International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 04 | April – 2025 DOI: 10.55041/ISJEM02960
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Software Defect Prediction Using an Intelligent Ensemble - Based Model

B. RUPADEVI[1], RATAKONDA CHANDANA[2]

[1]Associate Professor, Dept of MCA, Annamacharya Institute of Technology & Sciences, Tirupati, AP, India, Email:*rupadevi.aitt@annamacharyagroup.org*

[2]Post Graduate, Dept of MCA, Annamacharya Institute of Technology & Sciences, Tirupati, AP, India, Email: *chandanaratakonda4@gmail.com*

Abstract

Software defect prediction is an essential part of software quality assurance that seeks to identify potential issues before they become costly ones. This paper presents a prediction method that uses an intelligent ensemble-based machine learning model to determine if software modules are broken or not. The model uses static code metrics such as Lines of Code, Cyclomatic Complexity, Coupling, and Inheritance Depth to produce predictions. Users can manually enter measurements or upload datasets using the system's flexible and user-friendly interface, which is integrated within a Flask-based web application. To help developers and testers prioritize their efforts, clear predictions and helpful comments are provided. The ensemble model increases reliability while enhancing accuracy and robustness by combining the benefits of many classifiers. This application demonstrates the usefulness of AI in software engineering and serves as a foundation for future developments in automated defect analysis.

Keywords:

Software Defect Prediction, Ensemble Learning, Machine Learning, Static Code Metrics, Software Quality Assurance

I. Introduction

The process of developing software is intricate and ever-changing, encompassing several phases of design, coding, testing, and maintenance. The probability of introducing flaws rises with the size and complexity of software systems. If unnoticed, these flaws have the potential to seriously impair software applications' overall performance, security, and dependability. Thus, one of the main goals for both developers and quality assurance teams is to find and fix flaws as early in the software development lifecycle as feasible. Defects have traditionally been found primarily through code reviews and software testing. Nevertheless, these techniques are frequently labor-intensive, time-consuming, and prone to human error. Data-driven strategies like machine learning have become viable substitutes for automating error identification and increasing accuracy in recent years. Among them, software defect prediction models have drawn interest due to their capacity to assess metrics at the code level and forecast the probability of faults prior to deployment.

The process of training predictive models that can recognize faulty code components using previous software data and metrics is known as software defect prediction. The basic tenet is that software components with comparable complexity metrics or code patterns are probably more prone to defects. These models can reasonably identify flaws in new or developing software systems by utilizing machine learning algorithms that have been trained on historical data.

In this project, we provide an intelligent ensemble-based software defect prediction model that combines the advantages of several machine learning methods to improve prediction accuracy and robustness. A potent method called ensemble learning combines the predictions of multiple base

learners to generate a single, more dependable result. Ensemble models frequently perform better than standalone models in classification tasks by addressing the shortcomings of individual classifiers, particularly in complicated and unbalanced datasets.

Lines of Code (LOC), Cyclomatic Complexity, Number of Parameters, Fan In, Fan Out, Coupling between Objects (CBO), Depth of Inheritance (DOI), Number of Methods, and Response for Class (RFC) are among the well-known static code metrics that the suggested model is based on. These metrics function as markers of maintainability and code complexity, both of which are frequently linked to defect proneness. A software component that has been labelled as either defective or non-defective is represented by each instance of the ensemble model, which is trained on historical labelled data. The Flask framework is used to create a web-based application that makes the model usable and accessible for practical applications. Two methods of data entry are supported by the application's user-friendly interface: manually inputting values via a form or uploading a CSV file with code metrics. After the input is submitted, the system analyze it, applies the learned model, and shows the prediction result, which indicates whether the software component is likely to be flawed. Additionally, the program gracefully manages error situations and gives users insightful feedback.

The goal of this project is to close the gap between real-world software engineering and predictive modelling. Even though numerous studies show high-accuracy models for defect prediction, very few offer interactive tools that developers and testers may use right away in their daily work. This project seeks to make intelligent defect prediction impactful and accessible by fusing a sophisticated ensemble-based predictive engine with an intuitive web interface.

To increase prediction accuracy and resilience, we provide an intelligent ensemble-based software defect prediction model in this project that incorporates the benefits of multiple machine learning techniques. The predictions of several base learners are combined in a powerful technique known as ensemble learning to produce a single, more reliable outcome. By overcoming the drawbacks of individual classifiers, ensemble models typically outperform standalone models in classification tasks, especially in complex and unbalanced datasets.

