MIJST
MIST International Journal of Science and Technology

A Neural Network Based Software Defect Prediction Approach Using SMOTE and Noise Filtering-CLNI

Ahmmed Bin Ashfaque[1], Abdus Sattar[2], Hosney Jahan[*3], M. Akhtaruzzaman[4], and Fernaz Narin Nur[5]

[1]Department of CSE, Bangladesh Army University of Science and Technology, Fultola Khulna-9204, Bangladesh
[2]Department of Computer Science and Engineering, Military Institute of Science and Technology, Dhaka, Bangladesh
[3]Department of Computer Science and Engineering, East West University, Dhaka-1212, Bangladesh
[4]Department of Computer Science and Engineering, Daffodil International University, Savar, Dhake-1216, Bangladesh
[5]Quantum Robotics and Automation Research Group (QRARG), Mirpur, Dhaka-1216, Bangladesh

***Corresponding Email:** *hosney.jahan@ewubd.edu*

ARTICLE INFO

Article History:

Received: 19th May 2025
Revised: 29th September 2025
Accepted: 13th October 2025
Published: 30th December 2025

Keywords:

Software Defect Prediction
SMOTE
CLNI
Dense Neural Network
Data Balancing
Feature Selection

ABSTRACT

Software defects can cause significant loss and system failures in software development life cycle. Software Defect Prediction (SDP) is a vital step for ensuring the quality of software. Till now, a number of machine learning models have been proposed to predict potential defects and make the software more reliable. However, SDP models suffer from the problem of imbalanced dataset, resulting in poor prediction accuracy. To mitigate this, issue several data balancing techniques, i.e., over sampling, under sampling etc. have been proposed to balance the dataset. In some cases, the data balancing methods may further introduce noisy and mislabeled samples in the dataset. To deal with these issues, in this paper, we propose a neural network based approach that combines the oversampling technique Synthetic Minority Oversampling Technique (SMOTE) with the noise filtering technique Class Level Noise Identification (CLNI). Here, we applied three different CLNI methods which are Edited Nearest Neighbor (ENN), Repeated ENN (RENN) and All-KNN. Our aim is to make the dataset clean, balanced and efficient by combining SMOTE with CLNI. In addition, we applied a number of feature selection methods to identify the most important features, further contributing towards achieving better prediction accuracy. To evaluate the effectiveness of the proposed model, we conduct experiments on several benchmark datasets (MC1, PC1, PC2, PC3 and PC4) obtained from NASA MDP and (ML, LC and JDT) AEEEM repository. The experimental results have been evaluated and compared in terms of accuracy, precision, recall and AUC-ROC curve. The experimental results demonstrated that our proposed approach has achieved up to 98% accuracy and outperformed state-of-the-art approaches.

1. INTRODUCTION

In the domain of software engineering, the identification and classification of software defects is a critical activity during the testing phase of software development. With the increasing complexity of software systems and growing demands for higher reliability, it is necessary to ensure the quality of the software. In the modern era, where numerous software systems compete in terms of functionality and performance, predicting software defects has become a challenging task. Figure 1 and Figure 2 demonstrate the example of defective and non-defective code in the software.

The primary objective of predicting software defects is to reduce the effort of software testing through proper guideline. Proper software testing enables many software organizations to predict defects in a way that saves time, stays within budget, improves software quality, and facilitates better resource planning to meet project timelines (*Akintola et al., 2018*).

```
public static int defectiveSum(int[] arr) {
    int sum = 0;
    for (int num : arr) {
        sum = num;
        // Incorrectly overwrites the sum instead of accumulating
    }
    return sum;
}
```

Figure 1: Defective code example

```
public static int cleanSum(int[] arr) {
    int sum = 0;
    for (int num : arr) {
        sum += num;
        // Accumulate the sum correctly
    }
    return sum;
}
```

Figure 2: Non-defective code example

Till now a number of machine learning approaches have been applied in software defect prediction. However, the performance of these methods are often hindered by imbalanced datasets. Class imbalance is the major problem related to software defect dataset, where the unequal distribution between the number of defective instances (i.e. the minority class) is significantly smaller than the non-defective instances (i.e. the majority class) (*Feng et al., 2021*). This imbalance limits the classifiers ability to classify the minority class, often leading to misclassification error.

