Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Software Defect Prediction Using Machine Learning and Deep Learning Techniques

Muhammad Jumare[1], Haruna and Darius, Tienhua Chinyio[2]
[1,2] Department of Computer Science,
Nigerian Defence Academy, Kaduna-Kaduna Sate
muhammadjumare@gmail.com[1], dtchinyo@nda.edu.ng[2]

Abstract

Software defect prediction (SDP) has become a critical component in modern software development, aiming to identify potential bugs early in the development process. Despite advancements in the field, existing SDP models often struggle with accuracy and class imbalance issues, limiting their practical application in real-world software development environments. The increasing complexity of software systems necessitates more robust and accurate defect prediction techniques to enhance software quality and reduce development costs. This study aims to enhance software defect prediction through machine learning and deep learning techniques, focusing on improving accuracy and addressing class imbalance. The research seeks to develop and compare various machine learning and deep learning models to identify the most effective approach for early defect detection in software development processes. The study employs a comprehensive methodology utilizing Random Forest, Support Vector Machines (SVM), Naive Bayes, Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN) algorithms. The JM1 NASA software defect dataset was used, consisting of 10,885 instances with 22 code metric attributes. To address class imbalance, a hybrid sampling technique (SMOTE-Tomek) was implemented. Models were evaluated using multiple performance metrics including accuracy, precision, recall, F1-score, and AUC-ROC. This approach allows for a thorough comparison of traditional machine learning and deep learning techniques in the context of software defect prediction. The Random Forest model achieved the best overall performance with an accuracy of 82.3%, recall of 96.8%, F1-score of 0.898, and precision of 83.7%, significantly outperforming previous benchmarks. The CNN model also showed promise, achieving 81.95% accuracy and 95.90% recall. These results demonstrated substantial improvements in handling class imbalance and overall predictive performance compared to existing studies. The high recall rates suggest that these models, particularly Random Forest, are effective at identifying a large proportion of defects, which is crucial in software development where missing defects can be costly. However, the study also highlighted the ongoing challenge of balancing precision and recall in software defect prediction. The findings contribute to the refinement of software defect prediction methodologies, offering improved models for early defect detection in software development processes and opening avenues for further research in applying machine learning to software engineering challenges.

.

Keywords: Software Defect, Machine learning, Software Engineering, Deep learning, NASA JM1 dataset

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

1. Introduction

Software defects, also known as bugs or software faults, are errors or flaws in computer programs that lead to incorrect behavior or unintended results (Elentukh 2023; Shafiq et al. 2023). These defects can have severe consequences, ranging from system crashes and data corruption to security vulnerabilities and financial losses. As software systems continue to grow in complexity, the task of detecting and eliminating defects has become an increasingly critical challenge in software development and maintenance (Mahmoud et al. 2024; Mehmood et al. 2023).

The increasing intricacy of modern software systems has rendered manual defect detection not only extremely challenging but also highly time-consuming. This has necessitated the development and implementation of automated techniques for efficient and effective defect detection (Vogel-Heuser et al. 2015). In recent years, Machine Learning (ML) and Deep Learning (DL) techniques have emerged as powerful tools in addressing this challenge (Adam, Abatcha, and Aboaba 2022; Esteves et al. 2020; Li et al. 2017).

Machine Learning enables computers to learn from data and make predictions without explicit programming (LeCun, Bengio, and Hinton 2015). In the context of software defect detection, ML techniques analyze historical data such as source code, execution traces, and bug reports to identify patterns and predict potential defects. Deep Learning, a subset of machine learning, utilizes artificial neural networks with multiple layers to learn hierarchical data representations. When applied to software defect detection, DL models can be trained on large datasets to automatically learn complex patterns associated with defects (Zhao et al. 2024).

Numerous studies have explored the application of ML and DL techniques for software defect detection, encompassing supervised learning, unsupervised learning, and hybrid approaches (Hasanpour et al. 2020; Pandey, Mishra, and Tripathi 2021; Sanusi et al. 2019; Thomas and Kaliraj 2024). These techniques leverage historical data and patterns to build predictive models capable of identifying potentially defective code or components. Despite showing promising results, the adoption of these techniques in industry remains limited due to several challenges. These include issues related to data quality, model interpretability, and integration into existing software development processes (Ali et al. 2021; Nevendra and Singh 2021; Pandey et al. 2021; Shen and Chen 2020).

Software Defect Prediction (SDP) has become increasingly important in modern software development. However, existing models often struggle with accuracy and class imbalance issues. This research aims to enhance software defect prediction through advanced machine learning techniques, focusing on improving accuracy and addressing class imbalance. The study employs a comprehensive methodology, utilizing Random Forest, Support Vector Machines (SVM), Naive Bayes, Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN) algorithms.

By taking into account these challenges and exploring the potential of both traditional machine learning and deep learning approaches, this study seeks to contribute to the refinement of software defect prediction methodologies. The goal is to offer improved models for early defect detection in software development processes, thereby enhancing software quality assurance practices and opening avenues for further research in applying machine learning to software engineering challenges.

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

2. Related Works

This section presents a review of recent works in software defect prediction, The technique of predicting defective modules in a software system is known as SDP. Over the past twenty years, numerous methods have been presented in the literature that use historical defect datasets to try and determine the relationship between a collection of attributes or features of a particular software module and the presence of defects in that module. Software metrics, such as McCabe's Cyclomatic Complexity (McCabe 1976), Halstead Metrics Halstead (1977), and the Chidamber and Kemerer object-oriented metrics suite Chidamber and Kemerer (1994), are the most often utilised features to predict problems.

