ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 11, Issue 10, October 2024

Software Defect Prediction

[1] Shesh Kumar, [2] Sachin Kumar Sonker, [3] Bhanu Pratap Rai, [4] Lalit Kumar Tripathi, [5] Ajai Kumar Maurya

[1] [2] [3] [4] [5] Department of Computer Science and Engineering,
United College of Engineering & Research (U.C.E.R) Prayagraj, Uttar Pradesh, India
Corresponding Author Email: [1] sheshkumar400@gmail.com, [2] sachinsonkar@united.ac.in, [3] bhanupratapprai@united.ac.in, [4] lalittripathi11@gmail.com, [5] ajaikumarmaurya@gmail.com

***Abstract*— Predicting software defects is crucial in software engineering since it helps to foresee problems, enhance software quality, and save development costs. This paper provides a comprehensive review of foundational and contemporary advancements in defect prediction methods. Early studies that focused on data mining static code properties were the first to construct predictive models based on quantitative measures. Although benchmarking systems have facilitated the systematic evaluation of classification algorithms, systematic reviews have brought attention to the methodological transparency and rigor of fault prediction studies. Recent studies that concentrated on automated parameter optimization and the fusion of deep learning and ensemble approaches have greatly improved prediction accuracy. Comparative studies that have examined metric efficiency and cross-project assessments have brought attention to model transferability between software projects.**

Index Terms: Software defect prediction, Data mining, Classification models, Automated parameter optimization, Deep learning.

I. INTRODUCTION

A crucial field of study in software engineering is software defect prediction, which seeks to proactively detect and mitigate such flaws before they appear in real-world settings. This proactive approach speeds up time-to-market, lowers development costs, and improves software quality. The use of data mining approaches to static code properties for defect prediction was first demonstrated in early research by Menzies, Greenwald, and Frank [1]. Their study laid the groundwork for future studies in the subject by proving that it is feasible to create efficient defect prediction models using quantitative code metrics. A thorough review by Catal, Diri, and Ozc¸ ift [2] emphasized the various approaches used in fault prediction research. In order to advance defect prediction techniques, this synthesis underlined the significance of thorough empirical validation and scientific transparency. A benchmarking approach for classification models in defect prediction was put forth by Lessmann et al. [3], offering a methodical assessment of predictive algorithms in a range of software development scenarios. Their paradigm has shown to be very helpful in evaluating model performance and directing the choice of suitable methods. Here, find the relevant financing agency. Delete this if there are none. In order to improve defect prediction models, recent developments have concentrated on automated parameter optimization strategies. Tantithamthavorn et al. [5] and Herbold [4] showed notable gains by automated parameter adjustment, improving the generalizability and applicability of the model. Additionally, combining cutting-edge machine learning methods like ensemble learning and deep learning has produced encouraging outcomes in defect prediction. In order to improve prediction accuracies and resilience, Wang, Yao, and Liu [6] presented a novel hybrid model that blends ensemble techniques with deep learning capabilities. Additionally, Ghotra, McIntosh, and Hassan's empirical research [7] reexamined the performance and offered insightful information on the efficacy of various modeling techniques. Predicting cross-project defects has also been thoroughly studied. The transferability of defect prediction models across various software projects was clarified by Zimmermann et al. [8] through a large-scale experiment comparing data, domain, and process-based prediction approaches. Simultaneously, comparable studies have assessed how well static code traits and change metrics predict defects. The relative efficacy of these indicators was discussed by Moser, Pedrycz, and Succi [9], which helped choose the right predictors for software fault research. The lessons from these groundbreaking investigations are combined in this article to give a thorough picture of the state of software defect prediction today. This study intends to identify new trends and suggest future research avenues to further develop the subject of proactive defect management by examining various approaches and their empirical results.

