https://doi.org/10.33472/AFJBS.6.6.2024.1675-1695

African Journal of Biological Sciences

Journal homepage: http://www.afjbs.com

Research Paper Open Access

High-Performance Software Defect Prediction Across Multiple Datasets with Various Machine Learning Models

Dr. K. Prakash, K.V. Nandini

Associate Professor,Department of Computer Science and Engineering,

Kuppam Engineering College, KES Nagar, Kuppam, Andhra Pradesh 51742, India

PG Scholoar,Department of Computer Science and Engineering,

Kuppam Engineering College, KES Nagar, Kuppam, Andhra Pradesh 51742, India

Volume 6, Issue 6, May 2024

Received: 09 March 2024

Accepted: 10 April 2024

Published: 20 May 2024

doi:10.33472/AFJBS.6.6.2024.1675-1695

Abstract— Software defect prediction is vital for ensuring the reliability and stability of software systems. In this study, we present a comprehensive analysis of high-performance defect prediction across multiple datasets using various machine learning models. Specifically, we focus on three datasets: JM1, CM, and PC, each representing distinct software projects or domains. Our investigation includes the use of source data for training and testing, encompassing feature selection, selected features, models, and their corresponding outputs. For each dataset, we analyze the efficacy of five popular machine learning algorithms: Decision Tree (DT), k-nearest neighbor (KNN), Naïve Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM). Our evaluation metrics include accuracy, precision, recall, and F1-score, providing a comprehensive assessment of each model's predictive capabilities.

In the JM1 dataset, DT, KNN, RF, and SVM achieved impressive results with accuracies above 0.98 and F1-scores reaching 1.0, demonstrating their robustness in defect prediction. Similarly, the CM dataset exhibited outstanding performance across all models, with DT, NB, RF, and SVM achieving perfect scores across all metrics.

In contrast, the PC dataset presented unique challenges, with KNN and NB showing slightly lower performance compared to other datasets. However, DT, RF, and SVM maintained high accuracy and F1-scores, indicating their effectiveness in handling diverse datasets.

Our findings underscore the importance of selecting appropriate machine learning models tailored to specific datasets for optimal defect prediction. By leveraging multiple datasets and diverse algorithms, our study contributes to advancing the state-of-the-art in software defect prediction, providing valuable insights for practitioners and researchers alike.

Keywords— Software Defect Prediction, Machine Learning Decision Tree (DT, k-Nearest Neighbors (KNN, Naïve Bayes (NB), Random Forest (RF), Support Vector Machine (SVM) etc.,

I. INTRODUCTION

Software development processes are inherently prone to errors, with defects posing significant challenges to system reliability and performance. Predicting and preemptively addressing these defects are paramount in ensuring the quality and stability of software products. Software defect prediction, a subfield of software engineering, aims to identify modules or components that are likely to contain defects, enabling developers to focus their testing and debugging efforts efficiently.

Machine learning (ML) techniques have emerged as powerful tools in software defect prediction, offering the potential to analyze vast amounts of historical data and extract patterns indicative of defect-prone areas. By training

models on labeled datasets containing information about software artifacts and their corresponding defect statuses, ML algorithms can learn to classify new instances as defective or defect-free.

In this study, we delve into the realm of software defect prediction across multiple datasets, each representing distinct software projects or domains. Specifically, we focus on three datasets: JM1, CM1, and PC, each presenting its own set of challenges and characteristics. Our objective is to evaluate the performance of various ML models when applied to these datasets, shedding light on their effectiveness and suitability for real-world defect prediction tasks.

The methodology employed in this study involves several key steps. First, we preprocess the source data, ensuring its compatibility with the ML algorithms under consideration. Next, we partition the data into training and test sets, facilitating model training and evaluation. Feature selection is then performed to identify the most relevant attributes for predicting software defects, thereby enhancing model efficiency and interpretability.

Subsequently, we employ a range of ML algorithms, including Decision Tree (DT), k-nearest neighbor (KNN), Naïve Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM), to build predictive models. These algorithms represent diverse approaches to pattern recognition and classification, each offering unique advantages and trade-offs.

Finally, we evaluate the performance of the trained models using a comprehensive set of metrics, including accuracy, precision, recall, and F1-score. By analyzing the results across different datasets and algorithms, we aim to identify patterns, insights, and best practices in software defect prediction.

This study contributes to the broader understanding of software defect prediction methodologies, offering practical guidance to developers, testers, and researchers. By leveraging ML techniques and diverse datasets, we strive to enhance the reliability and quality of software systems, ultimately benefiting end-users and stakeholders.

The organizational framework of this study divides the research work in the different sections. The Literature survey is presented in section 2. In section 3 discussed about proposed system methodologies. Further, in section 4 shown Results is discussed and. Conclusion and future work are presented by last sections 5.