Sanusi et al. (2019), developed a software defect prediction system using machine learning algorithms to identify software bugs promptly. Their study employed Random Forest, Decision Tree, and Artificial Neural Network (ANN) algorithms, with Random Forest outperforming others in accuracy (83.40%), precision (53.18%), and F-score (52.04%). While the study demonstrated robust methodology through feature selection and cross-validation, it lacked detailed information about the datasets used and did not address the impact of class imbalance on algorithm performance.
Olorunshola et al. (2020), evaluated twelve machine learning classification algorithms using the PROMISE dataset. Their comprehensive study found that the Random Forest algorithm outperformed most others, while the Bayes Net classifier excelled in terms of the false positive rate. However, the study's focus on a single dataset raised concerns about result generalizability.

Hasanpour et al. (2020) addressed high dimensionality and class imbalance issues using Deep Belief Networks (DBN) and Stack Sparse Auto-Encoders (SSAE). Their findings indicated that deep learning models, particularly SSAE, provided more accurate predictions for most NASA datasets than traditional methods. However, the study identified weaknesses in handling severely imbalanced classes and datasets with insufficient examples.

Nevendra and Singh (2021) developed an enhanced Convolutional Neural Network (CNN) model for software defect prediction. Their two-stage approach, involving feature selection and data transformation, outperformed state-of-the-art techniques such as KNN, SVM, and AdaBoost across 19 open-source datasets. However, the study lacked detailed explanations for architectural choices and did not address cross-project defect prediction scenarios.

Ali et al. (2021), focused on early software defect prediction in NASA datasets, employing various machine learning classification algorithms. Their novel tuned XGBoost model achieved the highest accuracy of 95.98% on the MW1 dataset. While the study effectively handled class imbalance, it lacked detailed information on parameter tuning and model interpretability.

Benin et al.(2022) tried to investigate six re-sampling techniques: synthetic minority over sampling technique(SMOTE), borderline SMOTE ,safe-level SMOTE, Oversampling using adaptive synthetic (ADASYN), random over sampling, and random under sampling. The six approaches were conducted on 40 releases of 20 open-source projects with an imbalance ratio between the ranges of 3.8 to 17.46%, and two types of metrics were used. The study utilize used several machine learning algorithms and concluded that: first, resampling methods significantly improved the performance of the prediction model in terms of

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

all model metrics except for Area Under the Roc Curve; second the performance of resampling method depends on the imbalance ratio of the dataset (ratio of defects and clean instances); third, random under-sampling and border-line SMOTE provided more stable results across several performance measures and prediction models.

Mehmood et al. (2023) integrated feature selection with various machine learning classifiers to improve software defect prediction accuracy. Using five NASA datasets, the study achieved significant improvements, including an average 8% accuracy improvement for the Bayesian Net algorithm and over 93% accuracy for Logistic Regression with feature selection. While the study contributed to bridging the gap between software engineering and data mining, it lacked explanations for the chosen algorithms and comparisons with state-of-the-art techniques.

Chinenye, Anyachebelu, and Abdullahi (2023), proposed improving defect prediction accuracy through decision tree algorithms, addressing issues such as feature selection and hyperparameter tuning. Their structured framework included genetic algorithms for feature selection and comprehensive evaluation metrics. However, the study lacked analysis of important features and their impact on predictions.

Elshamy, AbouElenen, and Elmougy (2023), addressed challenges in software defect detection including class imbalance and hyperparameter optimization. Their approach, using SMOTE-SVM for dataset balancing and NDSGA-II with Hyperband for hyperparameter optimization, achieved high accuracy with the Random Forest classifier. However, the study lacked an analysis of computational complexity and time requirements.

Alkaberi and Assiri (2024), focused on predicting the number of software faults using CNN and multilayer perceptron (MLP) models. Their approach, which included oversampling techniques and log transformation, demonstrated improved performance over traditional regression models. However, the study's reliance on Java datasets with specific metrics limited its external validity.

Thomas and Kaliraj (2024), proposed an enhanced Random Forest-based approach for the NASA JM1 dataset. Their methodology, which included SMOTE for class imbalance and a two-fold optimization process, achieved an accuracy of 82.96% and an F1 score of 89.53%, outperforming standard models. However, the focus on a single dataset limited the generalizability of their approach.

Jude and Uddin (2024), introduced a hybrid machine learning algorithm combining multiple models through stacking. Their approach, which integrated Explainable AI (XAI) techniques for interpretability, showed improved performance across various metrics using NASA-MD datasets. The study acknowledged limitations in generalizability and recommended further research on computational requirements.

Ali et al. (2024), developed the Voting Ensemble-Based Software Defect Prediction model (VESDP), integrating four heterogeneous supervised machine learning classifiers. The VESDP model demonstrated remarkable accuracy across seven historical defect datasets, outperforming twenty state-of-the-art techniques. However, the study's focus on NASA MDP datasets and emphasis on accuracy metrics potentially limited its broader applicability.