To address this problem, various data balancing techniques are available, i.e., oversampling, undersampling etc. Among them, SMOTE (*khleel et al., 2024*) and Random oversampling (*Khleel et al., 2024*) has been widely adopted to balance datasets by generating synthetic samples for the minority class. However, SMOTE has a notable drawback - it can introduce noise by generating synthetic instances from or near mislabeled or noisy data points (*Alkhawaldeh et al., 2023*), result in over-fit prediction model (*Rathore et al., 2024*). Because, SMOTE lacks a mechanism to distinguish between clean and noisy samples, it may inadvertently amplify existing errors in the data.

To mitigate these drawbacks, in this paper, we propose a neural network based approach that combines the oversampling technique SMOTE with the noise removing technique CLNI. Here, we applied three noise filtering techniques which are Edited Nearest Neighbor (ENN), Repeated ENN (RENN) and All-KNN (*Gupta et al., 2023*). These techniques act as a preprocessing step, enhancing data quality and improving the performance of classifiers. Our aim is to reduce the misclassification rate and overfitting problem by Integrating SMOTE with CLNI. In addition to this, we further applied a number of feature selection methods to achieve more accurate prediction. The feature selection methods used in our research are Correlation Coefficient, Mutual Information, Variance Threshold and Chi-Square. The objective of this research is to develop accurate software defect prediction model by addressing the challenges of class imbalance, data noise and irrelevant features through an integrated approach.

This research paper is divided into sections as follows: Literature review is presented in Section II. Methodology is illustrated in Section III. Result analysis is presented in Section IV. Conclusions are drawn in section V.

2. LITERATURE REVIEW

In this section, we review previous research on Software Defect Prediction (SDP), highlighting significant contributions in the areas of handling class imbalance, feature selection techniques and neural network approaches for early defect prediction in the Software Development Cycle (SDLC).

Software defect datasets are challenging to use due to their imbalanced class distributions. (*Ali et al., 2024*) used the SMOTE preprocessing approach with NASA MDP datasets (CM1, JM1, PC1, PC4, MC2, PC3, and MW1). Using the voting ensemble approach combined with machine learning algorithms such as RF, SVM, NB, and ANN to improve prediction performance. Their method achieved a training accuracy of 87.2% and a testing accuracy of 86.87%. SMOTE may introduce unnecessary noise. To overcome with this problem, (*Vuttipittayamongkol et al., 2022*) proposed neighborhood-based undersampling algorithm. This algorithm reduced the majority class and balance the ratio between majority and minority class. Experimental results observed on NASA datasets, having an AUC of 95.9% and an accuracy of 96.9% with 95% confidence level. Undersampling technique may possess loss of important data.

A real-time deep ladder network proposed to tackle the problem of SMOTE and undersampling by (*J. et al., 2023*). This method integrated with migration learning, a deep belief network combined with transfer component analysis superior that others.

Software defect datasets consist of large number of metrics such as line of code (LOC), complexity measures, cyclomatic complexity and so on. Many of them are irrelevant in software defect prediction. To reduce model's overfitting, (*Cetiner et al., 2020*) applied Principal Component Analysis (PCA) as a feature reduction technique, while comparing ten machine learning algorithms. They used Decision Tree, Naïve Bayes, K-nearest neighbor, Support vector machine, random forest, extra tress, adaboost, gradient boosting, bagging and multi-layer perceptron. The evaluation performance showed that Random forest achieved higher accuracy on PC1 dataset.

A five-stage framework for software defect prediction using NASA datasets proposed by (*Ali et al., 2024*). They combined genetic algorithm-based feature selection with heterogeneous classifiers (Random Forest, SVM and Naïve Bayes). Their iterative optimization process, utilizing an ensemble voting technique, achieved a maximum classification accuracy of 95.1%, while reducing execution time by over 50%.