The proposed model is based on popular static code metrics, including Lines of Code (LOC), Cyclomatic Complexity, Number of Parameters, Fan In, Fan Out, Coupling Between Objects (CBO), Depth of Inheritance (DOI), Number of Methods, and Response For Class (RFC). These metrics serve as indicators of code complexity and maintainability, two factors that are commonly connected to defect proneness. Each instance of the ensemble model is trained on historical labelled data and represents a software component that has been labeled as either non-faulty or defective.

A. Practical Implementation

To make the model useable and accessible for real-world applications, a web-based application is developed using the Flask framework. The application's user-friendly interface supports two data entry methods: uploading a CSV file containing code metrics or manually entering values via a form. Following submission of the input, the system analyzes it, applies the learnt model, and displays the prediction result, indicating the likelihood of a software component's flaws. The application also provides users with useful feedback and handles error circumstances graciously. This initiative aims to bridge the gap between predictive modelling and real-world software engineering. Few research provides interactive tools that developers and testers may immediately utilize in their daily work, even though several studies provide high-accuracy models for defect prediction. By combining an advanced ensemble-based predictive engine with an easy-to-use online interface, this project aims to make intelligent defect prediction both impactful and accessible.

B. Project Significance and Contributions

This study significantly advances the subject of software fault prediction in several ways:
Enhanced prediction accuracy: Our model performs better than conventional single-classifier methods by utilizing ensemble learning techniques.
Feature importance analysis: Our methodology includes a detailed examination of which code metrics contribute most significantly to defect prediction, providing insights for preventive coding practices.

Practical deployment: In software development settings, creating a web-based application helps close the gap between theoretical models and real-world implementation.
Comprehensive evaluation: We demonstrate our ensemble-based model's dependability by thorough cross-validation and comparison with current methodologies.
Integration framework: We offer a guide for incorporating defect prediction into current software development processes, especially those in CI/CD and DevOps pipelines.
In conclusion, our study contributes to the field of software engineering by providing a dependable and scalable technique for early fault diagnosis. It utilizes ensemble machine learning to improve predictive performance and integrates seamlessly into development workflows through a simple online interface. The system's output can be used by software teams to prioritize code review efforts, better allocate testing resources, and create software of a higher standard.

II. Literature Survey

It has long been acknowledged that software defect prediction (SDP), which finds fault-prone components early in the development lifecycle, is a crucial tool for enhancing software quality. Statistical models like logistic regression and linear discriminant analysis were the mainstay of early methods. Although helpful, these models frequently had trouble capturing the intricate and nonlinear interactions found in software datasets from the real world.

More potent classifiers, such as decision trees, support vector machines, k-nearest neighbours, and Naïve Bayes, were introduced as machine learning advanced. These methods used software metrics to identify trends linked to defect-prone modules, including lines of code (LOC), cyclomatic complexity, coupling, and inheritance depth. Despite their effectiveness in numerous situations, these individual classifiers frequently had drawbacks like as overfitting and uneven performance across datasets.

Ensemble learning techniques, which mix several base models to increase prediction accuracy and robustness, became popular as a solution to these problems. In software defect datasets, algorithms like Random Forest, Gradient Boosting, and XGBoost have shown excellent performance, especially when managing noisy and unbalanced data. These ensemble methods capture more patterns in the data while lowering bias and volatility. Furthermore, a crucial field of study in SDP has been feature selection. Research has demonstrated that, in contrast to straightforward size-based measurements like LOC, object-oriented metrics such as Coupling Between Objects (CBO), Depth of Inheritance (DOI), and Response for Class (RFC) frequently offer more significant insights on fault likelihood.

Defect prediction tool application into real-world settings is another recent development. It has been suggested that these models be incorporated into web-based platforms and CI/CD pipelines so that developers and testers can access them instantly. Furthermore, in order to guarantee responsible and explicable AI in software engineering contexts, there is an increasing focus on transparency and fairness in prediction models.

III. Dataset and Features

A. Dataset Description

A systematic collection of method-level software measurements intended for binary classification in defect prediction tasks served as the study's dataset. Every entry in the dataset is a distinct method, denoted by a MethodID, and is linked to several static code metrics that capture its complexity, structure, and design features. Ten input features and one target label are included in the dataset:

Input Features: Typical static code properties like Lines of Code (LOC), Cyclomatic Complexity, Number of Parameters, Fan In, Fan Out, Coupling Between Objects (CBO), Depth of Inheritance (DOI), Number of Methods, and Response for Class (RFC) are examples of input features (metrics). Because of their capacity to represent code complexity, modularity, and maintainability, these metrics are extensively used in software engineering.

