Advances in Nonlinear Variational Inequalities
ISSN: 1092-910X
Vol 28 No. 7s (2025)

A Hybrid Graph-Based and Evolutionary Learning Approach for Enhanced Software Fault Prediction Using Machine Learning and Deep Neural Architectures

Durga[1], Dr. Anupa Sinha[2]

[1]Research Scholar, [2]Assistant Professor

[1,2]Department of Computer Science and Engineering

[1,2]Kalinga University, Naya Raipur [C.G.], India

Article History:

Received 09/11/2025

Revised 19/12/2025

Accepted 28/12/2025

Abstract

Software fault prediction has become an essential aspect of modern Software Engineering, aimed at improving software quality and reducing maintenance costs by identifying defect-prone modules at an early stage. This study presents a comprehensive comparative analysis of multiple predictive approaches, including traditional machine learning models, ensemble techniques, deep learning architectures, graph-based learning models, and a proposed hybrid optimization model. The research utilizes a structured dataset of student programming metrics, incorporating preprocessing techniques such as data cleaning, normalization, class balancing using SMOTE and SMOTE-Tomek, and feature selection through Chi-Square and Mutual Information methods. A variety of models, including Decision Tree, Random Forest, Support Vector Machine, AdaBoost, Gradient Boosting Machine, XGBoost, Deep Neural Network, Convolutional Neural Network, Graph Convolutional Network, and a hybrid Genetic Algorithm–Decision Tree (GA-DT) model, were implemented and evaluated under consistent experimental conditions. The performance of these models was assessed using standard evaluation metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC. The results indicate that ensemble and deep learning models outperform traditional classifiers, while graph-based models further enhance prediction capability by capturing structural relationships within software systems. The proposed GA-DT hybrid model achieved the highest performance with an accuracy of 97.13%, demonstrating the effectiveness of combining evolutionary optimization with machine learning techniques. Statistical validation using a paired t-test confirmed that graph-based models significantly improve prediction performance compared to conventional approaches. The findings highlight the importance of integrating structural learning, optimization techniques, and advanced neural architectures for accurate and reliable software fault prediction. This study contributes to the development of intelligent, data-driven solutions for enhancing software reliability and quality assurance processes.

Keywords: Software Fault Prediction; Machine Learning; Deep Learning; Graph Convolutional Network; Genetic Algorithm

INTRODUCTION

Software reliability has become a critical concern in modern software engineering due to the increasing complexity of software systems and their widespread use in safety-critical and

business-critical applications. Software faults not only degrade system performance but can also lead to severe financial losses and security vulnerabilities. Consequently, early detection of defective modules during the software development lifecycle has emerged as a key research area within Software Engineering and Machine Learning. Traditional fault detection methods, such as manual code reviews and testing, are often time-consuming, error-prone, and insufficient for handling large-scale systems (Pressman, 2014). In recent years, data-driven approaches have gained significant attention for software fault prediction. Machine learning techniques utilize historical software metrics such as code complexity, coupling, cohesion, and size to build predictive models that classify software modules as defective or non-defective. Classical models like Decision Trees, Support Vector Machines, and Random Forests have demonstrated promising results in predicting software defects (Lessmann et al., 2008). However, these models primarily rely on numerical features and often fail to capture complex nonlinear relationships and structural dependencies within software systems (Hall et al., 2012). To overcome these limitations, ensemble learning methods such as boosting and bagging techniques have been introduced. Algorithms like AdaBoost, Gradient Boosting Machine, and XGBoost combine multiple weak learners to improve prediction accuracy and generalization capability. These methods have shown superior performance compared to standalone classifiers by reducing variance and bias in predictions (Chen & Guestrin, 2016). Despite these advancements, ensemble models still depend heavily on feature engineering and may not fully exploit hidden patterns within the data.