Enhanced exploratory whale optimizer-based feature selection method along with random forest ensemble learning proposed (*Mafarja et al., 2023*) to software defect prediction while Siamese Dense Neural Network (SDNN) introduced by (*Zhao et al., 2018*), which combines similarity feature learning and distance metric learning using a contrast loss function with cosine proximity. Their experimental results demonstrated that SDNN outperforms traditional methods, particularly in scenarios with limited data and imbalanced class distributions.

The literature study highlights several challenges in software defect prediction, including noise introduction during resampling methods like SMOTE, the complexity of feature selection, and the high computational costs associated with

ensemble models and deep learning methods. The purpose of the research is to improve accuracy and efficiency in software defect prediction, addressing the limitations identified in previous studies.

3. METHODOLOGY

This research presents a software defect prediction approach that combines SMOTE and CLNI for data balancing, feature selection for selecting important features, and a Dense Neural Network for classification. Figure 3 depicts an overview of the approach consisting four phases: 1) Dataset collection, 2) Data preprocessing, 3) Feature selection, and 4) Prediction.

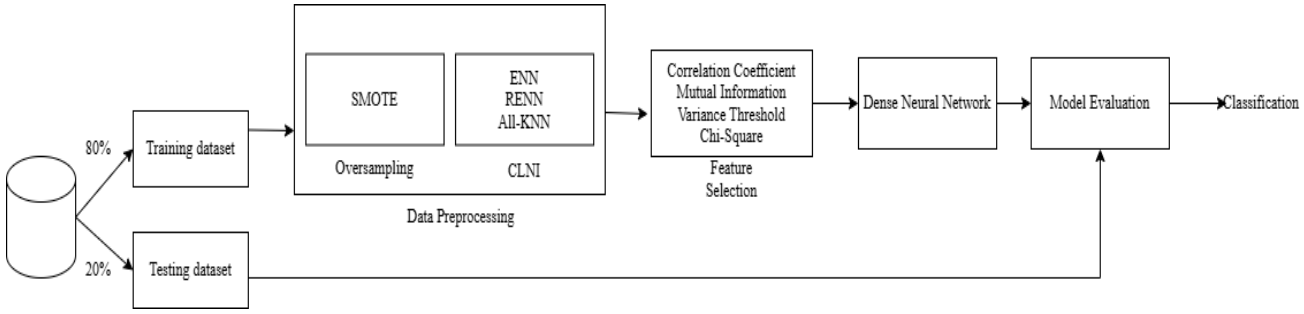

Figure 3: Overview of the Software Defect Prediction Model

3.1 Dataset Collection

To measure the effectiveness of our proposed approach, we considered a number of well-known and widely used datasets from NASA MDP (Metrics Data Program) repository and AEEEM dataset repository. They contain various software metrics which were organized and collected from various projects and widely used in software defect prediction related tasks. We selected MC1, PC1, PC2, PC3 and PC4 datasets from NASA MDP and JDT, ML and LC from AEEEM repository for our approach. Each of them consists of multiple attributes like Halstead metrics, code complexity metrics and many others. Table 1 and Table 2 describe the details of the NASA MDP and AEEEM dataset respectively.

Table 1
NASA MDP Dataset

Dataset	Attributes	Modules	Defective	Non-defective
MC1	39	9466	68	9398
PC1	40	1107	76	1031
PC2	40	1563	160	1403
PC3	40	1270	176	1094
PC4	36	1694	458	1236

Table 2
AEEEM Dataset

Project	Number of files	Percentage of Buggy Files	Number of metrics
LC	399	9.26	71
ML	1862	13.16	71
JDT	997	20.66	71

3.2 Data Preprocessing

To preprocess the defect datasets, we applied two methods: data balancing with SMOTE and noise removing with CLNI. Though there are many noise filtering methods, but in our research we used only three of them. The details of these methods are discussed in the following subsections.