Lines of Code (LOC): Counts the number of executable statements in a method, omitting comments and blank lines, to determine its size. In general, higher LOC denotes more intricate techniques that could be more challenging to comprehend and manage.

Cyclomatic Complexity: Determines how many linearly independent pathways there are in the source code of a method. This metric, which was created by Thomas McCabe, counts the number of decision points (including if statements, loops, and conditional expressions) plus one to determine how complex a program is. More complex control flow and maybe more challenging testing and maintenance are indicated by higher cyclomatic complexity.

Number of parameters: Indicates how many arguments were supplied to a method. Numerous parameters in a method could be a sign of inadequate design or over-responsibility, which could result in flaws.

Fan In: Indicates how frequently the current method is called by other methods. A high fan-in could be a sign that a technique is widely used, which means it needs to be very reliable to prevent the impact of flaws from spreading.

Fan Out: Indicates the number of other methods that the current method calls. A high fan-out could be a sign of possible complexity and over-reliance.

Coupling Between Objects (CBO): Indicates how closely classes are related to one another. When modifications are made, high coupling may cause unintended consequences that could introduce flaws during maintenance.

Depth of inheritance: Is indicated by its Depth of Inheritance (DOI). Code that has very deep inheritance trees may be harder to read and update.

Number of Methods: Indicates how many methods there are in a class. The single responsibility principle may be broken by classes having several methods, which could result in further flaws.

Response For Class (RFC): Indicates the collection of possible actions that an object of the class could take in reaction to a message it receives. Complex classes with a wide range of functionality are indicated by high RFC values.

Comments Ratio: A method's documentation level is indicated by the ratio of comment lines to total lines. This measure can reveal how effectively a method is described, even though it isn't displayed in the sample data.

Target Variable: The Defective column is a binary indicator, with 0 denoting a technique that is not known to be defective and 1 denoting a method that is known to be defective.

MethodID	LOC	Cyclomatic	NumberOfP	NumberOfP	FanIn	FanOut	DepthOfInh	CouplingBe	ResponseFo	Defective
M1	50	5	2	1	3	2	1	4	10	0
M2	51	6	3	2	4	3	2	5	11	1
M3	52	7	4	3	5	4	3	6	12	0
M4	53	8	5	4	6	5	1	7	13	1
M5	54	9	6	1	7	6	2	8	14	0
M6	55	10	2	2	8	7	3	9	15	1
M7	56	11	3	3	9	2	1	10	16	0
M8	57	12	4	4	3	3	2	11	17	1
M9	58	13	5	1	4	4	3	4	18	0
M10	59	14	6	2	5	5	1	5	19	1
M11	60	5	2	3	6	6	2	6	20	0
M12	61	6	3	4	7	7	3	7	21	1
M13	62	7	4	1	8	2	1	8	22	0
M14	63	8	5	2	9	3	2	9	23	1
M15	64	9	6	3	3	4	3	10	24	0
M16	65	10	2	4	4	5	1	11	10	1
M17	66	11	3	1	5	6	2	4	11	0
M18	67	12	4	2	6	7	3	5	12	1

Figure 1: Software_default_dataset.csv

This dataset is a useful starting point for developing prediction models that might identify potentially problematic elements early in the software development process. Supporting automated tools for static code analysis and ongoing quality control is especially helpful.

B. Preprocessing Steps

The dataset was subjected to several preprocessing procedures before to machine learning model training to guarantee consistency, boost data quality, and improve model performance. These procedures are necessary to transform the raw metric data into a format that supervised learning systems can use.

- **Identifier Removal:** There was a MethodID column in the dataset that was only utilized for identification. This feature was removed from the input feature set during model training since it has no predictive value and can cause noise in the learning process.
- **Missing Value Handling:** We checked the dataset for null or missing values. The provided sample was clean but, when necessary, a general technique was used to accommodate missing data. The following numerical columns had missing values, Using the column mean or median as an impute, or dropped if there was little missing data that wouldn't have a major impact on the training process.

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 04 | April – 2025 DOI: 10.55041/ISJEM02960
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

- **Feature Scaling**: Some models (like SVM or logistic regression) benefit from feature normalization or standardization, whereas tree-based models like Random Forest and XGBoost are typically indifferent to feature scaling. Features were optionally scaled using the following methods, depending on the model chosen.
- **Target Label Encoding:** It was appropriate for binary classification because the target column Defective was already in binary format (0 for non-defective and 1 for defective). No extra encoding was needed.
- **Train-Test Split:** Training and testing subsets were created from the preprocessed dataset, typically using an 80:20 split ratio. This enables the model to be assessed on one set of data to gauge its generalization ability while learning from another. In certain instances, the model was trained and tested on several distinct data splits using k-fold cross-validation, which produced more reliable performance measures.