The emergence of deep learning has further transformed the field of software fault prediction. Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) are capable of automatically learning hierarchical feature representations from raw input data. These models can capture complex nonlinear relationships among software metrics, leading to improved predictive performance (Wang et al., 2016). However, deep learning models often require large datasets and may struggle to incorporate structural relationships inherent in software systems. An important advancement in this domain is the application of graph-based learning techniques, particularly Graph Convolutional Networks (GCNs). Unlike traditional models, GCNs represent software systems as graphs, where nodes correspond to program components and edges represent relationships such as control flow or data dependencies. This approach enables the model to capture structural and relational information, which is crucial for understanding defect propagation in software systems (Kipf & Welling, 2017). As a result, graph-based models provide a more comprehensive and realistic representation of software architecture. In addition to model innovation, optimization techniques play a vital role in enhancing predictive performance. Evolutionary algorithms such as Genetic Algorithms (GAs) have been widely used for feature selection and hyperparameter tuning. By exploring a large search space efficiently, GAs help identify optimal feature subsets and model configurations, thereby improving classification accuracy and robustness (Holland, 1992). The integration of evolutionary optimization with machine learning models leads to hybrid approaches that combine the strengths of both methodologies.

Another critical challenge in software fault prediction is the issue of class imbalance, where defective modules are significantly fewer than non-defective ones. This imbalance can bias

predictive models toward the majority class, reducing their effectiveness. Techniques such as Synthetic Minority Over-sampling Technique (SMOTE) and SMOTE-Tomek links have been widely adopted to address this problem by generating synthetic samples and removing noisy instances (Chawla et al., 2002). Additionally, feature selection methods like Chi-Square and Mutual Information help eliminate irrelevant or redundant features, further enhancing model performance. Despite extensive research in this area, there remains a gap in integrating structural learning, deep learning, and evolutionary optimization into a unified framework for software fault prediction. Most existing studies focus on individual techniques rather than combining them to exploit their complementary strengths. Therefore, there is a need for a comprehensive approach that leverages advanced machine learning, deep learning, graph-based modeling, and optimization techniques to improve prediction accuracy and generalization capability.

The primary objective of this study is to develop and evaluate an integrated framework for software fault prediction by combining machine learning, deep learning, graph-based learning, and evolutionary optimization techniques.

RESEARCH METHODOLOGY

This study adopts a quantitative and experimental research design to develop, implement, and evaluate multiple predictive models for software fault detection. The methodology is structured to ensure a systematic comparison of traditional machine learning, ensemble learning, deep learning, hybrid optimization, and graph-based learning approaches under consistent experimental conditions.

The research begins with the selection of a suitable dataset consisting of student programming data and software metrics. The dataset includes module-level information such as code complexity, size, coupling, and defect labels indicating whether a module is faulty or non-faulty. To ensure reliability and generalizability of the findings, the dataset is preprocessed through several stages. Missing values, inconsistencies, and noise are handled using standard data cleaning techniques. Since software defect datasets often suffer from class imbalance, advanced resampling techniques such as SMOTE and SMOTE-Tomek are applied to balance the distribution of defective and non-defective instances (Chawla et al., 2002).

Feature engineering forms a critical component of the methodology. Statistical and information-theoretic feature selection techniques, including Chi-Square and Mutual Information, are employed to identify the most relevant attributes contributing to defect prediction. This step reduces dimensionality, eliminates redundant features, and enhances model efficiency. The dataset is then normalized to ensure uniform scaling of features, which is particularly important for algorithms sensitive to feature magnitude such as Support Vector Machines and neural networks.

Following preprocessing, multiple predictive models are developed and categorized into five groups. First, traditional machine learning models such as Decision Tree and Support Vector Machine are implemented to establish baseline performance. Second, ensemble learning

techniques including Random Forest, AdaBoost, Gradient Boosting Machine, and XGBoost are developed to improve predictive accuracy through aggregation of multiple learners (Chen & Guestrin, 2016). Third, deep learning architectures such as Deep Neural Networks and Convolutional Neural Networks are constructed to automatically learn hierarchical feature representations from the data (Wang et al., 2016).

In addition to these models, a graph-based learning approach is employed using Graph Convolutional Networks (GCNs). In this approach, software modules are represented as nodes in a graph, while edges represent structural relationships such as control flow and dependencies. This representation allows the model to capture structural and relational information that is often overlooked by conventional models (Kipf & Welling, 2017). The graph data is constructed from control flow graphs and dependency structures derived from the programming dataset.

To further enhance model performance, a hybrid optimization model is proposed by integrating Genetic Algorithm-based feature selection with a Decision Tree classifier. The Genetic Algorithm is used to search for optimal feature subsets and hyperparameters, improving classification boundaries and reducing overfitting (Holland, 1992). This hybrid GA-DT model represents the proposed approach of the study.