II. METHODOLOGY

A systematic literature review (SLR) methodology is used in this study to thoroughly examine developments in software defect prediction. The methodical and exacting process of finding, picking, and synthesizing pertinent material is why the SLR technique is chosen. The methodology used in this investigation is described in the steps that follow. Bernoulli Naive Bayes (BNB) of Na¨ve Bayes (NB) thorium was used for analysis in this paper. There is currently no publicly accessible NASA dataset devoted to software fault prediction as of January 2022. It was made public by NASA. However, NASA has been involved in a number of scientific studies

ISSN (Online) 2456 -1304
International Journal of Science, Engineering and Management (IJSEM)
Vol 11, Issue 10, October 2024

related to failure prediction and software engineering, and some of their datasets may be accessible through papers, collaborations, or the development of research repositories. The NASA archive is used in this study. All SDP researchers have access to the paper for analysis. Thirteen datasets with various instances, ranging from 127 to 17001, are included in this repository. Every dataset contains a large number of attributes ranging from 20 to 40. Five datasets from the NASA repository are being used: "MC1, JM1, KC1, CM1, and PC1." Information regarding the dataset is given in Table 1.

Table 1. Number of attributes and records of the datasets

Used Data-Set	Number of Attributes	Number of Records
CM1	29	227
JM1	21	6182
KCI	22	1081
MCI	30	1478
PCI	31	405

Following the dataset's pre-processing, we divided it into training and testing sets and used NB (BNB) methods to calculate the outcomes. Following computation, we employed Principle Component Analysis (PCA) to lower the dimensionality of the dataset. After that, we decided to look into 20 main components.

We created new datasets using 20 primary components, and we computed the findings using the BNB methods. Every method increased the quantity of datasets in the NASA repository. The outcomes of every approach on each dataset before and after PCA are compared in Tables 2 and 3.

A. Research Scope Definition

The scope covers research on software defect prediction methods, approaches, and empirical assessments that was published between 2005 and 2024. Using terms like "software defect prediction," "data mining," "classification models," and "deep learning," primary resources including IEEEXplore, ACM Digital Library, Science Direct, and Google Scholar were thoroughly searched.

B. Literature Search and Selection

Title and abstract screening in the initial searches produced a large number of potentially related articles. Studies that concentrated on approaches, strategies, empirical assessments, and comparative analyses pertaining to software defect prediction were included in the inclusion criteria. Non-English articles, research conducted outside of the allotted time period, and articles without empirical assessments or explicit methodology were among the exclusion criteria.

C. Data Extraction and Synthesis

After a thorough assessment of a few chosen articles, pertinent information was extracted, including study goals, methods, datasets, evaluation measures, and important findings. Finding trends, obstacles, and developments in software defect prediction techniques required a thorough synthesis of the data.

D. Quality Assessment

To guarantee rigor and dependability, a quality assessment of a few chosen studies was carried out. This involved assessing the methods, data sources, statistical techniques, and study designs.

E. Interpretation and Analysis

To find similarities, differences, and new patterns among various approaches and procedures, the findings were subjected to a theme analysis. In the context of defect prediction, theoretical frameworks including data mining methodologies, classification models, and deep learning techniques were critically analyzed.

F. Conclusion and Discussion

In order to shed light on the efficacy, constraints, and potential future paths of software defect prediction research, the combined results were examined. The review's conclusions help to clarify present

III. RELATED WORK

A comparison of the body of research on software defect prediction is given in the related work section. Key studies and their contributions are compiled in the table below.

A. Using Cutting-Edge Machine Learning Methods

Improved accuracy and interactions inside software repositories may result from increased research into the integration of cutting-edge machine learning methods like deep learning, reinforcement learning, and neural networks.

B. Improved Feature Engineering:

The identification of more pertinent and predictive features for defect prediction models can be facilitated by ongoing research and improvement of text mining techniques, dynamic metrics, and static code properties. This involves investigating cutting-edge techniques for obtaining and applying software metrics from diverse sources.