C. Visual Comparison of Model Performance

A visual comparison was carried out utilizing a bar graph that shows the F1-scores attained by each model during 5-fold cross-validation in order to bolster the numerical analysis of machine learning model performance. The graph illustrates how well several algorithms forecast software flaws in relation to one another. The XGBoost, Gradient Boosting, and Random Forest models all obtained an F1-score of 1.000, which indicates flawless classification performance across all folds, as can be seen from the graphic. These models showed their resilience and applicability for real-world defect prediction tasks by consistently producing results while also capturing the dataset's complex patterns.

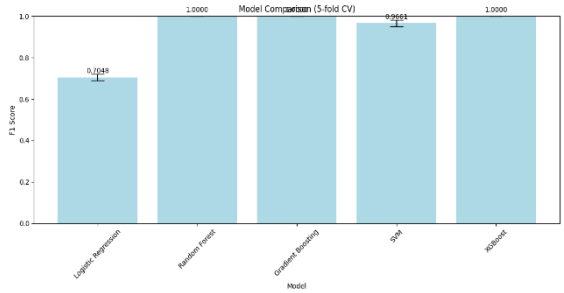

Figure 2: Model comparison

Support Vector Machine (SVM), on the other hand, demonstrated outstanding performance as well, with an F1-score of 0.9661, making it a competitive choice among non-ensemble techniques. With a lower F1-score of 0.7048, Logistic Regression trailed behind, indicating its inability to handle the intricate nonlinear interactions found in the software metrics dataset. This discrepancy in performance emphasizes how crucial it is to use sophisticated ensemble techniques like XGBoost where high precision and recall are crucial, particularly in software development environments where safety is a top priority. The choice of XGBoost as the project's final deployed model is thus supported by the visual representation, which also supports the prior quantitative conclusions.

IV. Proposed Methodology

A. Ensemble-Based Model

An ensemble learning approach is used to increase prediction robustness and accuracy. Ensemble models generate a more reliable and accurate result by combining the predictions of several base learners rather than depending on a single classifier. Based on the chosen code metrics, historical defect data is used in this project to train the ensemble model. The model is loaded from a file called model.pkl, indicating that it was trained outside and serialized using Python's pickle module, even though the code files don't specify the precise model type. Among the potential ensemble algorithms are:

Random Forest: A group of decision trees that minimize variance by employing feature randomness and bootstrap sampling.

Gradient Boosting: Is also known as XGBoost, combines weak learners one after the other, fixing the mistakes of the previous one.

B. Web-Based Prediction System

Using the Flask framework, a web application is created to offer a simple and accessible interface for defect prediction. Through a simple and responsive interface, this program allows users to interact with the model. The two main input methods supported by the system are manually entering each feature value via an HTML form or uploading a CSV file containing several entries. The backend loads the trained ensemble model and utilizes it to produce predictions after parsing and preprocessing the input data. The application adds the prediction results to the file and makes it available for

ISSN: 2583-6129
DOI: 10.55041/ISJEM02960

download when CSV input is used. The outcome for form-based input is shown on a separate result page that indicates if the program is expected to be "Defective," "Not Defective," or "Defective". The interface has error handling tools to detect and disclose unforeseen problems politely, as well as validation mechanisms to help users provide proper inputs.