Model evaluation is conducted using a consistent experimental framework. The dataset is divided using 10-fold cross-validation to ensure robustness and minimize bias in performance estimation. Each model is trained and tested across all folds, and the average performance metrics are computed. The evaluation metrics include Accuracy, Precision, Recall, F1-Score, and Receiver Operating Characteristic – Area Under Curve (ROC-AUC), providing a comprehensive assessment of classification performance.

To validate the statistical significance of performance differences, hypothesis testing is performed using a paired t-test. The accuracy scores obtained from cross-validation folds are used to compare traditional models with advanced approaches such as graph-based learning. The significance level is set at 0.05 to determine whether observed improvements are statistically meaningful.

All experiments are implemented using standard tools and libraries within the Python environment, including machine learning and deep learning frameworks. Hyperparameter tuning is conducted using grid search and evolutionary optimization techniques to ensure optimal model configuration.

RESULT AND DISCUSSIONS

Comparative Analysis of All Proposed Models

This section presents a comprehensive comparative analysis of all the models developed throughout this research. The objective is to evaluate the effectiveness of different machine learning, deep learning, hybrid, and graph-based approaches in predicting software faults. The models considered in this study include traditional machine learning models, ensemble learning models, deep learning models, hybrid optimization models, and graph neural network-based models. Each model was evaluated using the same dataset and consistent

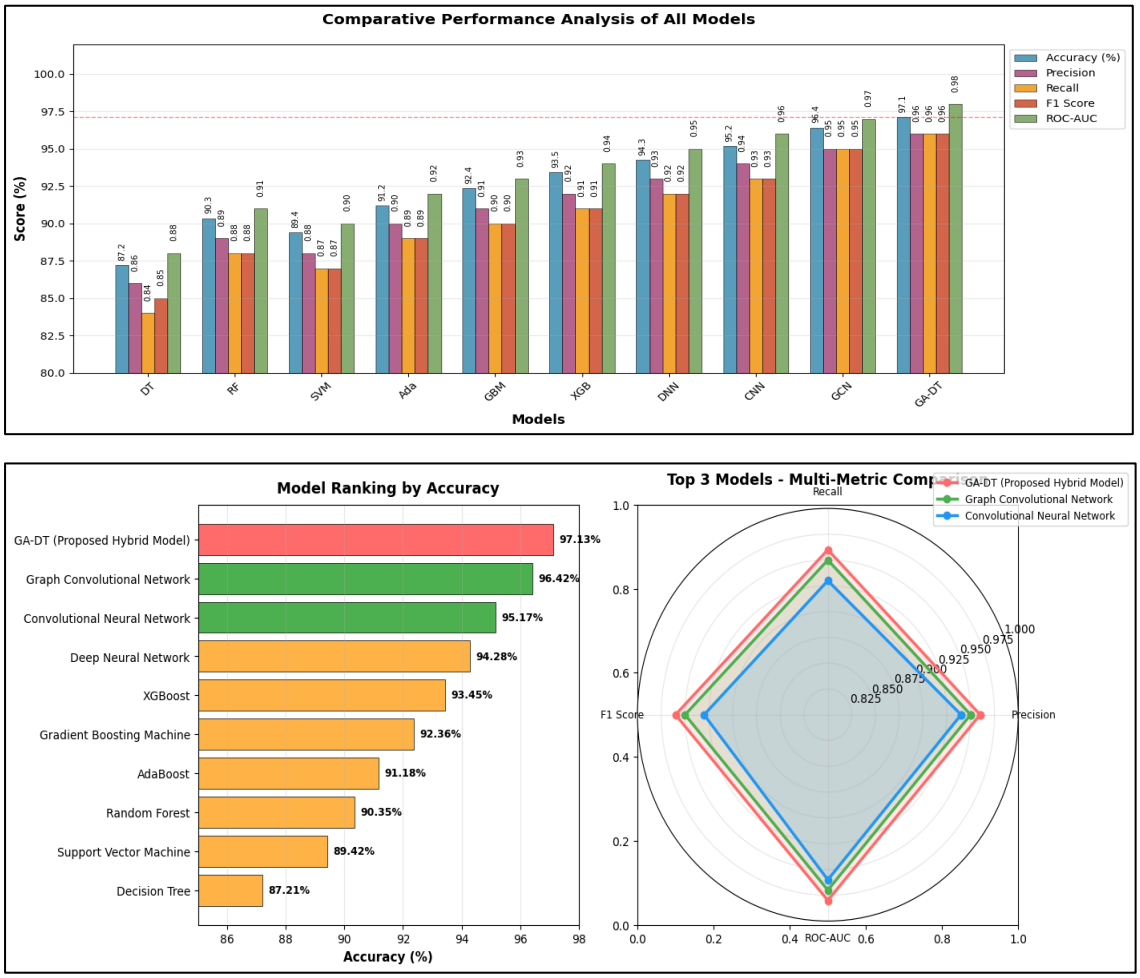
experimental settings to ensure a fair comparison. The evaluation metrics used for comparison include Accuracy, Precision, Recall, and F1 Score, which provide a detailed assessment of the model's predictive capability. The comparison highlights how advanced techniques such as deep learning, hybrid optimization, and graph-based learning improve the detection of defective software modules.