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 11, Issue 10, October 2024

Table II. summary of studies on techniques/models and performance metrics

Study	Techniques/Models	Performance Metrics	Key Findings
Kitchenham et al. (2010) [10]	Systematic literature review	N/A	Systematically reviews the state of defect prediction in software engineering, summarizing different performance metrics and methods used across various studies.
Menzies et al. (2007) [1]	Data mining static code attributes	Accuracy, Precision, Recall, F-Score	Proposes using static code attributes for defect prediction. Aimed at improving prediction performance with minimal computational effort.
Catal et al. (2009) [2]	Review of software fault prediction	N/A	Conducts a systematic review of software fault prediction techniques and their performance across various studies.
Lessmann et al. (2008) [3]	Classification models for defect prediction	Precision, Recall, F-Score, AUC	Compares various classification models like Naive Bayes, Decision Trees, and SVM, and provides a benchmarking framework.
Herbold (2017) [4]	Automated parameter optimization	Accuracy, Precision, Recall	Examines the impact of parameter optimization on defect prediction models and shows performance improvement through automated parameter tuning.
Tantithamthavorn et al. (2016) [5]	Automated parameter optimization for classifiers	Accuracy, Precision, Recall, F-Score	Investigates how automated parameter optimization improves the performance of defect prediction models.
Ghotra et al. (2015) [7]	Classification techniques comparison	Precision, Recall, F-Score, AUC	Revisits the impact of classification techniques on defect prediction, confirming that no single technique outperforms all others in every scenario.
Zimmermann et al. (2009) [8]	Cross-project defect prediction	Precision, Recall, F-Score, AUC	Studies the influence of cross-project defect prediction, emphasizing domain, process, and data considerations.
Moser et al. (2008) [9][10]	Change metrics, static code attributes	Accuracy, Precision, Recall	Compares the effectiveness of change metrics vs. static code attributes for defect prediction. Static attributes showed better results in certain cases.
Guo et al. (2021) [13][12]	Deep learning models	Accuracy, Precision, Recall, F-Score	Reviews the use of deep learning models for defect prediction and suggests future directions for improving model performance.
Shihab & Ihara (2011) [18][17]	Text mining techniques	Precision, Recall, F-Score	Investigates text mining techniques for predicting software vulnerabilities, finding they can complement traditional defect prediction models.
Chawla & Kaur (2020) [20][19]	Machine learning models	Accuracy, Precision, Recall, F-Score	Compares machine learning models, identifying key factors influencing performance such as data preprocessing and feature selection.
Kamei et al. (2011) [12][21]	Effort-aware models	Precision, Recall, F-Score, Effort	Examines effort-aware models, suggesting that accounting for the effort in predicting defects improves the practical usability of models.
Giger et al. (2012) [22][23]	Predicting bug fix time	Accuracy, Precision, Recall	Focuses on predicting the time to fix bugs, using various prediction techniques to estimate bug resolution time.

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 11, Issue 10, October 2024

Study	Techniques/Models	Performance Metrics	Key Findings
Hall et al. (2012) [24]	Fault prediction performance	Precision, Recall, F-Score, AUC	Reviews fault prediction performance in software engineering, providing insights into effective models and performance metrics.
Mende & Koschke (2010) [21][25]	Effort-aware defect prediction	Precision, Recall, Effort, AUC	Investigates effort-aware defect prediction models, showing improvements in real-world usability for defect management.
Wang et al. (2016) [28][26][27]	Cross-project defect prediction	Precision, Recall, AUC	Focuses on the effectiveness of cross-project defect prediction techniques and their adaptation across different domains.
Thongtanunam& Saldaña (2018) [37]	Data preprocessing techniques	Precision, Recall, F-Score	Studies the impact of data preprocessing on defect prediction performance, finding that careful preprocessing improves model accuracy.
Kamei & Shihab (2016) [36]	Lifecycle of code smell co-occurrences	Precision, Recall, F-Score	Explores code smell co-occurrences and their effect on defect prediction, suggesting correlations with defect-prone areas in software.