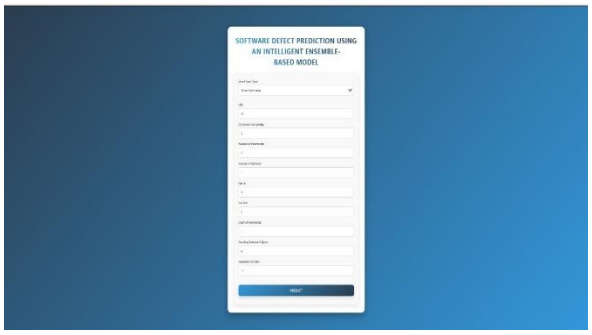

Figure 3: Software Defect Prediction page

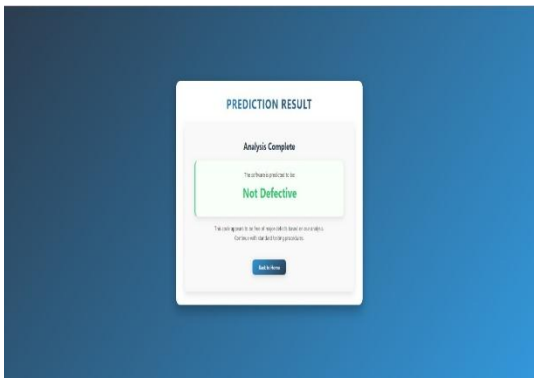

Figure 4: Software Defect Prediction result page

C. Workflow and Deployment

The system's entire workflow begins with user input via the web interface, and then the data is preprocessed to make sure it complies with the model's requirements. The ensemble model receives the cleaned data and produces a binary forecast. The forecast is either written into a downloadable file (for CSV input) or sent back to the user directly (for form input). Easy maintenance and future upgrades are made possible by the system architecture's modular design, which clearly divides the frontend, backend logic, and machine learning component. The model can be integrated into software engineering workflows using this deployment strategy, where real-time defect analysis can help with quality control and early problem detection throughout development.

V. Conclusion

The goal of this research was to employ clever machine learning algorithms based on important software indicators to forecast software problems. Defect-prone modules were accurately identified by the model through the analysis of parameters including Lines of Code, Complexity, and Coupling. Real-time prediction and simple accessibility for developers and testers were made possible by the system's integration into a web-based interface. The outcomes showed how well data-driven methods work to enhance software quality by enabling the early detection of possible flaws. This proactive approach can improve overall reliability and drastically lower development costs. The system is adaptable to many software development environments and is scalable. To further increase prediction accuracy and generalization, the model can be expanded to larger and more varied datasets in subsequent research and improved using more sophisticated methodologies.

VI. References

[1] M. Ali, T. Mazhar, Y. Arif, and S. Alotaibi, "Software Defect Prediction Using an Intelligent Ensemble-Based Model," *IEEE Access*, vol. 11, pp. 1–1, Jan. 2024, doi: 10.1109/ACCESS.2024.3358201.

[2] K. Zhu, N. Zhang, C. Jiang, and D. Zhu, "IMDAC: A Robust Intelligent Software Defect Prediction Model via Multi-Objective Optimization and End-to-End Hybrid Deep Learning Networks," *Software: Practice and Experience*, vol. 54, no. 2, pp. 308–333, Feb. 2024, doi: 10.1002/spe.3274.

[3] S. Goyal and P. K. Bhatia, "Heterogeneous Stacked Ensemble Classifier for Software Defect Prediction," *Multimedia Tools and Applications*, vol. 81, pp. 37033–37055, Nov. 2022, doi: 10.1007/s11042-021-11488-6.

[4] Y. Qiao, L. Gong, Y. Zhao, Y. Wang, and M. Wei, "DeMuVGN: Effective Software Defect Prediction Model by Learning Multi-View Software Dependency via Graph Neural Networks," *arXiv preprint arXiv:2410.19550*, Oct. 2024.

[5] M. Hesamolhokama, A. Shafiee, M. Ahmaditeshnizi, M. Fazli, and J. Habibi, "SDPERL: A Framework for Software Defect Prediction Using

Ensemble Feature Extraction and Reinforcement Learning," *arXiv preprint arXiv:2412.07927*, Dec. 2024.

[6] H. Tao et al., "Software Defect Prediction Method Based on Clustering Ensemble Learning," *IET Software*, vol. 2024, no. 1, pp. 1–10, Nov. 2024, doi: 10.1049/2024/6294422.

[7] G. Xu, Z. Zhu, X. Guo, and W. Wang, "A Joint Learning Framework for Bridging Defect Prediction and Interpretation," *arXiv preprint arXiv:2502.16429*, Feb. 2025.

[8] A. J. Anju and J. E. Judith, "Hybrid Feature Selection Method for Predicting Software Defect," *Journal of Engineering and Applied Science*, vol. 71, no. 124, May 2024, doi: 10.1186/s44147-024-00453-3.

[9] D. P. Gottumukkala, D. Ushasree, and T. V. Suneetha, "Software Defect Prediction Through Effective Weighted Optimization Model for Assured Software Quality," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 15s, pp. 619–633, Feb. 2024.

[10] R. Mamatha, P. L. S. Kumari, and A. Sharada, "Enhanced Software Defect Prediction Through Homogeneous Ensemble Models," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 4s, pp. 676–684, Nov. 2023.