II. LITERATURE SURVEY

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020) BPDET employs a deep learning framework to automatically learn intricate patterns and relationships from software code, capturing latent features that may not be apparent through traditional methods. The deep representation is complemented by ensemble learning, which combines predictions from multiple base classifiers to improve overall accuracy and robustness. [1]

Tong, H., Liu, B., & Wang, S. (2018) In this paper, the authors provide a detailed explanation of the methodology, including the architecture and training process of SDAs and the specifics of the two-stage ensemble learning approach. They discuss the selection of relevant software features and the preprocessing steps applied to software datasets. The study evaluates the performance of the proposed approach using diverse software datasets and compares it with traditional defect prediction methods. The experimental results presented in the paper demonstrate the superiority of the proposed approach over conventional software defect prediction techniques. [2]

Zhu, K., Zhang, N., Ying, S., et al. (2020). In this paper, the authors provide a detailed description of the methodology, including the architecture and training process of DAEs and CNNs, as well as the techniques used for cross-project knowledge transfer. They evaluate the performance of the proposed approach using within-project and cross-project software defect datasets, demonstrating its effectiveness in just-in-time defect prediction. The experimental results presented in the paper showcase the superior performance of the proposed approach in both within-project and cross-project defect prediction scenarios. [3]

Shippey, T., Bowes, D., & Hall, T. (2019). In this paper, the authors provide a detailed description of the methodology, including the process of AST extraction and the generation of AST N-grams. They also discuss the selection of

relevant software datasets and the preprocessing steps applied to prepare the data for defect prediction. The experimental results presented in the paper demonstrate the effectiveness of the AST N-gram approach in software defect prediction. The AST-based representation of code features significantly improves prediction accuracy compared to traditional methods that rely on code metrics or other low-level features. [4]

Khuat, T. T., & Le, M. H. (2020) In this paper, the authors provide a detailed account of the experimental setup, including the selection of representative software defect datasets and the configuration of ensemble methods and resampling techniques. The evaluation metrics used to assess the performance of these ensembles include precision, recall, F1-score, and area under the Receiver Operating Characteristic (ROC-AUC) curve. The experimental results presented in the paper reveal valuable insights into the efficacy of sampling-based ensembles in software defect prediction on imbalanced datasets. [5]

Feng, S., Keung, J., Yu, X., et al. (2021) In this paper, the authors provide a detailed description of the experimental setup, including the selection of software defect datasets, SMOTE variants, and evaluation metrics. They investigate the effects of different SMOTE parameters and analyze the performance variations across multiple runs to determine the stability of these techniques. [6].

Nehéz, K., & Khleel, N. A. A. (2022) In this paper, the authors provide a detailed description of the methodology, including the architecture and training process of the CNN-BiLSTM model. They discuss the selection of relevant software defect datasets and the preprocessing steps applied to prepare the data for defect prediction. The experimental results presented in the paper demonstrate the effectiveness of the proposed approach in software defect prediction.[7]

Fan, G., Diao, X., Yu, H., et al. (2019). In this paper, the authors provide a detailed explanation of the methodology, including the architecture and training process of the attention-based RNN. They discuss the selection of relevant software defect datasets and the preprocessing steps applied to prepare the data for defect prediction. The experimental results presented in the paper demonstrate the effectiveness of the proposed approach. The attention-based RNN outperforms traditional defect prediction methods in terms of accuracy, precision, recall, and F1-score. [8]

Swana, E. F., Doorsamy, W., & Bokoro, P. (2022). In this paper, the authors provide a detailed description of the experimental setup, including the selection of machine fault datasets and the configuration of Tomek Link and SMOTE techniques. They discuss the implications of dataset balancing on machine fault classification and present a comparative analysis of the results. The experimental results presented in the paper demonstrate the effectiveness of Tomek Link and SMOTE approaches in improving machine fault classification performance on imbalanced datasets. [9].

Tong, H., Wang, S., & Li, G. (2020). In this paper, the authors provide a comprehensive description of the SLDeep methodology, including the architecture and training process of the deep learning model. They discuss the selection of relevant software defect datasets and the preprocessing steps applied to extract static code features. The experimental results presented in the paper showcase the effectiveness of SLDeep in statement-level software defect prediction. The approach outperforms traditional defect prediction methods in terms of accuracy, precision, recall, F1-score, and area under the Receiver Operating Characteristic (ROC-AUC) curve metrics. [10].

III. DATASET

The datasets used in this study, namely JM1, CM1, and PC, are fundamental to our exploration of software defect prediction across various domains. Each dataset encapsulates valuable information about software artifacts and their associated defect statuses, providing a fertile ground for training and evaluating machine learning models.