3.2.1 Dataset Balancing with SMOTE

SMOTE creates new minority class by linear interpolation between existing samples and their neighbors (Elreedy et al., 2019). For each minority samples X_0, one of its K nearest neighbors X is chosen at random, and a synthetic dataset is created as follows:

Algorithm: *SyntheticSampleGen(X_minority, N, k)*

Input: *Minority class samples* $x\_minority$, *number of synthetic samples N, number of nearest neighbors k*

***Output**: Augmented dataset with synthetic samples*

Repeat N times:
Select a random sample x_0 from $x\_minority$
Find its k nearest neighbors in $x\_minority$
Choose one neighbor x at random
Generate a random value w between 0 and 1
Create a new sample z between x_0 and x using

$$z = x_0 + w \times (x - x_0)$$

Add z to the set of synthetic samples
Return $x\_minority \cup \{z_1, z_2, \ldots . z_n\}$ as the augmented dataset

3.2.2 Noise Removal using CLNI

CLNI is a method to reduce the noise generated from the oversampling technique SMOTE. Many types of CLNI methods are available, among them, we performed ENN, RENN and All-KNN in our research.

Edited Nearest Neighbors: Edited Nearest Neighbors is an enhanced version of k-nearest Neighbors to eliminate noisy and mislabeled instances from the dataset.

Algorithm *ENN (D, k)*

***Input**: Dataset D, number of neighbors k*

For each instance x in D:
Find k nearest neighbors of x
Determine majority class among neighbors
If class(x) ≠ majority class:
Remove x from D
Return cleaned dataset D

Repeated ENN: RENN removes noisy instances that differ from their neighbors. It repeats the process until no more instances are removed in an iteration.

Algorithm: *RENN (D, k)*

***Input**: Dataset D, number of neighbors k*

Repeat:
Initialize change_flag = False
For each instance x in D:
Find k nearest neighbors of x
Determine majority class among neighbors
If class(x) ≠ majority class:
Remove x from D
Set change_flag = True
Until change_flag is False
Return cleaned dataset D

All-KNN: All-KNN reduces the borderline and mislabeled instances, in noisy or mislabeled datasets.

Algorithm: *AllKNN (D, k_max)*

***Input**: Dataset D, maximum neighbors k_max*

For k from 1 to k_max:
For each instance x in D:
Find k nearest neighbors of x
Determine majority class
If class(x) ≠ majority class:
Mark x for removal
Remove all marked instances
Return cleaned dataset D

3.3 Feature Selection

To reduce the dimension of data to reduce the memory wastage, this paper applied Correlation Coefficient, Mutual Information, Variance Threshold and Chi-Square feature selection methods. The most significant features identified are included:

- LOC_BLANK,
- PERCENT_COMMENTS,
- LOC_COMMENTS,
- LOC_CODE_AND_COMMENT,
- NUMBER_OF_LINES,
- CALL_PAIRS,
- NUM_UNIQUE_OPERANDS,
- NUM_UNIQUE_OPERATORS,
- HALSTEAD_CONTENT,
- LOC_TOTAL,
- DECISION_DENSITY, and
- LOC_EXECUTABLE

3.3.1 Correlation Coefficient

Correlation coefficient is a statistical technique used to measure and evaluate the strength and direction of the linear relationship between two variables (*Schober et al., 2018*). It is commonly quantified using metrics like Pearson's correlation coefficient for continuous data. Mathematically, it can be written as (Venkatesh et al., 2019),

$$\rho(x, y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \, (y_i - \bar{y})^2}} \qquad (1)$$

The correlation coefficient ranges from -1 to 1, where values close to 1 indicate a strong positive relationship, values near -1 represent a strong negative relationship and values close to 0 suggest no linear relationship.