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

The review of related works in software defect prediction reveals several significant research gaps. These include a reliance on limited datasets, particularly NASA datasets, which constrains the generalizability of findings across diverse software projects. The persistent challenge of class imbalance in defect prediction datasets remains inadequately addressed. There's a notable lack of focus on model interpretability, which is crucial for practical adoption in software engineering contexts. While some studies have begun to explore deep learning techniques, there is insufficient investigation into optimizing various deep learning architectures specifically for defect prediction. The field also lacks studies on real-time prediction models that can adapt to evolving software projects, as well as research on effectively integrating prediction models into actual development workflows. Furthermore, there is a scarcity of comprehensive comparative studies benchmarking a wide range of traditional and deep learning techniques. Lastly, most studies treat all defects equally, neglecting the potential value in predicting specific types of defects such as security vulnerabilities or performance issues. Find solutions to these gaps could significantly advance the field, leading to more accurate, interpretable, and practically applicable defect prediction models in real-world software development scenarios.

3. Methodology

The research design of this study is depicted in Figure 1. The input dataset JM1 Dataset is first pre-processed. The preprocessing step includes data cleaning, data normalization, feature engineering, data scaling and data splitting. After the preprocessing of the input dataset JM1, a hybrid sampling technique was applied to the training dataset and evaluation was done on test data based on the algorithms including Decision Trees (DT), Random Forests (RF), Naive Bayes (NB), Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN). Finally, the results of each model were reported.

Figure 1 illustrates the research design, which encompasses the following key steps:

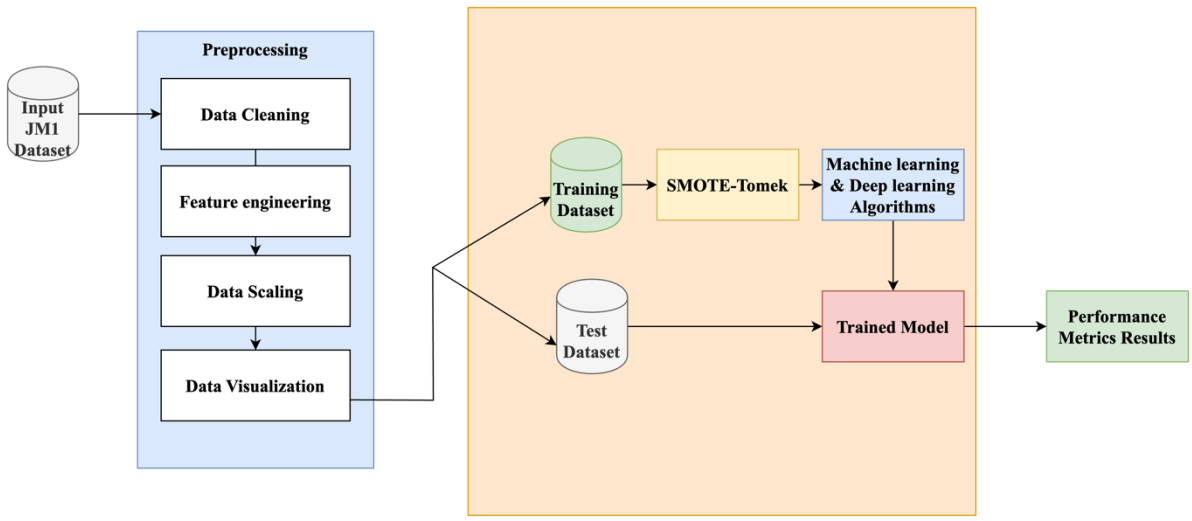

Figure 1: Research Design

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

a. Dataset Description

The study uses a secondary data source from the NASA repository (Sayyad Shirabad and Menzies 2005; Thomas and Kaliraj 2024). The dataset contains 10,885 instances and 22 attribute columns, including various code metrics. The class distribution for the 'defects' field is 19.35% false (2,106 instances) and 80.65% true (8,779 instances). Table 1 provides a detailed description of the dataset features:

Table 1:Dataset Columns Description

SNO	Feature Name	Description
1	loc	Lines of Code - the number of lines in the code.
2	v(g)	Cyclomatic Complexity - a measure of the code's complexity, indicating the number of linearly independent paths through the program.
3	ev(g)	Essential Complexity - a metric indicating the complexity of a module's structure.
4	iv(g)	Design Complexity - a measure of the complexity of a module's design.
5	n	Total Number of Nodes - indicates the number of nodes in the control flow graph of the code.
6	v	Total Number of Edges - the number of edges in the control flow graph.
7	l	Lines of Comment - the number of lines containing comments.
8	d	Lines of Documentation - the number of lines containing documentation.
9	i	Number of Input Variables - the number of variables that take input.
10	e	Number of Output Variables - the number of variables that produce output.
11	b	Number of Internal Variables - the number of variables used internally in the code.
12	t	Number of Temporary Variables - the number of temporary variables used in the code.
13	lOCode	Logical Operators - the number of logical operators in the code.
14	lOComment	Logical Operands - the number of logical operands in the code.
15	lOBlank	Number of Blank Lines - the number of blank lines in the code.
16	locCodeAndComment	Lines of Code and Comment - the combined number of lines of code and comments.
17	uniq_Op	Unique Operators - the number of unique operators used in the code.
18	uniq_Opnd	Unique Operands - the number of unique operands used in the code.
19	total_Op	Total Operators - the total number of operators in the code.
20	total_Opnd	Total Operands - the total number of operands in the code.
21	branchCount	Number of Branches - the number of branches in the code.
22	Defects	Has one or more defects.

b. Dependent and Independent Variables

The dependent variable *false* in this study is binary (i.e., *false* $\in \{0,1\}$) where 0 means that the class does not have a defect and 1 means the class has a defect.