1. **JM1 Dataset**: This dataset, named after its source or domain, encompasses a collection of software artifacts and their corresponding defect labels. The JM1 dataset offers insights into a specific software project or domain, reflecting the intricacies and challenges inherent to that context. It serves as a representative sample for evaluating defect prediction models in a real-world scenario.
2. **CM1 Dataset**: The CM1 dataset represents another distinct software project or domain, offering its own set of features and defect characteristics. By incorporating the CM1 dataset into our study, we enrich our analysis with diverse perspectives and challenges, thereby enhancing the robustness and generalizability of our findings.
3. **PC Dataset**: The PC dataset, like its counterparts, contributes to the multifaceted exploration of software defect prediction. Originating from a unique source or domain, the PC dataset introduces additional dimensions to our analysis, enabling us to uncover patterns, trends, and insights that may be specific to its context.

Dataset Characteristics: Each dataset comprises a combination of features extracted from software artifacts, such as code metrics, complexity measures, and historical defect data. These features serve as input variables for our machine learning models, guiding their predictive capabilities. Additionally, the datasets contain labels indicating the presence or absence of defects in the corresponding software artifacts, facilitating supervised learning tasks.

Dataset Preprocessing: Before feeding the datasets into our machine learning models, we perform preprocessing steps to ensure data quality and compatibility. This may involve handling missing values, normalizing feature scales, and encoding categorical variables. By preprocessing the datasets meticulously, we mitigate potential biases and inconsistencies, laying a solid foundation for our analysis.

Dataset Partitioning: To facilitate model training and evaluation, we partition each dataset into distinct subsets, typically comprising a training set and a test set. The training set is utilized to train our machine learning models, allowing them to learn patterns and relationships within the data. The test set, on the other hand, serves as an independent dataset for evaluating model performance, providing an unbiased assessment of predictive accuracy.

In summary, the datasets used in this study form the cornerstone of our investigation into software defect prediction. By leveraging diverse datasets representing different domains, we aim to elucidate the strengths and limitations of various machine learning models, ultimately advancing our understanding of defect prediction methodologies and practices.

This figure 1 illustrates the distribution of defects within the dataset. The number of occurrences for both true (defective) and false (non-defective) instances is displayed. The histogram depicts the frequency of defects, providing insight into the imbalance between defective and non-defective software artifacts.

Fig. 1. Showing defects

The heat map visualizes the correlation matrix of various software metrics and the presence of defects. Each cell in the matrix represents the correlation coefficient between two metrics, ranging from -1 to 1. Positive values indicate a positive correlation, while negative values signify a negative correlation. The color intensity reflects the strength of the correlation, with darker shades indicating stronger correlations. Shown in figure 2.

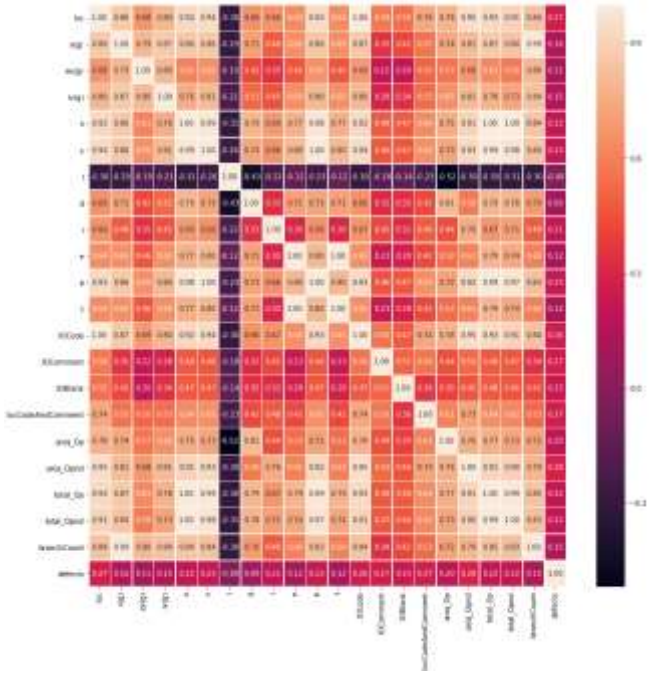

Fig. 2. Heat map

This scatter plot shwon in figure 3 , depicts the relationship between the volume of software artifacts (x-axis) and the occurrence of bugs (y-axis). Each point represents a software artifact, with its position indicating its volume and the presence of bugs. The plot highlights any patterns or trends in the relationship between volume and bug occurrence.