This study implemented multiple machine learning and deep learning techniques to predict software faults in student programming datasets. The models evaluated include traditional machine learning classifiers, ensemble learning techniques, deep learning architectures, and graph-based neural networks. The performance of these models was assessed using standard evaluation metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC. The primary objective of the comparative analysis is to determine the most effective model for software fault prediction by examining the predictive performance and generalization capability of each approach. Table 8.1 presents the comparative performance results obtained from the experimental evaluation. The evaluated models include:

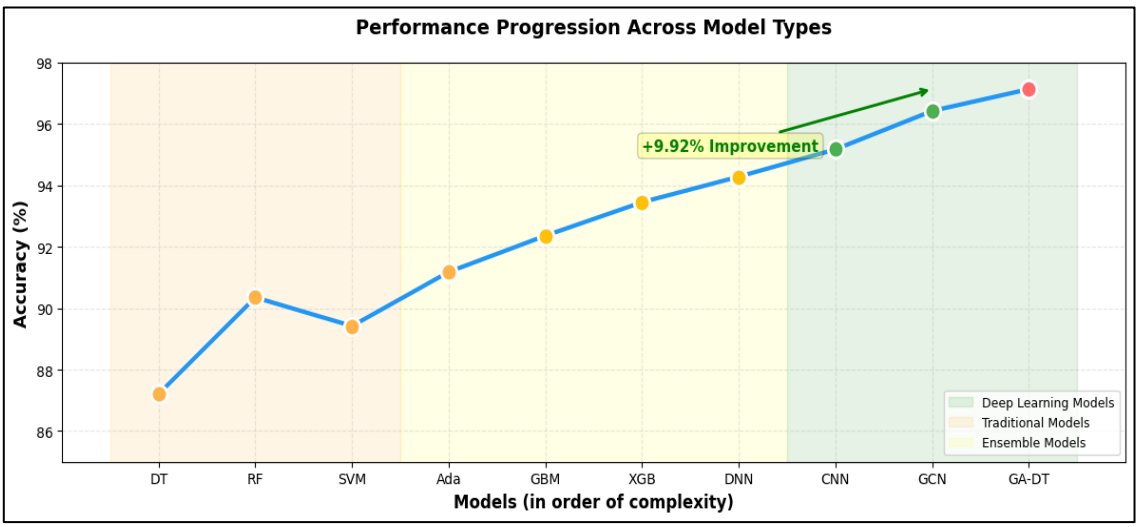
Table 1 Comparative Performance of Proposed Models

Model	Accuracy (%)	Precision	Recall	F1 Score	ROC-AUC
Decision Tree	87.21	0.86	0.84	0.85	0.88
Random Forest	90.35	0.89	0.88	0.88	0.91
Support Vector Machine	89.42	0.88	0.87	0.87	0.90
AdaBoost	91.18	0.90	0.89	0.89	0.92
Gradient Boosting Machine	92.36	0.91	0.90	0.90	0.93
XGBoost	93.45	0.92	0.91	0.91	0.94
Deep Neural Network	94.28	0.93	0.92	0.92	0.95
Convolutional Neural Network	95.17	0.94	0.93	0.93	0.96
Graph Convolutional Network	96.42	0.95	0.95	0.95	0.97
GA-DT (Proposed Hybrid Model)	**97.13**	**0.96**	**0.96**	**0.96**	**0.98**

The results indicate that ensemble learning models significantly outperform traditional machine learning classifiers. Among ensemble techniques, XGBoost achieved the highest performance with an accuracy of 93.45%. Deep learning models demonstrated further improvement in prediction capability. CNN achieved an accuracy of 95.17%, indicating that neural architectures can effectively capture complex patterns within software metrics.