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Figure 2: Dataset Defects Column Distribution

c. Data Preprocessing

The data preprocessing stage involved several critical steps to prepare the software metrics for defect prediction modelling. Feature engineering was performed on the 'defects' column, transforming it into a binary classification. Values greater than 0 were labelled as 'Defect', while 0 values were labelled as 'No Defect'. This transformation simplified the target variable for subsequent classification tasks. Data scaling was applied using MinMaxScaler to standardize selected features. This step normalized software metrics such as lines of code (loc), McCabe's cyclomatic complexity (v(g)), and Halstead effort (e), ensuring that all features contribute equally to the model training process. The standardization improved model performance, accelerated convergence, and enhanced generalization capabilities. A correlation analysis was conducted, and visualized through a heatmap (refer to Figure 3). This heatmap revealed important relationships between various code metrics and defect occurrence. Key metrics like lines of code (loc), cyclomatic complexity (v(g), and essential complexity (ev(g)) showed stronger correlations with defects, indicating their potential as significant predictors. The heatmap also highlighted possible multicollinearity between certain metrics, an important consideration for model development. The dataset was split into training and testing sets using the train_test_split function with a 70:30 ratio. This stratified split maintained the distribution of the target variable across both sets, ensuring a representative sample for model training and evaluation. This step was crucial for assessing the model's predictive capabilities on unseen data and preventing overfitting. These preprocessing steps collectively prepared the data for effective defect prediction modelling by transforming features, normalizing scales, analysing correlations, and creating appropriate training and testing sets.

d. Sampling

In software defect prediction, where non-defective instances often outnumber defective ones, hybrid techniques of SMOTE-Tomek were essential for addressing class imbalance in this research. SMOTE generates synthetic samples of defective instances, while Tomek Links refines decision

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

boundaries by removing borderline non-defective instances. By balancing the dataset and improving class separability, the technique enhances the six models and creates the ability to accurately predict defects, leading to more reliable and effective software quality assessments.