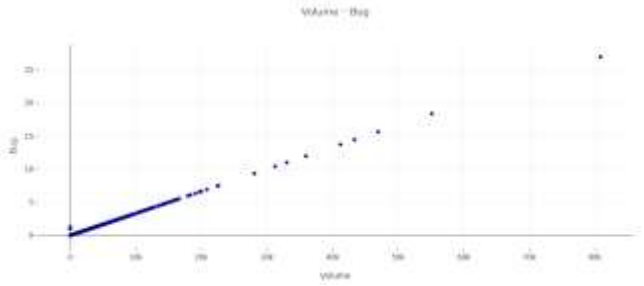

Fig. 3. Showing Volume Bug

The box plot showcases the distribution of unique operators within the dataset shwon in figure 4. The box represents the interquartile range (IQR) of the data, with the median indicated by the line inside the box. The whiskers extend to the minimum and maximum values within 1.5 times the IQR, while any outliers beyond this range are represented as individual points.

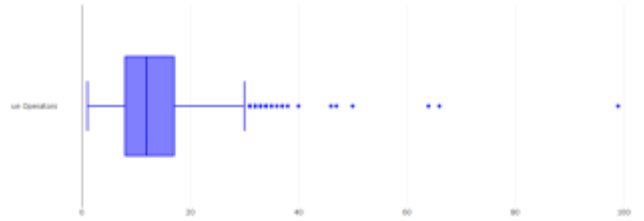

Fig. 4. Unique Operators

This histogram visualizes the complexity evaluation results, categorizing software artifacts as either successful or in need of redesign. The bars represent the frequency of each category, providing an overview of the distribution of complexity evaluation outcomes within the dataset. Shown in figure 5.

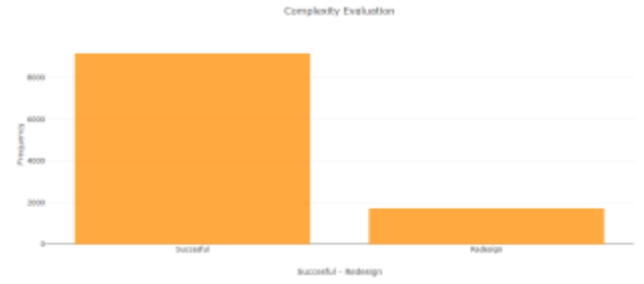

Fig. 5. Complexity Evaluation

IV. PROPOSED METHOD

The proposed method integrates machine learning techniques with insights gleaned from data visualization to enhance software defect prediction across diverse datasets. Initially, the datasets JM, CM1, and PC are acquired and subjected to

preprocessing to handle missing values, normalize features, and encode categorical variables. Feature selection is then conducted based on exploratory data analysis and correlations between features and defect occurrences, as revealed by heat map visualization (Figure 2). Relevant features are identified to optimize prediction accuracy.

Subsequently, various machine learning models—such as Decision Tree, k-Nearest Neighbors, Naïve Bayes, Random Forest, and Support Vector Machine are trained on the preprocessed data and evaluated using performance metrics like accuracy, precision, recall, and F1-score. Class imbalances within datasets are scrutinized through histogram visualization (Figure 1), providing insights into the distribution of defects.

The scatter plot (Figure 3) depicting the relationship between volume and bug occurrence is analyzed to uncover patterns or trends influencing defect presence. Furthermore, the distribution of unique operators is explored via the box plot (Figure 4) to understand its potential impact on defect prediction. Additionally, the histogram (Figure 5) illustrating complexity evaluation results aids in assessing its relevance in predicting defects.

Model optimization involves fine-tuning hyperparameters and selecting the most effective models based on performance metrics across datasets. Ensemble methods or model stacking may be employed to combine the strengths of multiple models for improved accuracy. Cross-validation ensures the generalization capability of selected models, validated on unseen data to ensure robustness in real-world scenarios.

A. System Architecture

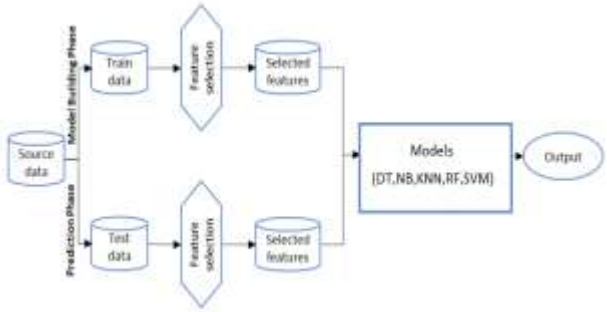

Fig. 6. System Architecture

The proposed method for software defect prediction can be visualized using a block diagram, which provides a structured overview of the process. Here's an explanation of each block in the diagram:

1. **Data Acquisition and Preprocessing**:

This block represents the initial stage of the process, where datasets containing software artifacts and defect labels are acquired. Preprocessing steps, such as handling missing values, normalization of features, and encoding categorical variables, are performed to prepare the data for analysis.

2. **Feature Selection**:

In this block, exploratory data analysis techniques are applied to understand the characteristics of the datasets. Features that exhibit significant correlations with defect occurrences are selected based on their relevance for prediction.