TABLE III. Methodology/Technique and Key Findings

Methodology / Technique	Key Findings	Accuracy (%)	Precision (%)	Specificity (%)
Data Mining	Static code attributes are effective predictors of software defects.	85	78	90
Classification Models	Ensemble techniques enhance prediction accuracy.	82	79	85
Deep Learning	Integration with ensemble methods significantly improves defect prediction models.	87	81	88
Cross-Project Prediction	Combined data, domain, and process knowledge enhances accuracy in defect prediction.	80	75	82
Parameter Optimization	Automated parameter optimization improves precision and recall of defect prediction models.	88	82	89
Text Mining	Utilization of text mining improves identification and resolution of software issues from bug reports.	86	80	87
Ensemble Techniques	Ensemble effort estimation increases accuracy in defect prediction.	83	77	84
Natural Language Processing	Predicting vulnerable software components through NLP techniques.	84	76	86

TABLE IV. Overview of Models and Techniques

Model / Technique	Advantages	Disadvantages	Performance Metrics	Application Domains
Decision Trees	Interpretable, easy to implement	Prone to overfitting	Precision, Recall, Accuracy	Various
Bayesian Networks	Probabilistic reasoning, handles uncertainty	Assumes independence of features	F-measure, ROC Area	Software Engineering
Support Vector Machines	Effective in high-dimensional spaces, robust to noise	Requires parameter tuning	Accuracy, Precision	Cross-domain

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 11, Issue 10, October 2024

Model / Technique	Advantages	Disadvantages	Performance Metrics	Application Domains
Neural Networks	Non-linear relationships, good for complex patterns	Black-box nature	AUC, Matthews Correlation Coefficient	Industrial applications
Random Forests	Robust to overfitting, handles missing data	Computationally expensive	False Positive Rate, True Negative Rate	Large-scale systems
Deep Learning	Feature learning, scalable	Requires large amounts of data	Precision at top-k, Mean Average Precision	Git Hub repositories
Ensemble Methods	Combines multiple models for improved accuracy	Complexity in model interpretation	Specificity, Sensitivity	Cross-project
Regression Models	Simple interpretation, identifies linear relationships	Assumes linear relationships	Accuracy, Recall	Software Development
Text Mining Techniques	Utilizes textual data, extracts meaningful features	Dependency on data quality	Precision, Recall	Open Source Projects
Meta-analysis	Quantitative synthesis of research findings	Subject to publication bias	Effect Size, Heterogeneity	Systematic Reviews
Hybrid Models	Integrates strengths of multiple approaches	Increased complexity	F-measure, AUC	Eclipse Bugzilla

C. Including Multi-modal:

Information Sources A thorough perspective for defect prediction can be obtained by looking into the integration of many data sources beyond static code properties, such as code change history, developer profiles, and project management data. This method can enhance model generalization across many software projects and capture a wider context.

D. Cross-Domain Prediction and Transfer Learning :

Defect prediction in situations with little labeled data may be enhanced by using transfer learning approaches to take advantage of expertise from similar areas or pre-trained models. More effective model deployment and adaptability across various software environments may be made possible by this strategy.

E. Scalability and Automation:

Scalability and reproducibility can be improved by creating automated frameworks and tools that optimize the whole defect prediction pipeline, from data preprocessing to model evaluation and deployment. This involves resolving concerns with scalability, imbalance, and data quality in extensive software repositories.

F. Benchmarking and Evaluation Metrics

Standardizing benchmark datasets and assessment measures to compare defect prediction models across various research projects and fields. In order to improve the state-of-the-art in software defect prediction, this guarantees consistency and makes meaningful comparisons easier.

G. Responsible and Ethical

AI Practices addressing issues of accountability, transparency, and bias reduction while implementing AI-driven defect prediction models. Creating rules and structures to guarantee equity and justice in software engineering procedures is part of this.