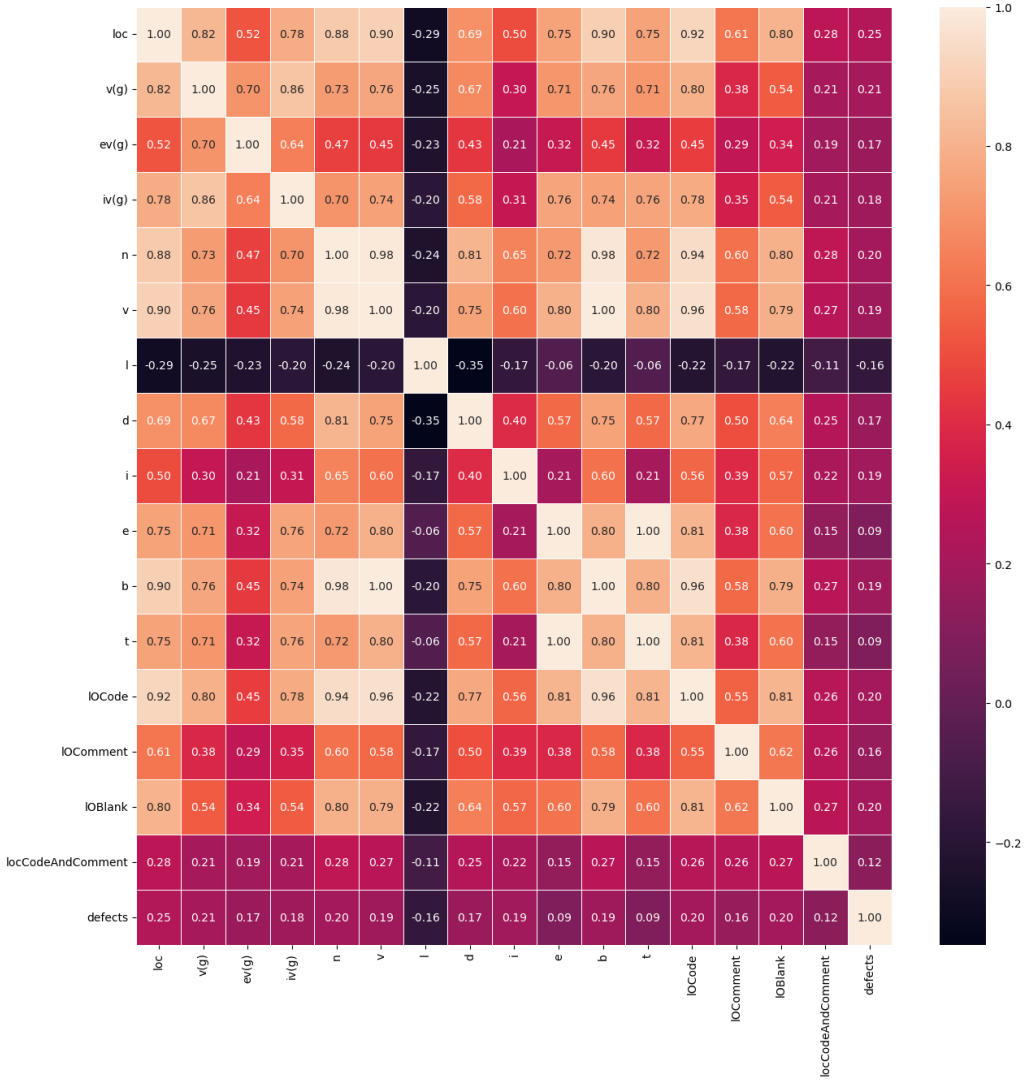

Figure 3 Correlational Heatmap

e. Model training and Evaluation phases

Post-SMOTE-Tomek, the model is trained using supervised learning and deep learning algorithms on the rebalanced dataset, employing methods such as the 'fit' function on the resampled and standardized training data. The subsequent evaluation phase is instrumental in assessing the model's efficacy and generalizability. This phase encompasses making predictions on both the resampled training data and previously unseen test data, followed by the computation of various performance metrics. These metrics include but are not limited to, accuracy, precision, recall, F1 score, confusion matrix, and the Area Under the Receiver Operating Characteristic curve (ROC

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

AUC). This comprehensive set of metrics provides a multifaceted view of the model's predictive capabilities, particularly its ability to discriminate between defective and non-defective software modules. Through meticulous interpretation of these results, researchers can identify the model's strengths and limitations, ensure its robustness against overfitting and underfitting, and validate its readiness for deployment in real-world software defect prediction scenarios. This rigorous evaluation process is paramount in establishing the model's reliability and effectiveness in the context of software quality assurance and defect management.

f. Evaluation Metrics

The study used five performance metrics to evaluate the developed model's performance which are accuracy, precision, recall F1 score and AUC-ROC . These metrics are widely used in the related literature to evaluate Software Defect Prediction models.

True positives (TP), False negatives (FN), False positives (FP), and True negatives (TN), False positives rate (FPR), True positives rate (TPR).

The following metrics are calculated as follows:

$$\textbf{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \quad (1)$$

$$\textbf{Recall} = \frac{\text{TP}}{\text{TP+FN}} \quad (2)$$

$$\textbf{Precision} = \frac{\text{TP}}{\text{TP+FP}} \quad (3)$$

$$\textbf{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}} \quad (4)$$

$$\boldsymbol{AUC} = \sum_{i-1}^{n-1} \frac{(FPR_{i+1} - FPR_i) \times (TPR_{i+1} + TPR_i}{2} \quad (5)$$

where,

True Positive (TP): This occurs when the model correctly predicts a software defect, and the defect is confirmed (i.e. when a defective instance is accurately identified as defective).

True Negative (TN): This occurs when the model correctly predicts that a software component is not defective, and it is confirmed as non-defective (i.e. when a non-defective instance is accurately identified as non-defective).

False Positive (FP): This occurs when the model incorrectly predicts a defect in a software component that is non-defective (i.e. when a non-defective instance is mistakenly identified as defective).

False Negative (FN): This occurs when the model fails to detect a defect in a software component that is defective (i.e. when a defective instance is mistakenly identified as non-defective).

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

4. Results and discussion

This section presents the results of experiments conducted on various machine learning and deep learning algorithms for software defect prediction. The algorithms evaluated include Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), Naive Bayes (NB), Artificial Neural Network (ANN), and Convolutional Neural Network (CNN).

a. Decision Tree Algorithm

The Decision Tree algorithm achieved an accuracy of 79.56% on the test data, with a precision of 84.35%, a recall of 91.68%, and an F1 score of 0.8786. The model showed some signs of overfitting, as indicated by a slight performance drop from training (85.17%). A notable class imbalance was observed, with better performance in the majority "No Defect" class. The ROC AUC score of 0.7022 suggested moderate discriminative ability.

```
Accuracy: 0.7956495098039216
Precision: 0.843466107617051
Recall: 0.9168249145461451
F1 Score: 0.8786169244767971
ROC AUC Score:0.7022419937607701
```

b. Random Forest Algorithm

Random Forest demonstrated improved performance over the Decision Tree, achieving 82.32% accuracy on test data with good generalization. It exhibited exceptional recall (96.89%) and good precision (83.75%), resulting in a strong F1 score of 0.8984. The ROC AUC score of 0.7529 indicated improved discriminative ability compared to the Decision Tree.

```
Accuracy: 0.8232230392156863
Precision: 0.8374917925147735
Recall: 0.9688568173186479
F1 Score: 0.8983976052121853
ROC AUC Score:0.7529280622695124
```

c. Support Vector Machine (SVM) Algorithm

The SVM algorithm showed good overall performance with 81.19% accuracy on test data and consistent generalization. It achieved exceptionally high recall (99.62%) and solid precision (81.28%), resulting in an F1 score of 0.8952. However, the model struggled significantly with the minority class, showing extremely low recall (4%) for the "Defect" class.

```
Accuracy: 0.8118872549019608
Precision: 0.8128292531763247
Recall: 0.996202050892518
F1 Score: 0.895221843003413
ROC AUC Score:0.6761830069765496
```

d. Naive Bayes Algorithm

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Naive Bayes demonstrated solid performance with 80.94% accuracy on test data and good generalization. It achieved high recall (94.49%) and good precision (83.91%), resulting in an F1 score of 0.8889. The ROC AUC score of 0.7020 indicated moderate discriminative ability.

```
Accuracy: 0.8094362745098039
Precision: 0.839123102866779
Recall: 0.9449297379415116
F1 Score: 0.888888888888889
ROC AUC Score:0.7020424660065498
```