3.3.2 Mutual Information

Mutual information is the measurement of how two variables mutually correlated with each other (*Venkatesh et al., 2019*). Mathematically,

$$\mathrm{I(m, n)} = \sum_{b \in B} \sum_{a \in A} p(m, n) \log \left(\frac{p(m, n)}{p(m) p(n)} \right) \qquad (2)$$

Where, $p(m, n)$ is the joint probability distribution of m and n, and $p(m)\ and\ p(n)$ are the marginal probability distribution of m and n.

3.3.3 Variance Threshold

Variance Threshold is a simple and commonly used feature selection technique that removes low variance features from a dataset. The assumption behind this method is that features with very little variance across the samples are unlikely to contain useful information for distinguishing between classes or predicting the target variable. By specifying a threshold, features with variance below this threshold are eliminated (*McHugh et al., 2008*).

3.3.4 Chi-Square

Chi-Square (*McHugh et al., 2008*) is a statistical test used to measure the association between categorical variables. Mathematically, it can be written as,

$$\chi^2 = \sum \frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}} \quad (3)$$

It calculates the discrepancy between the observed and expected outcome for each category of data.

3.4 Prediction Model

In this stage, this paper used Dense Neural Network to train the dataset and classify them into Defective and Non-defective. We Applied grid search to identify the best parameters for our Dense Neural Network. The list of parameters used in our work is shown in Table 3.

Table 3
Parameter List for Dense Neural Network

Parameters	Value
Neurons	64
Dropout_rate	0.3
Batch_size	32
Epoch	30
Optimizer	adam

4. Result analysis

This section shows the performance of software defect prediction using Dense Neural Network in brief manner. This experiment was conducted on an 8GB RAM PC on Google Colaboratory using Python Programming Language. Experimental dataset was collected and processed for neural network analysis.

4.1 Evaluation Metrics

In order to evaluate the performance of our proposed approach, we used a number of evaluation metrics, i.e., accuracy, precision, recall and ROC-AUC. ROC-AUC plots the True Positive Rate or Recall against the False Positive Rate (FPR). AUC indicates the model's performance to distinguish between the classes.

4.2 Classification Result

For the classification purpose, we have used five datasets named MC1, PC1, PC2, PC3 and PC4 from NASA MDP and three datasets named LC, JDT and ML from AEEEM dataset repository. Performance evaluation metrics for both of them are shown from Table 4 to Table 7. Table 4 and Table 5 presents a comparative analysis of different CLNI techniques, ENN, RENN and All-KNN evaluated with feature selection and without feature selection respectively across three performance metrics: Accuracy, precision and Recall for MC1, PC1, PC2, PC3 and PC4 dataset.

Table 4
Performance comparison between different CLNI methods using NASA MDP dataset without feature selection

	Accuracy			Precision			Recall		
	ENN	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**
MC1	0.94	0.94	0.93	0.94	0.93	0.93	0.94	0.93	0.92
PC1	0.90	0.90	0.91	0.90	0.92	0.92	0.90	0.92	0.91
PC2	0.93	0.92	0.91	0.91	0.92	0.93	0.91	0.94	0.92
PC3	0.83	0.84	0.83	0.85	0.83	0.82	0.83	0.83	0.84
PC4	0.89	0.90	0.90	0.89	0.90	0.90	0.90	0.90	0.90

Table 5
Performance comparison between different CLNI methods using NASA MDP dataset with feature selection

	Accuracy			Precision			Recall		
	ENN	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**
MC1	0.98	0.99	0.98	0.98	0.98	0.98	0.98	0.99	0.98
PC1	0.95	0.95	0.96	0.94	0.95	0.96	0.95	0.95	0.97
PC2	0.98	0.98	0.97	0.97	0.97	0.97	0.98	0.98	0.98
PC3	0.87	0.87	0.88	0.87	0.86	0.88	0.87	0.87	0.88
PC4	0.93	0.93	0.95	0.93	0.93	0.95	0.95	0.94	0.95

Across all five datasets, RENN consistently achieves either the highest or comparable performance in all three metrics, accuracy, precision, and recall that indicating its effectiveness as an efficient noise-handling technique. All-KNN also performs competitively, surpassing RENN in datasets like PC1 and PC4, particularly in precision and accuracy. In contrast, ENN, generally ranks slightly below RENN and All-KNN across most datasets, especially in datasets like PC3, where its Precision drops. Overall, RENN appears to be the most balanced and reliable technique across diverse datasets, while All KNN shows strengths in select cases, and ENN maintains moderate but consistent performance. Dataset specific variation, especially in PC3,

highlights the importance of utilizing the noise identification method to dataset characteristics.

