0.1421	0.3681	98.46	140.08
SMO	-0.0151	111 (91.73%)	0.0826	0.2875	57.25	109.40
Decision Stump	-0.0278	110 (90.91%)	0.1444	0.2995	100.0	113.97
J48	0.0785	109 (90.08%)	0.127	0.2997	87.98	114.05
Naïve Bayes	0.2331	103 (85.12%)	0.1518	0.3734	105.15	142.09

With the highest accuracy (91.73%), SMO followed closely by Lazy-IBK (90.90%), and Decision Stump (90.91%), so indicating their great predictive power among the classifiers. Naïve Bayes (85.12%) and Lazy-K Star (85.95%) had the lowest accuracy, therefore indicating less performance in identifying software flaws in classifying. Indicating modest agreement with the true classification labels, the Kappa Statistic—which gauges agreement beyond chance—was highest for Lazy-IBK (0.3042) and Naïve Bayes (0.2331). With SMO having the lowest MAE (0.0826) and Decision Stump having the highest RAE (100.0%), the error metrics (MAE, RMSE, RAE, RRSE) underline even more

performance variances in their predictions. While Decision Stump and Naïve Bayes show higher error rates and are therefore less appropriate for exact defect identification, the results imply that SMO and Lazy-IBK are the most reliable classifiers for defect prediction.

Table 4.2: Performance Evaluation of Different Classifiers on the AR1 Dataset

Using 10-Fold Cross-Validation (TP, FP Rate, Precision, F-Measure, Recall, and AUC)

Classifier	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area (AUC)
Lazy-IBK	0.909	0.62	0.904	0.909	0.907	0.776
Lazy-K Star	0.86	0.829	0.866	0.86	0.863	0.729
SMO	0.917	0.926	0.856	0.917	0.886	0.496
Decision Stump	0.909	0.927	0.856	0.909	0.882	0.549
J48	0.893	0.826	0.874	0.893	0.883	0.545
Naive Bayes	0.851	0.523	0.899	0.851	0.871	0.685

Using 10-fold cross-validation, the performance measures of several machine learning classifiers applied to the AR1 dataset for software defect prediction are presented in table 4.2. True Positive Rate (TP Rate), False Positive Rate (FP Rate), Precision, Recall, F-Measure, Receiver Operating Characteristic (ROC) Area (AUC) guides evaluation of the classifiers. Together, these measures reveal how well the classifiers minimise false positives and correctly identify faulty and non-defective software modules.

With a TP Rate (0.917), SMO among the classifiers shows great capacity to accurately identify faulty modules. Additionally displaying high TP rates were Lazy-IBK and Decision Stump (both 0.909). While SMO and Decision Stump showed the highest FP Rates (0.926 and 0.927, respectively), suggesting a greater tendency to misclassify non-defective modules as defective, Naïve Bayes displayed the lowest FP Rate (0.523), indicating less false alarms. Naïve Bayes performed best (0.899), thus it generated less erroneous positive classifications in terms of Precision. Confirming their dependability in defect prediction, F-Measure—which strikes a mix between Precision and Recall—was highest for Lazy-IBK (0.907), followed by SMO (0.886) and J48 (0.883). Whereas SMO (0.496) and J48 (0.545) had the lowest AUC values, implying limited discriminative power, the AUC values, which indicate the classifiers' capacity to distinguish between defective and non-defective modules, were highest for Lazy-IBK (0.776) and Naïve Bayes (0.685). While SMO and Decision Stump, despite high TP Rates, suffer from higher false positive rates and lower AUC scores, so making them less dependable for defect classification; Lazy-IBK and Naïve Bayes emerge as the most balanced classifiers overall.

4.2 Hypothesis Testing Results:

$H_0$1: There is no significant impact of Machine Learning Classification Algorithms on software defect prediction.

H_a1: There is significant impact of Machine Learning Classification Algorithms on software defect prediction.

Table 4.3: H1: Hypothesis Testing Result (One-Sample Statistics)

One-Sample Statistics				
	N	Mean	Std. Deviation	Std. Error Mean
Classification Algorithms: Performance	24	.8194	.12523	.02556

Table 4.4: H1: Hypothesis Testing Result (One-Sample T-Test)

One-Sample Test						
	Test Value = 0					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Classification Algorithms: Performance	32.054	23	.000	.81942	.7665	.8723

Whether machine learning classification algorithms significantly affect software defect prediction was the aim of the hypothesis test. With a standard deviation of 0.12523 and a standard error mean of 0.02556 the One-Sample Statistics table displays that the mean performance of classification algorithms is 0.8194. These values show that, with some variation, the classification algorithms show good performance generally. The strong departure from the null hypothesis is suggested by the One- Sample Test results, which also show a t-value of 32.054 with 23 degrees of freedom (df). The results are statistically significant since the p-value (0.000) is much less than the accepted criterion of 0.05. Furthermore, the 95% confidence interval (0.7665, 0.8723) excludes zero, so underlining the dependability of the results. The p-value is less than 0.05 thus we reject the null hypothesis ($H_0$1) and embrace the alternative hypothesis (H_a1). This underlines how statistically significantly Machine Learning Classification Algorithms affect software defect prediction. The findings imply that the accuracy and efficiency of software defect prediction models are much influenced by the choice of classification technique. These results underline the need of choosing suitable machine learning methods to increase software quality and defect detection enhancement.

$H_0$2: There is no significant difference between various classification machine learning algorithms used for software defect prediction.