e. Artificial Neural Network (ANN) Algorithm

The ANN showed strong performance with 81.95% accuracy and good generalization. It achieved high recall (96.39%) and good precision (83.71%), resulting in an F1 score of 0.8960. The ROC AUC score of 0.7295 indicated moderate discriminative ability, better than most models except Random Forest.

```
Accuracy: 0.819546568627451
Precision: 0.8370712401055409
Recall: 0.9639194834789214
F1 Score: 0.8960282436010591
ROC AUC Score:0.7294969432829569
```

f. Convolutional Neural Network (CNN) Algorithm

The CNN demonstrated strong performance with 81.95% accuracy, high recall (95.90%), and good precision (84.00%), resulting in an F1 score of 0.8955. The ROC AUC score of 0.7323 indicated moderate discriminative ability, slightly better than most models but lower than Random Forest.

```
Accuracy: 0.819546568627451
Precision: 0.8399866932801064
Recall: 0.9589821496391948
F1 Score: 0.8955488561801737
ROC AUC Score:0.7322963507788204
```

g. Comparative Analysis of Models

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 527-543
https://doi.org//10.47514/kjcs/2024.1.3.0010
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

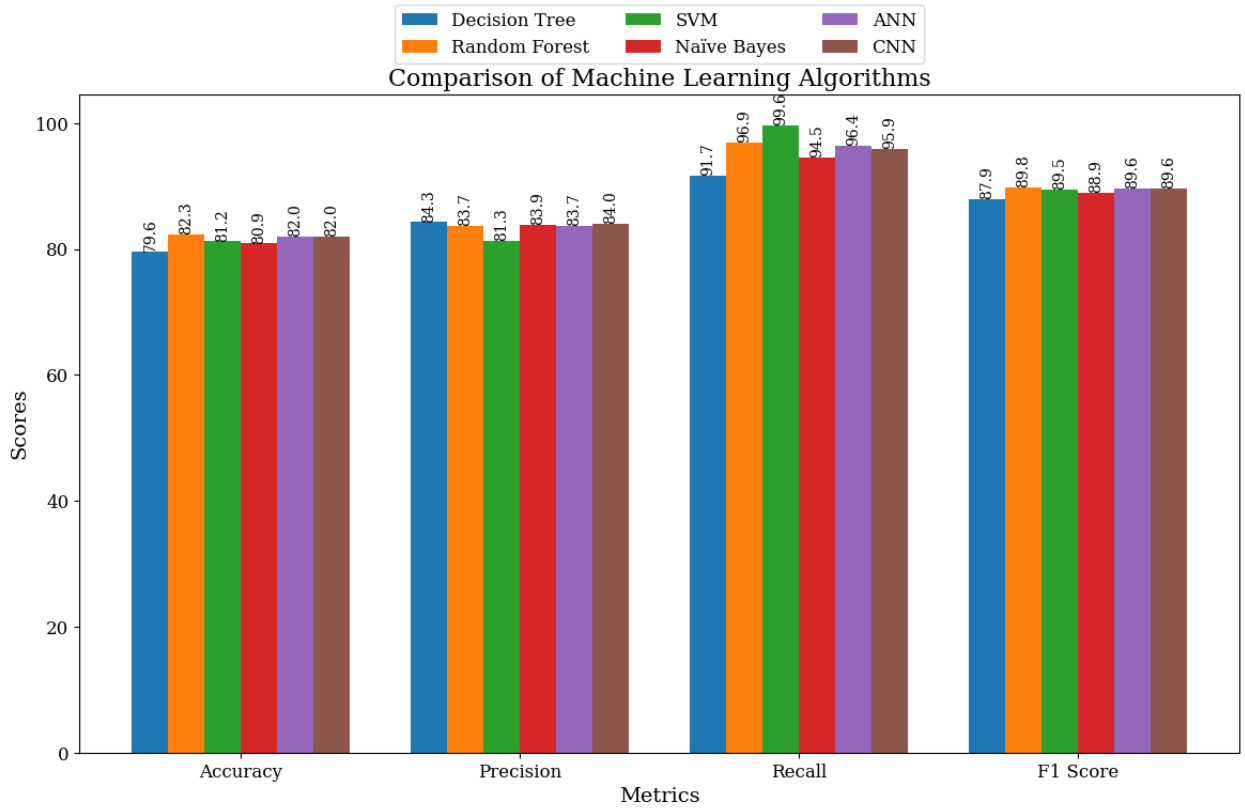

Figure 4: comparison of the performance metrics across all evaluated algorithms

Figure 4 presents a comparison of the performance metrics across all evaluated algorithms. Random Forest achieved the highest accuracy (82.3%), closely followed by ANN and CNN (both 82.0%). Decision Tree showed the highest precision (84.3%), while SVM demonstrated the highest recall (99.6%). The F1 score, balancing precision and recall, was highest for Random Forest (89.8%), with ANN and CNN close behind (both 89.6%).

h. Comparison with Existing Studies

Table 2 presents a comparison of the Random Forest algorithm's performance in this study with two previous studies.