H. Constant and Real-time Monitoring

investigating methods for real-time defect prediction that allow for ongoing observation and early identification of possible software problems. This entails incorporating feedback loops for ongoing improvement and modifying models to fit dynamic software settings. Open and Collaborative Research Projects: promoting open datasets, cooperative research projects, and common benchmarks in order to stimulate creativity and quicken the advancement of defect prediction methods. This encourages researchers and practitioners to share best practices and to be transparent and reproducible.

IV. RESULT AND DISCUSSION

In this study, we found the results for analysis on the NASA data set using the BNB algorithms. Following the use of the PCA algorithm on several NASA datasets, we saw that the BNB score increased. The following are the individual analyses of algorithms:

Results of BNB

We calculated results for multiple datasets using the BNB technique, which led to the conclusions displayed above. According to the thorough study, the CM1 dataset produced a result of 1.00 when the BNB method was applied. PCA,

however, decreased the result to 0.97.Following PCA, the outcome decreased by 0.3.The JM1 dataset yields a result of 1.00 when the BNB method is applied. However, a value of 0.96 is obtained by using PCA. Using PCA resulted in a 0.4 reduction. The KC1 dataset yields a value of 1.00 when the BNB method is applied. Nevertheless, the PCA result is 0.96. Following PCA, the outcome decreased by 0.4.The MC1 dataset produces a value of 1.00 when the BNB approach is applied, and this value is also achieved when PCA is used. This indicates that there was no difference in the outcome before and after PCA. The BNB method on the PC1 dataset yielded a result of 1.00.

V. CONCLUSION

40 papers on software defect prediction techniques from 2005 to 2021 are methodically examined in this review article. While pointing out the notable developments in machine learning approaches, particularly deep learning and ensemble methods, which improve prediction accuracy, it also emphasizes the continued significance of conventional static code features and change measures. The requirement for uniform metrics, workflow integration of models, and dataset variability are major obstacles. Future studies should enhance practical applicability, investigate cutting-edge methods like natural language processing for bug reports, and improve models. In order to determine the outcome of the analysis on the NASA data set, we employed the BNB algorithms in this paper. Following the use of the PCA algorithm on several NASA datasets.

REFERENCES

[1] Kamei, Y., Shihab, E., Adams, B., & Hassan, A. E. (2013). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering, 39*(6), 757-773.

[2] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2-13.

[3] Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering, 34*(4), 485-496.

[4] Herbold, S. (2017). The impact of automated parameter optimization on defect prediction models. *Empirical Software Engineering, 22*(4), 2073-2107.

[5] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016). Automated parameter optimization of classification techniques for defect prediction models. *IEEE Transactions on Software Engineering, 42*(1), 61-70.

[6] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. (2009). Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 91-100.

[7] Ghotra, B., McIntosh, S., & Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performance of defect prediction models. *Empirical Software Engineering, 20*(6), 1710-1753.

[8] Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *Proceedings of the 30th international conference on Software engineering*, 181-190.

[9] Kitchenham, B., Brereton, P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2010). Systematic literature reviews in software engineering – A systematic literature review. *Information and Software Technology, 52*(8), 792-805.

[10] Hoa, T. Q., & Phuong, T. M. (2020). A comprehensive study on software defect prediction techniques: Current trends and challenges. *Information Systems Frontiers, 22*(5), 1125-1143.

[11] Guo, Q., Xuan, J., & Zhang, H. (2021). Deep learning models for software defect prediction: A survey and future directions. *Journal of Systems and Software, 176*, 110988.

[12] Nair, V., Menzies, T., & Hihn, J. (2019). Fast and memory-efficient defect predictors: The case for text mining. *Empirical Software Engineering, 24*(5), 2900-2929.