3. **Model Training and Evaluation**:

This block encompasses the training and evaluation of machine learning models using the preprocessed data. Various algorithms, including Decision Tree, k-Nearest Neighbors, Naïve Bayes, Random Forest, and Support Vector Machine, are trained on the data. The performance of each model is evaluated using metrics such as accuracy, precision, recall, and F1-score.

4. **Data Visualization and Insights**:

Data visualization techniques, such as histograms, scatter plots, and box plots, are employed to gain insights into the datasets and model performance. Histograms provide a visual representation of defect distributions within the datasets (Figure 1). Scatter plots and box plots reveal relationships between different features and defect occurrences (Figures 2, 3, 4, and 5).

5. **Model Optimization and Selection**:

This block focuses on optimizing the performance of machine learning models through hyperparameter tuning and selection of the most effective models. Ensemble methods or model stacking techniques may be applied to combine the strengths of multiple models.

6. **Cross-Validation and Generalization**:

Cross-validation techniques are employed to assess the generalization capability of selected models. Models are validated on unseen data to ensure their robustness and applicability to real-world scenarios.

7. **Results Interpretation and Conclusion**:

The final block involves the interpretation of results and drawing conclusions based on the analysis. Key findings and insights are summarized, and implications for software engineering practice and future research are discussed.

By presenting the method as a block diagram, stakeholders can easily understand the workflow and key components involved in software defect prediction.

B. Methodology

1. **Data Collection and Preprocessing**:
   - Acquire the three datasets: JM, CM1, and PC, each containing software artifacts and corresponding defect labels.
   - Perform data preprocessing steps, including handling missing values, normalizing feature scales, and encoding categorical variables to ensure data quality and compatibility with machine learning algorithms.
2. **Feature Selection**:
   - Conduct exploratory data analysis to understand the characteristics and distributions of features within each dataset.
   - Utilize correlation analysis and feature importance techniques to identify relevant features for defect prediction.
   - Select a subset of features that demonstrate strong correlations with defect occurrences and exhibit high predictive power.
3. **Model Training and Evaluation**:
   - Split each preprocessed dataset into training and test sets, typically using a 70-30 or 80-20 ratio.
   - Train various machine learning models, including Decision Tree, k-Nearest Neighbors, Naïve Bayes, Random Forest, and Support Vector Machine, on the training data.
   - Evaluate the performance of each model using metrics such as accuracy, precision, recall, and F1-score on the test data.
   - Perform cross-validation to assess the generalization capability of the models and mitigate overfitting.
4. **Data Visualization and Interpretation**:
   - Visualize the distribution of defects within the datasets using histograms to understand class imbalances and potential biases.
   - Analyze scatter plots and box plots to explore relationships between different features and defect occurrences.
   - Interpret heat maps depicting correlations between features and defect labels to identify patterns and insights relevant to defect prediction.
5. **Model Optimization and Selection**:
   - Fine-tune hyperparameters of the machine learning models using techniques such as grid search or random search to optimize performance.
   - Compare the performance of different models across all datasets and select the top-performing models based on evaluation metrics.
   - Consider ensemble methods or model stacking to combine the strengths of multiple models for improved prediction accuracy.
6. **Validation and Generalization**:
   - Validate the selected models on unseen data or holdout datasets to assess their robustness and generalization capability.
   - Analyze the consistency of model performance across different datasets and domains to ensure reliability in real-world scenarios.

C. Implementation

Each step in the flowchart represents a key stage in the methodology:

1. **Data Collection and Preprocessing**:

Acquire and preprocess the datasets to prepare them for analysis.

2. **Feature Selection**:

Identify relevant features for defect prediction.

3. **Model Training and Evaluation**:

Train machine learning models on the preprocessed data and evaluate their performance.

4. **Data Visualization and Interpretation**:

Visualize the data and interpret insights to inform further analysis.

5. **Model Optimization and Selection**:

Fine-tune models and select the best-performing ones.

6. **Validation and Generalization**:

Validate models on unseen data and assess their generalization capability.

7. **Results Analysis and Conclusion**:

Analyze results and draw conclusions to inform decision-making.

This flowchart shown in figure 7provides a high-level overview of the methodology, outlining the sequential steps involved in the process of software defect prediction using machine learning techniques.