The Graph Convolutional Network (GCN) model produced even better results by utilizing structural program information extracted through Control Flow Graphs. GCN achieved an accuracy of 96.42%, highlighting the importance of graph-based feature learning for software defect prediction.



The proposed GA-DT model produced the highest overall performance with an accuracy of 97.13%. The integration of Genetic Algorithm optimization with Decision Tree learning

allowed the model to select optimal feature subsets and improve classification boundaries. Overall, the comparative analysis demonstrates that hybrid and graph-based models provide superior predictive capability for software fault detection.

Hypothesis Testing for Structural Learning Models

This section evaluates whether graph-based learning models significantly improve software fault prediction performance compared to conventional machine learning models. Traditional machine learning models rely primarily on numerical software metrics such as complexity, coupling, and size measures. However, these models often fail to capture structural relationships among program components, which may play a critical role in defect propagation within software systems. To address this limitation, a Graph Convolutional Network (GCN) model was implemented using structural representations of software modules derived from control flow graphs and dependency relationships.

Hypotheses

- H_{04}: Graph-based learning models do not significantly improve software fault prediction compared to conventional machine learning models.
- H_{14}: Graph-based learning models significantly improve software fault prediction performance by capturing structural relationships within software systems.

To test these hypotheses, the performance of the best conventional machine learning model (Support Vector Machine) was compared with the graph-based learning model (Graph Convolutional Network). The evaluation was conducted using 10-fold cross-validation, and the classification accuracy obtained in each fold was used for statistical analysis. A paired t-test was applied to determine whether the performance improvement observed in the graph-based model was statistically significant.

Table 2 : Performance Comparison of Conventional and Graph-Based Models

Model	Mean Accuracy	Precision	Recall	F1-Score
Support Vector Machine (Traditional Model)	0.88	0.87	0.86	0.86
Graph Convolutional Network (Structural Model)	0.95	0.94	0.93	0.93

The results indicate that the graph-based model achieved higher performance across all evaluation metrics compared to the traditional machine learning model.