H_a2: There is significant difference between various classification machine learning algorithms used for software defect prediction.

Table 4.5: Classification Algorithms and Level of Impact: Cross Tabulation

Classification Algorithms and Level of Impact						
		Level of Impact				Total
		High	Low	Very High	Very Low	
Classification Algorithms	Decision Stump	4	0	2	2	8
	J48	6	0	0	2	8
	Lazy-IBK	0	2	6	0	8
	Lazy-K Star	6	2	0	0	8
	Naïve Bayes	4	2	2	0	8
	SMO	4	0	2	2	8
Total		24	6	12	6	48

Table 4.6: Chi-Square Test Results (H2)

Chi-Square Tests			
	Value	df	Asymptotic Significance (2-sided)
Pearson Chi-Square	30.000[a]	15	.012
Likelihood Ratio	39.550	15	.001
N of Valid Cases	48		
a. 24 cells (100.0%) have expected count less than 5. The minimum expected count is 1.00.			

Various Machine Learning Classification Algorithms applied for software defect prediction were investigated using the Chi-Square test to find any appreciable variations. The degree of impact of the classification algorithms—High, Low, Very High, and Very Low—classified them. The contingency table indicates that some algorithms, such J48 and Lazy-K Star, tend to show a higher impact while others like Lazy-IBK and Decision Stump have a more varied distribution. This indicates the distribution of classification algorithms across many impact levels. The results of the Chi-Square Tests show an asymptotic significance (p-value) of 0.012 and a Pearson Chi-Square value of 30.000 with 15 degrees of freedom (df). Likewise, both of which are below the 0.05 significance threshold, the Likelihood Ratio test finds a value of 39.550 with a p-value of 0.001. We reject the null hypothesis ($H_0$2) and embrace the alternative hypothesis (H_a2) since the p-value is less than 0.05, so verifying the statistically significant variation among the several classification machine learning algorithms applied for software defect prediction. This implies that not all classifiers perform equally in defect detection and that several algorithms affect defect prediction performance at different degrees. The outcomes underline the need of choosing suitable classification techniques depending on the particular need of the defect prediction model. Future studies might investigate hybrid models and ensemble learning methods to improve robustness and predictive accuracy.

5. CONCLUSION

Using the AR1 dataset and 10-fold cross-validation, the study assessed how well different machine learning classifiers performed in predicting software defects. Key performance metrics such as True Positive Rate (TP Rate), False Positive Rate (FP Rate), Precision, Recall, F-Measure, and ROC Area (AUC) were used to evaluate the classifiers. The findings show that, with a TP Rate of 0.909 and an AUC of 0.776, the Lazy-IBK classifier performed best across the majority of metrics. In a similar vein, Naïve Bayes had a comparatively lower ROC Area than SMO and Decision Stump, which showed competitive performance. The fact that both null hypotheses (H01 and H02) were rejected demonstrates that machine learning classification algorithms have a substantial influence on software defect prediction and that their efficacy varies noticeably. These results emphasise how crucial it is to choose the right classifier depending on the performance requirements and dataset properties. To further improve defect prediction accuracy and optimise classifier selection strategies for practical software projects, future research can investigate hybrid models.

6. REFERENCE

[1] Albattah, W., & Alzahrani, M. (2024). Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach. AI, 5(4), 1743-1758. https://doi.org/10.3390/ai5040086.

[2] Boussaïd, I., Siarry, P., & Ahmed-Nacer, M. (2017). A survey on search-based model-driven engineering. Automated Software Engineering, 24(2), 233–294.

[3] Chhabra, J. K. (2017). Improving package structure of object-oriented software using multi-objective optimization and weighted class connections. Journal of King Saud University - Computer and Information Sciences, 29(3), 349–364.

[4] Elish, M. O., Aljamaan, H., & Ahmad, I. (2015). Three empirical studies on predicting software maintainability using ensemble methods. Soft Computing, 19(9), 2511–2524.

[5] Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. Journal of Systems and Software, 81(2), 186–195.

[6] Guo, J., Yang, D., Siegmund, N., Apel, S., Sarkar, A., Valov, P., Czarnecki, K., Wasowski, A., & Yu, H. (2018). Data-efficient performance learning for configurable systems. Empirical Software Engineering, 23(4), 1826–1867.

[7] Huang, Q., Shihab, E., Xia, X., Lo, D., & Li, S. (2018). Identifying self-admitted technical debt in open source projects using text mining. Empirical Software Engineering, 23(1), 418–451.

[8] Iyer, S., Konstas, I., Cheung, A., & Zettlemoyer, L. (2016). Summarizing source code using a neural attention model. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (pp. 2073–2083). Berlin, Germany: Association for Computational Linguistics.

[9] Malhotra, R., & Chug, A. (2016). Software maintainability: Systematic literature review and current trends. International Journal of Software Engineering and Knowledge Engineering, 26(9-10), 1221–1253.

[10] Mishra, S., & Sharma, A. (2015). Maintainability prediction of object-oriented software by using adaptive network-based fuzzy system technique. International Journal of Computer Applications, 119(19), 24–27.

[11] Mohd Adnan, M., Sarkheyli, A., Mohd Zain, A., & Haron, H. (2015). Fuzzy logic for modeling machining process: A review. Artificial Intelligence Review, 43(3), 345–379.

[12] Ahmed, M. A., & Al-Jamimi, H. A. (2013). Machine learning approaches for predicting software maintainability: A fuzzy-based transparent model. IET Software, 7(6), 317–326.