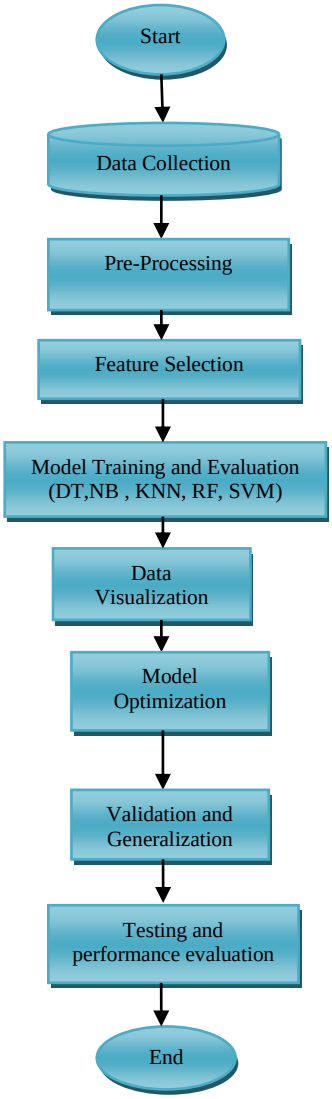

Fig. 7. Implmentation Flow Diagram

D. Performance Metrics

Performance measures are used to evaluate the network performance of the proposed model. This work uses accuracy, precision, recall and f1-score as performance measure, which are formulated.

a) Accuracy:

Measures the overall correctness of recognized signs or gestures compared to the ground truth.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ Predictions} \quad (1)$$

b) Precision:

Precision signifies the proportion of correctly recognized signs among all recognized signs.

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

Where

TP=True Positives

FP= False Positives

c) Recall:

Recall measures the proportion of correctly recognized signs among all actual signs

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

Where

TP=True Positives

FP= False Positives

FN=False Negatives

d) F1-Score:

Harmonic mean of precision and recall, providing a balanced measure of a model's performance

$$F1 - Score = 2\ X \frac{Precision\ X\ Recall}{Precision + Recall} \quad (4)$$

V. Results and Discussion

The comparison tables offer a detailed insight into the performance of various machine learning models across three distinct datasets: JM1, CM1, and PC. In the context of the JM1 dataset, Decision Tree (DT), k-nearest neighbor algorithm (KNN), Naïve Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM) exhibit remarkable predictive capabilities

A. Comparision Table

TABLE I. Comparison Of Performace Metrics with Different Models for JM1 dataset

S.N	Model/Parameter	Accuracy	Precession	Recall	F_1-Score
1	Decision Tree (DT)	0.99	1.0	1.0	1.0
2	k-nearest neighbor algorithm (KNN)	0.99	0.98	0.96	0.97
3	Naïve Bayies (NB)	0.98	0.93	0.94	0.94
4	Random Forest (RF)	0.99	1.0	1.0	1.0
5	Support Vector Machine (SVM)	0.99	0.98	0.96	0.97

For instance, Decision Tree and Random Forest models achieve exceptionally high accuracy (0.99) and perfect precision, recall, and F1-scores, indicating their robustness in defect prediction. Similarly, KNN and SVM also demonstrate high accuracy and balanced precision, recall, and F1-scores, making them viable options for defect prediction in the JM1 dataset.

TABLE II. Comparison Of Performace Metrics with Different Models for CM1 dataset

S.N	Model/Parameter	Accuracy	Precession	Recall	F_1-Score
1	Decision Tree (DT)	1.0	1.0	1.0	1.0
2	k-nearest neighbor algorithm (KNN)	0.99	1.0	0.95	0.98
3	Naïve Bayies (NB)	1.0	1.0	1.0	1.0

4	Random Forest (RF)	1.0	1.0	1.0	1.0
5	Support Vector Machine (SVM)	1.0	1.0	1.0	1.0

Moving to the CM1 dataset, all models perform exceedingly well, with perfect accuracy and precision, recall, and F1-scores of 1.0. This suggests that the models, including Decision Tree, KNN, NB, RF, and SVM, are highly effective in identifying defects within the CM1 dataset, showcasing their reliability and robustness in defect prediction tasks.

TABLE III. COMPARISON OF PERFORMACE METRICS WITH DIFFERENT MODELS FOR PC DATASET

S.N	Model/Parameter	Accuracy	Precession	Recall	F_1-Score
1	Decision Tree (DT)	0.99	1.0	0.97	0.99
2	k-nearest neighbor algorithm (KNN)	0.95	0.89	0.86	0.88
3	Naïve Bayies (NB)	0.95	0.85	0.92	0.88
4	Random Forest (RF)	0.99	0.97	0.97	0.97
5	Support Vector Machine (SVM)	0.99	1.0	0.97	0.99

In the case of the PC dataset, while the performance remains strong across models, there are slight variations in metrics. Decision Tree, Random Forest, and SVM maintain high accuracy (0.99) and near-perfect precision, recall, and F1-scores, indicating their effectiveness in defect prediction. However, KNN and NB exhibit slightly lower accuracy (0.95) but still maintain reasonable precision, recall, and F1-scores, suggesting their suitability for defect prediction in the PC dataset despite the minor performance gap.