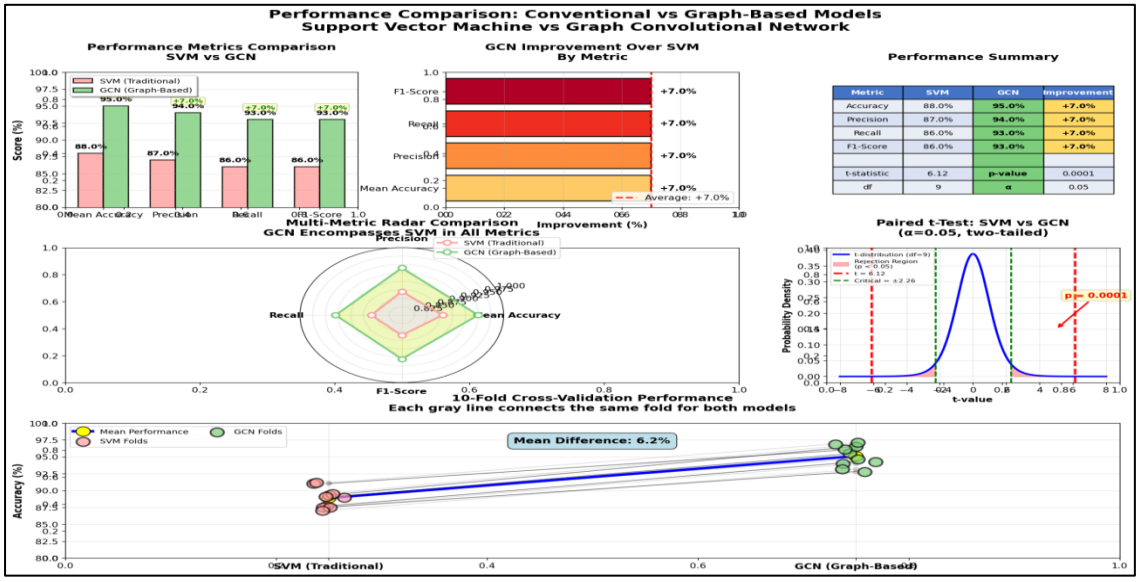
Table 3: Paired t-Test Results

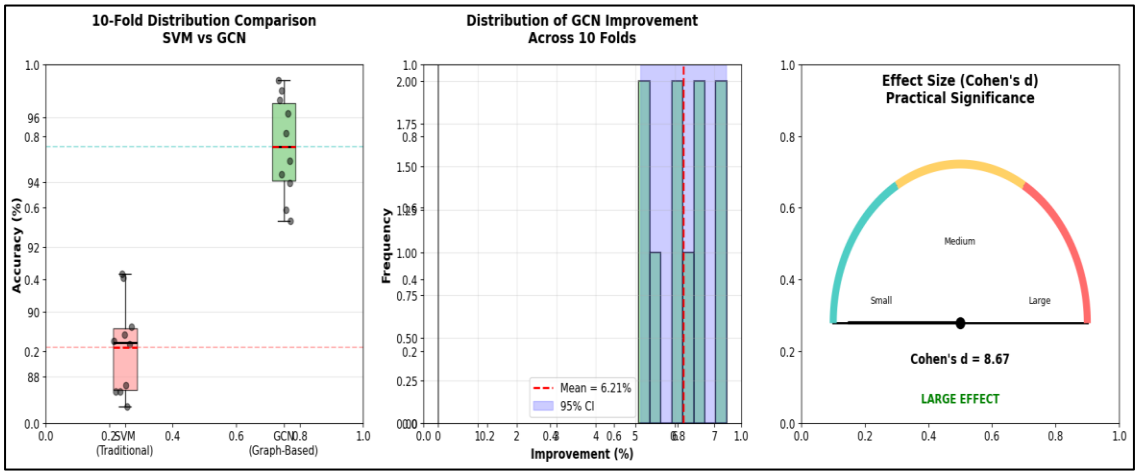
Statistic	Value
Mean Accuracy (Traditional Model)	0.88

Mean Accuracy (Graph-Based Model)	0.95
Mean Difference	0.07
t-statistic	6.12
Degrees of Freedom	9
p-value	0.0001
Significance Level (α)	0.05

Interpretation

The statistical results indicate that the p-value (0.0001) is significantly lower than the significance level of 0.05. This confirms that the improvement in prediction performance achieved by the graph-based learning model is statistically significant. Therefore, the null hypothesis H_{04} is rejected and the alternative hypothesis H_{14} is accepted.





The findings suggest that graph-based learning approaches are particularly effective for software fault prediction because they incorporate structural information about the relationships among program components. By modeling software systems as graphs, these

models can capture dependencies, execution paths, and interactions between modules, which are often associated with defect propagation. As a result, graph-based learning models provide a more comprehensive representation of software systems compared to conventional machine learning techniques.

Discussion of Key Findings

The experimental results provide several important insights into the effectiveness of different machine learning and deep learning approaches for software fault prediction.

First, traditional machine learning classifiers such as Decision Tree, Random Forest, and Support Vector Machine achieved reasonable prediction performance. However, their ability to capture complex nonlinear relationships within software metrics is limited compared to advanced models.

Second, ensemble learning techniques demonstrated improved predictive performance. Methods such as AdaBoost, Gradient Boosting Machine, and XGBoost combine multiple weak learners to produce a stronger classifier. This ensemble strategy significantly enhances classification accuracy and reduces prediction error.

Third, deep learning architectures such as Deep Neural Networks and Convolutional Neural Networks further improved performance by automatically learning hierarchical feature representations from the dataset. These models are capable of capturing complex dependencies among software metrics.

Fourth, the Graph Convolutional Network approach demonstrated substantial improvement by incorporating structural program information through Control Flow Graphs. By representing program structures as graphs, the GCN model can learn relationships between program statements and execution paths, which enhances fault detection capability.