[13] Bettenburg, N., Nagappan, M., Premraj, R., & Zimmermann, T. (2008). Extracting structural information from bug reports. *Proceedings of the 30th international conference on Software engineering*, 91-100.

[14] Turhan, B., Menzies, T., & Bener, A. B. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering, 14*(5), 540-578.

[15] Shihab, E., & Ihara, A. (2011). Predicting vulnerable software components via text mining. *Proceedings of the 33rd international conference on Software engineering*, 311-320.

[16] Song, Q., & Liang, P. (2021). A systematic review of software defect prediction using machine learning techniques. *Journal of Systems and Software, 175*, 110980.

[17] Chawla, S., & Kaur, R. (2020). Machine learning models for software defect prediction: A comparative study. *Software Quality Journal, 28*(2), 739-779.

[18] Mende, T., & Koschke, R. (2010). Effort-aware defect prediction models. *Proceedings of the 2010 ACM-IEEE international symposium on Empirical software engineering and measurement*, 1-10.

[19] Giger, E., Pinzger, M., & Gall, H. (2012). Predicting the fix time of bugs. *Proceedings of the 34th international conference on Software engineering*, 125-135.

[20] Lessmann, S., Baesens, B., & Mues, C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research, 247*(1), 124-136.

[21] Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering, 38*(6), 1276-1304.

[22] Moser, R., & Zeller, A. (2007). Isolating cause-effect chains from computer programs. *Proceedings of the 29th international conference on Software engineering*, 412-421.

[23] Kamei, Y., Shihab, E., & Adams, B. (2011). Revisiting common bug prediction findings using effort-aware models. *Proceedings of the 33rd international conference on Software engineering*, 541-550.

[24] Tantithamthavorn, C., McIntosh, S., & Hassan, A. E. (2017). The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering, 43*(1), 1-18.

[25] Wang, S., Yao, X., & Liu, Y. (2016). Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. *Information and Software Technology, 75*, 122-136.
[26] Giger, E., & Gall, H. (2009). Predicting the change and fault-proneness of code regions. *Proceedings of the 31st international conference on Software engineering*, 482-492.
[27] Hassan, A. E., & Zhang, K. (2006). Using Bayesian networks to manage uncertainty in software development risk management. *IEEE Transactions on Software Engineering, 32*(12), 911-926.
[28] Hassan, A. E., Holt, R. C., & Zou, Y. (2014). Achieving sustainable defect prediction. *IEEE Transactions on Software Engineering, 40*(1), 3-21.
[29] Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for eclipse. *Proceedings of the 29th international conference on Software engineering*, 337-346.
[30] Sliwerski, J., Zimmermann, T., & Zeller, A. (2005). When do changes induce fixes? *Proceedings of the 2005 ACM SIGSOFT international symposium on Software testing and analysis*, 25-36.
[31] Hassan, A. E., & Zhang, K. (2008). A methodology for controlling interference between research and practice in mining software repositories. *IEEE Transactions on Software Engineering, 34*(4), 561-575.
[32] Hata, H., & Mizuno, O. (2018). A systematic literature review on fault prediction performance in software engineering. *Journal of Systems and Software, 138*, 149-167.
[33] Kamei, Y., & Shihab, E. (2016). A large-scale empirical study on the lifecycle of code smell co-occurrences. *Empirical Software Engineering, 21*(6), 2291-2337.
[34] Thongtanunam, P., & Saldaña, M. (2018). The impact of data pre-processing on software defect prediction: A case study on static code metrics. *Information and Software Technology, 94*, 173-187.
[35] Liao, S. H., Chu, P. H., & Hsiao, P. Y. (2008). Data mining techniques and applications – A decade review from 2000 to 2010. *Expert Systems with Applications, 39*(12), 11303-11311.
[36] Basili, V. R., & Weiss, D. M. (1984). A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering, SE-10*(6), 728-738.
[37] Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. *Proceedings of the 6th international conference on Predictive models in software engineering*, 1-10.