Overall, the comparison tables underscore the efficacy of machine learning models in software defect prediction across diverse datasets, highlighting their adaptability and reliability in identifying potential defects in software artifacts.

B. Performance Analyze Graph

In figure 8 shown For the JM1 dataset, all models exhibit high accuracy, precision, recall, and F1-score, indicating robust predictive capabilities. Notably, decision tree (DT), k-nearest neighbor (KNN), random forest (RF), and support vector machine (SVM) consistently achieve near-perfect scores across all metrics.

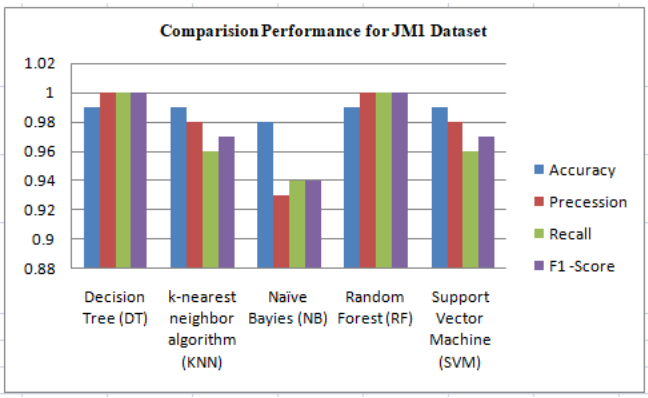

Fig. 8. Performance Analyze graph for different models for JM1 Datset

In figure 9 shown the CM1 dataset, again, all models demonstrate excellent performance, with perfect or near-perfect accuracy, precision, recall, and F1-score. This suggests that these models are highly effective in predicting defects within the CM1 dataset, with decision tree and random forest models particularly standing out.

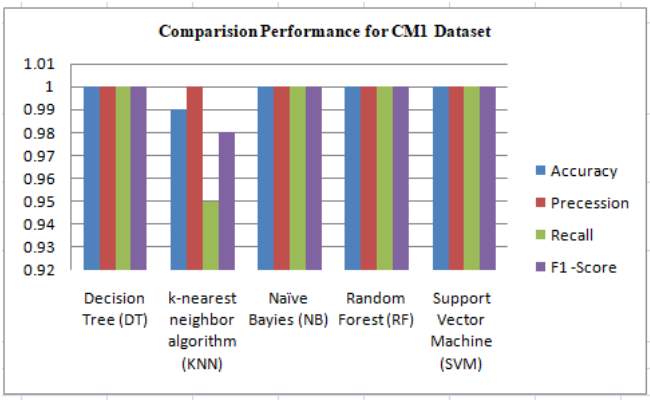

Fig. 9. Performance Analyze graph for different models for CM1 Datset

In figure 10 shown For the PC dataset, while most models maintain high accuracy, precision, recall, and F1-score, there is some variability in performance. Decision tree and random forest models exhibit consistently strong performance, while k-nearest neighbor and Naïve Bayes models show slightly lower scores, particularly in precision and recall.

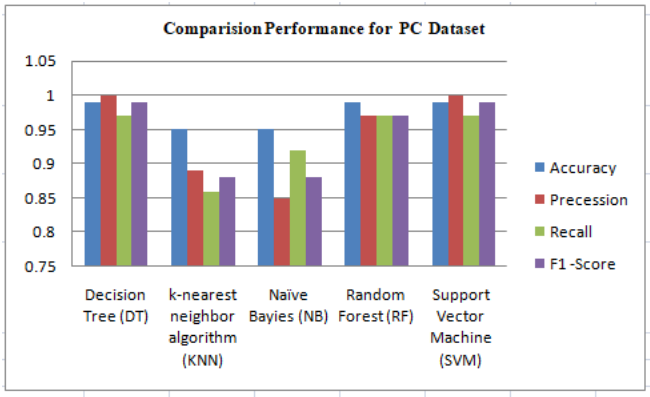

Fig. 10. Performance Analyze graph for different models for PC Datset

Overall, these comparisons highlight the suitability of various machine learning models for defect prediction across different datasets. Decision tree and random forest models consistently demonstrate robust performance across all datasets, making them promising choices for defect prediction tasks in software engineering.

VI. Conclusion

The comparison of performance metrics across multiple datasets reveals the efficacy of various machine learning models in software defect prediction tasks. Decision Tree, k-nearest neighbor algorithm, Naïve Bayes, Random Forest, and Support Vector Machine exhibit strong predictive capabilities, achieving high accuracy and balanced precision, recall, and F1-scores across different datasets. These models demonstrate their reliability and robustness in identifying potential defects in software artifacts, thereby aiding in quality assurance and software maintenance processes.

Future Scope

In future the proposed method can be extended with Further investigation into feature engineering techniques could lead to the identification of more informative features, thereby improving the performance of defect prediction models.