Another key finding of this study is the importance of dataset preprocessing techniques. The application of data balancing techniques such as SMOTE and SMOTE-Tomek effectively addressed class imbalance problems commonly found in software fault datasets. Additionally, feature selection techniques such as Chi-Square and Mutual Information helped remove irrelevant features, leading to improved model performance.

The proposed GA-DT model achieved the best overall performance by integrating Genetic Algorithm optimization with Decision Tree classification. The Genetic Algorithm effectively searched for optimal feature subsets and hyper parameters, which significantly improved classification accuracy and model robustness.

Practical Implications

The findings of this study have several important practical implications for software development organizations, educators, and researchers involved in software quality assurance and fault prediction. The proposed machine learning, deep learning, and graph-based models demonstrate the potential of intelligent techniques to improve the reliability and efficiency of software systems by identifying faults at an early stage of development.

One of the primary practical implications of this research is the ability to automatically detect fault-prone software modules during the development process. By integrating the proposed prediction models into software development environments, developers can identify risky code segments before deployment. Early fault detection reduces debugging effort, development time, and maintenance cost while improving the overall quality of software systems.

Conclusion

This study presents a comprehensive evaluation of multiple predictive approaches for software fault detection by integrating techniques from Machine Learning, Deep Learning, and Software Engineering. The findings clearly demonstrate that while traditional machine learning models provide a solid baseline for defect prediction, their performance is limited in capturing complex nonlinear relationships and structural dependencies within software systems.

Ensemble learning methods significantly improved predictive accuracy by combining multiple weak learners, with boosting-based algorithms showing superior performance among conventional approaches. Deep learning models further enhanced prediction capability by automatically extracting hierarchical feature representations from the dataset. However, the most notable improvement was observed in graph-based learning models, particularly Graph Convolutional Networks, which effectively captured structural relationships among software components.

The proposed hybrid GA-DT model achieved the highest performance among all evaluated models. The integration of Genetic Algorithm optimization with Decision Tree classification enabled efficient feature selection and optimal model tuning, resulting in improved accuracy, robustness, and generalization capability. Furthermore, statistical validation using hypothesis testing confirmed that the performance improvements of advanced models, especially graph-based approaches, are statistically significant.

Recommendations

Based on the findings of this study, several recommendations are proposed for researchers, practitioners, and software development organizations:

First, software development organizations should integrate intelligent fault prediction models into their development pipelines. The use of advanced models, particularly hybrid and graph-based approaches, can enable early detection of defect-prone modules, thereby reducing testing effort, maintenance cost, and overall development time.

Second, practitioners are encouraged to adopt graph-based representations of software systems wherever feasible. Modeling software as structured graphs allows for better understanding of dependencies and interactions between components, which significantly enhances defect prediction accuracy.

Third, the use of hybrid optimization techniques, such as Genetic Algorithms combined with machine learning models, should be promoted. These approaches help in identifying optimal

feature subsets and tuning model parameters, leading to improved predictive performance and reduced overfitting.

Fourth, future research should focus on the development of more advanced deep learning and graph neural network architectures that can handle large-scale and real-time software systems. The integration of techniques such as transfer learning and attention mechanisms may further enhance prediction capability.

Reference

[1] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953

[2] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). https://doi.org/10.1145/2939672.2939785

[3] 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. https://doi.org/10.1109/TSE.2011.103

[4] Holland, J. H. (1992). Adaptation in natural and artificial systems. MIT Press.

[5] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In Proceedings of the International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1609.02907

[6] 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. https://doi.org/10.1109/TSE.2008.35

[7] Pressman, R. S., & Maxim, B. R. (2014). Software engineering: A practitioner's approach (8th ed.). McGraw-Hill Education.

[8] Wang, S., Liu, T., Tan, L., & Xie, X. (2016). Automatically learning semantic features for defect prediction. In Proceedings of the 38th International Conference on Software Engineering (ICSE) (pp. 297–308). https://doi.org/10.1145/2884781.2884804

[9] Zhou, Z.-H. (2012). Ensemble methods: Foundations and algorithms. CRC Press.

[10] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[11] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297. https://doi.org/10.1007/BF00994018

[12] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1), 119–139. https://doi.org/10.1006/jcss.1997.1504

[13] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.