Table 2: Comparison with existing studies

Study	Algorithm	Accuracy	Recall	F1-Score	Precision
(Olorunshola et al. 2020)	Random Forest	0.818	0.818	0.787	0.787
(Thomas and Kaliraj 2024)	Random Forest	0.7276	0.885	0.7801	0.6974
This Study	Random Forest	0. 823	0.968	0.898	0.837

The current study's implementation of Random Forest demonstrated the strongest overall performance, achieving the highest accuracy (82.3%), recall (96.8%), F1-score (0.898), and precision (83.7%) compared to the studies by Olorunshola et al. (2020), and Thomas, and Kaliraj (2024).

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

The results of this study demonstrate significant advancements in software defect prediction using machine learning techniques, particularly with the Random Forest algorithm. The performance improvements can be attributed to several factors, including advanced feature engineering, improved handling of class imbalance, and optimized hyperparameter tuning. The high recall (96.8%) of our model is particularly noteworthy in the context of software defect prediction. As (D'Ambros et al. 2012) points out, in software engineering, the cost of missing a defect (false negative) is generally higher than the cost of a false alarm (false positive). The model's ability to identify a high proportion of defects aligns well with this priority. However, it is important to acknowledge the persistent challenge of class imbalance in software defect prediction, as highlighted by (Mahmoud et al. 2024). While the model shows improved handling of this issue compared to previous studies, there is still room for enhancement, particularly in precision for the minority class. The performance of the Random Forest model aligns with findings from (Esteves et al. 2020), who identified Random Forest as one of the top-performing algorithms for software defect prediction. The study emphasized the importance of proper tuning and validation techniques, which were incorporated into our methodology. While this study demonstrates significant progress in software defect prediction, it also highlights areas for future work. These include further addressing the class imbalance, exploring advanced deep learning architectures, and investigating the interpretability of model decisions, an aspect emphasized by (Mahmoud et al. 2024; Mehmood et al. 2023) as crucial for practical adoption in software engineering.

5. Conclusion

This paper has made significant strides in advancing the field of software defect prediction through the application of machine learning and deep learning techniques. The research findings offer several important contributions and implications for both academic research and practical software development. The Random Forest algorithm emerged as the top performer, achieving an accuracy of 82.3%, recall of 96.8%, F1-score of 0.898, and precision of 83.7%. This performance significantly surpasses previous benchmarks, demonstrating the potential of ensemble methods in software defect prediction. The high recall rate is particularly noteworthy, as it indicates the model's strong capability in identifying a large proportion of defects, which is crucial in software development contexts where missing defects can lead to significant costs and risks. While traditional machine learning methods showed strong performance, the study also highlighted the promise of deep learning approaches, particularly Convolutional Neural Networks (CNN). The CNN model achieved comparable results to traditional methods, suggesting that with further optimization, deep learning could offer powerful tools for software defect prediction. The use of hybrid sampling techniques (SMOTE-Tomek) proved effective in addressing the persistent challenge of class imbalance in software defect datasets. This approach could be valuable for future research dealing with imbalanced datasets in various domains of software engineering. Also, by evaluating multiple algorithms on the same dataset, this study provides a comprehensive comparison that can serve as a benchmark for future research in software defect prediction. The improved accuracy and high recall rates of the models developed in this study have significant implications for practical software development. These models could be integrated into development workflows to identify potential defects early, thereby reducing development costs and improving software quality.

However, the study also revealed areas that require further investigation:

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

- While the models performed well on the NASA JM1 dataset, future work should focus on validating these approaches across diverse software projects and development environments to ensure generalizability.
- As model complexity increases, so does the need for interpretable results. Future research should explore ways to make the predictions of these models more explainable to developers and project managers.
- Developing models that can adapt and provide predictions in real-time as software projects evolve remains an important area for future work.
- Advanced feature engineering techniques specifically tailored for software metrics could potentially improve model performance further.
- Future studies could explore the prediction of specific types of defects (e.g., security vulnerabilities, performance issues) which could be more valuable in certain development contexts.

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

References

Adam, Hassan, Muhammad Abatcha, and Abdulfattah Aboaba. 2022. 'Design of a Hybrid Machine Learning Base-Classifiers for Software Defect Prediction'. *International Journal of Innovative Research and Development*. doi: 10.24940/ijird/2022/v11/i10/OCT22020.

Ali, Misbah, Tehseen Mazhar, Yasir Arif, Shaha Al-Otaibi, Yazeed Yasin Ghadi, Tariq Shahzad, Muhammad Amir Khan, and Habib Hamam. 2024. 'Software Defect Prediction Using an Intelligent Ensemble-Based Model'. *IEEE Access*.

Ali, Sikandar, Muhammad Adeel, Sumaira Johar, Muhammad Zeeshan, Samad Baseer, and Azeem Irshad. 2021. Classification and Prediction of Software Incidents Using Machine Learning Techniques'. *Security and Communication Networks* 2021(1):9609823.

Alkaberi, Wahaj, and Fatmah Assiri. 2024. 'Predicting the Number of Software Faults Using Deep Learning'. *Engineering, Technology & Applied Science Research* 14(2):13222–31. doi: 10.48084/etasr.6798.

Alzahrani, Musaad. 2022. 'Using Machine Learning Techniques to Predict Bugs in Classes: An Empirical Study'. *International Journal of Advanced Computer Science and Applications* 13(5).

Bennin, K.E., Tahir, A., MacDonell, S.G., B¨orstler, J. (2022) An empirical study on the effectiveness of data resampling approaches for cross-project software defect prediction. IET Software 16(2),185–199

Chidamber, Shyam R., and Chris F. Kemerer. 1994. 'A Metrics Suite for Object-Oriented Design'. *IEEE Transactions on Software Engineering* 20(6):476–93.