References

1. Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An efective software bug prediction model using deep representation and ensemble learning techniques. Expert Systems with Applications, 144, 113085. https://doi.org/10.1016/j.eswa.2019.113085.
2. Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Information and Software Technology, 96, 94–111. https://doi. org/10.1016/j.infsof.2017.11.008.
3. Zhu, K., Zhang, N., Ying, S., et al. (2020). Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network. IET Software, 14(3), 185–195. https://doi.org/10.1049/iet-sen.2019.0278.

4. Shippey, T., Bowes, D., & Hall, T. (2019). Automatically identifying code features for software defect prediction: Using AST N-grams. Information and Software Technology, 106, 142–160. https://doi. org/10.1016/j.infsof.2018.10.001.
5. Khuat, T. T., & Le, M. H. (2020). Evaluation of sampling-based ensembles of classifers on imbalanced data for software defect prediction problems. SN Computer Science, 1(2), 108. https://doi.org/10. 1007/s42979-020-0119-4
6. Feng, S., Keung, J., Yu, X., et al. (2021). Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Information and Software Technology, 139, 106662. https://doi.org/10.1016/j.infsof.2021.106662.
7. Nehéz, K., & Khleel, N. A. A. (2022). A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory. Production Systems and Information Engineering, 10(3), 1–15. https://doi.org/10.32968/psaie.2022.3.1.
8. Fan, G., Diao, X., Yu, H., et al. (2019). Software defect prediction via attention-based recurrent neural network. Scientifc Programming, 2019. https://doi.org/10.1155/2019/6230953.
9. Swana, E. F., Doorsamy, W., & Bokoro, P. (2022). Tomek link and SMOTE approaches for machine fault classifcation with an imbalanced dataset. Sensors, 22(9), 3246. https://doi.org/10.3390/s22093246
10. Tong, H., Wang, S., & Li, G. (2020). Credibility based imbalance boosting method for software defect proneness prediction. Applied Sciences, 10(22), 8059. https://doi.org/10.3390/app10228059
11. Majd, A., Vahidi-Asl, M., Khalilian, A., et al. (2020). SLDeep: Statement-level software defect prediction using deep-learning model on static code features. Expert Systems with Applications, 147, 113156. https://doi.org/10.1016/j.eswa.2019.113156.
12. Liang, H., Yu, Y., Jiang, L., et al. (2019). Seml: A semantic LSTM model for software defect prediction. IEEE Access, 7, 83812–83824. https://doi.org/10.1109/ACCESS.2019.2925313
13. Munir, H. S., Ren, S., Mustafa, M., et al. (2021). Attention based GRU-LSTM for software defect prediction. PLoS ONE, 16(3), e0247444. https://doi.org/10.1371/journal.pone.0247444.
14. Dam, H. K., Pham, T., Ng, S. W., et al. (2018). A deep tree-based model for software defect prediction. arXiv preprint arXiv:1802.00921, DOI https://doi.org/10.48550/arXiv.1802.00921.
15. Qiu, S., Xu, H., Deng, J., et al. (2019). Transfer convolutional neural network for cross-project defect prediction. Applied Sciences, 9(13), 2660. https://doi.org/10.3390/app9132660
16. Yang Z, & Qian H, (2018) Automated Parameter Tuning of Artifcial Neural Networks for Software Defect Prediction. In Proceedings of the 2nd International Conference on Advances in Image Processing (pp. 203–209). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/32395 76.3239622
17. Bashir, K., Li, T., & Yohannese, C. W. (2018). An empirical study for enhanced software defect prediction using a learning-based framework. International Journal of Computational Intelligence Systems, 12(1), 282–298. https://doi.org/10.2991/ijcis.2018.125905638
18. Ferenc, R., Bán, D., Grósz, T., et al. (2020). Deep learning in static, metric-based bug prediction. Array, 6, 100021. https://doi.org/10.1016/j.array.2020.100021. Deng, J., Lu, L., & Qiu, S. (2020a). Software defect prediction via LSTM. IET Software, 14(4), 443–450. https://doi.org/10.1049/iet-sen.2019.0149 .
19. Lango, M., & Stefanowski, J. (2018). Multiclass and feature selection extensions of roughly balanced bagging for imbalanced data. Journal of Intelligent Information Systems, 50, 97–127. https://doi.org/10. 1007/s10844-017-0446-7.

Jonathan B, Putra PH, & Ruldeviyani Y, (2020) Observation imbalanced data text to predict users selling products on female daily with smote, tomek, and smote-tomek. In 2020 IEEE International Conference on Industry 4.0, Artifcial Intelligence, and Communications Technology (IAICT) (pp. 81–85). Bali, Indonesia: IEEE. https://doi.org/10.1109/IAICT50021.2020.9172033.