Chinenye, Obidike, Kene Tochukwu Anyachebelu, and Muhammad Umar Abdullahi. 2023. 'Software Defect Prediction System Based on Decision Tree Algorithm'. *Asian Journal of Research in Computer Science* 16(4):32–48.

D'Ambros, M., M. Lanza, and R. Robbes. 2012. 'Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison'. *Empirical Software Engineering* 17(4–5):531–77.

Elentukh, Alex. 2023. 'People Make Mistakes–A Survey of Common Causes of Software Defects'. Pp. 117–33 in *International Conference on Computer Science and Education in Computer Science*.

Elshamy, Nawal, Amal AbouElenen, and Samir Elmougy. 2023. 'Automatic Detection of Software Defects Based on Machine Learning'. *International Journal of Advanced Computer Science and Applications* 14(3).

Esteves, Geanderson, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, and Nivio Ziviani. 2020. 'Understanding Machine Learning Software Defect Predictions'. *Automated Software Engineering* 27(3):369–92.

Goyal, Somya. 2022. 'Handling Class-Imbalance with KNN (Neighbourhood) under-Sampling for Software Defect Prediction'. *Artificial Intelligence Review* 55(3):2023–64.

Halstead, Maurice H. 1977. *Elements of Software Science*. Elsevier Science Inc.

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Hasanpour, Ahmad, Pourya Farzi, Ali Tehrani, and Reza Akbari. 2020. 'Software Defect Prediction Based on Deep Learning Models: Performance Study'. *ArXiv Preprint ArXiv:2004.02589*.

Jude, Agboeze, and Jia Uddin. 2024. 'Explainable Software Defects Classification Using SMOTE and Machine Learning'. *Annals of Emerging Technologies in Computing (AETiC)* 8(1).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. 'Deep Learning'. *Nature* 521(7553):436–44.

Li, Jian, Pinjia He, Jieming Zhu, and Michael R. Lyu. 2017. 'Software Defect Prediction via Convolutional Neural Network'. Pp. 318–28 in *2017 IEEE international conference on software quality, reliability and security (QRS)*.

Mahmoud, Alia Nabil, Ahmed Abdelaziz, Vitor Santos, and Mario M. Freire. 2024. 'A Proposed Model for Detecting Defects in Software Projects'. *Indonesian Journal of Electrical Engineering and Computer Science* 33(1):290–302.

McCabe, Thomas J. 1976. 'A Complexity Measure'. *IEEE Transactions on Software Engineering* SE-2(4):308–20.

Mehmood, Iqra, Sidra Shahid, Hameed Hussain, Inayat Khan, Shafiq Ahmad, Shahid Rahman, Najeeb Ullah, and Shamsul Huda. 2023. 'A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine Learning'. *IEEE Access*.

Nevendra, Meetesh, and Pradeep Singh. 2021. 'Software Defect Prediction Using Deep Learning'. *Acta Polytechnica Hungarica* 18(10):173–89.

Olorunshola, Oluwaseyi Ezekiel, Martins E. Irhebhude, Abraham E. Evwiekpaefe, and Francisca Nonyelum Ogwueleka. 2020. 'Evaluation of Machine Learning Classification Techniques in Predicting Software Defects'. *Trans. Mach. Learn. Artif. Intel* 8:1–15.

Pandey, Sushant Kumar, Ravi Bhushan Mishra, and Anil Kumar Tripathi. 2021. 'Machine Learning Based Methods for Software Fault Prediction: A Survey'. *Expert Systems with Applications* 172:114595.

Sanusi, B. A., S. O. Olabiyisi, A. O. Olowoye, and B. L. Olatunji. 2019. 'Software Defect Prediction System Using Machine Learning Based Algorithms'. *Journal of Advances in Computational Intelligence Theory* 1(3):1–9.

Sayyad Shirabad, J., and T. J. Menzies. 2005. 'The PROMISE Repository of Software Engineering Databases.'

Shafiq, Muhammad, Fatemah H. Alghamedy, Nasir Jamal, Tahir Kamal, Yousef Ibrahim Daradkeh, and Mohammad Shabaz. 2023. 'Retracted: Scientific Programming Using Optimized Machine Learning Techniques for Software Fault Prediction to Improve Software Quality'. *IET Software* 17(4):694–704.

Shen, Zhidong, and Si Chen. 2020. 'A Survey of Automatic Software Vulnerability Detection, Program Repair, and Defect Prediction Techniques'. *Security and Communication Networks* 2020(1):8858010.

Thomas, Nikhil Saji, and S. Kaliraj. 2024. 'An Improved and Optimized Random Forest Based Approach to Predict the Software Faults'. *SN Computer Science* 5(5):530.

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Vogel-Heuser, Birgit, Alexander Fay, Ina Schaefer, and Matthias Tichy. 2015. 'Evolution of Software in Automated Production Systems: Challenges and Research Directions'. *Journal of Systems and Software* 110:54–84.

Zhao, Guoliang, Stefanos Georgiou, Safwat Hassan, Ying Zou, Derek Truong, and Toby Corbin. 2024. 'Enhancing Performance Bug Prediction Using Performance Code Metrics'. Pp. 50–62 in *Proceedings of the 21st International Conference on Mining Software Repositories*.