Table 6 and Table 7 presents a comparative analysis of different CLNI techniques, ENN, RENN and All-KNN evaluated with feature selection and without feature selection respectively across three performance metrics: accuracy, precision and recall for ML, JDT and LC dataset. By analyzing the performance of all datasets, it is observed that PC3 dataset underperforms due to is limited number of, highlighting the importance of having a sufficient amount of training data.

Table 6
Performance comparison between different CLNI methods using AEEEM dataset without feature selection

	Accuracy			Precision			Recall		
	ENN	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**
LC	0.93	0.90	0.90	0.89	0.91	0.91	0.90	0.89	0.90
ML	0.90	0.90	0.89	0.90	0.89	0.88	0.90	0.87	0.88
JDT	0.89	0.90	0.88	0.90	0.88	0.88	0.93	0.88	0.88

Table 7
Performance comparison between different CLNI methods using AEEEM dataset with feature selection

	Accuracy			Precision			Recall		
	ENN	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**	**ENN**	**RENN**	**All-KNN**
LC	0.98	0.95	0.94	0.98	0.95	0.94	0.97	0.94	0.95
ML	0.92	0.93	0.90	0.92	0.92	0.90	0.92	0.90	0.90
JDT	0.91	0.94	0.90	0.91	0.95	0.90	0.90	0.92	0.90

Among the evaluated CLNI techniques, ENN consistently demonstrates balanced and high performing results. This trend is observed across all datasets. It maintains a strong performance in terms of accuracy, precision, and recall. ENN proves to be the most efficient. RENN performs competitively, particularly excelling in precision, but shows slight tradeoffs in accuracy and recall. All-KNN, while perform yielding acceptable results, generally underperforms relative to ENN and RENN, indicating less efficient in handling noisy data. On the other hand, the LC dataset shows higher conflict and superior results across all methods, suggesting it is less affected by noise, whereas JDT and ML appear more sensitive to the choice of noise-filtering techniques.

Figure 4 and Figure 5 represents the comparative bar chart representation of the performance of various CLNI techniques on the dataset.

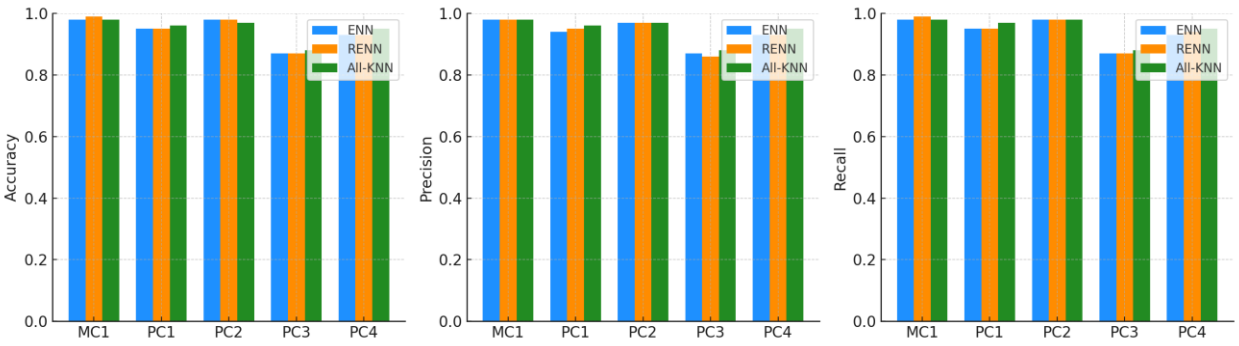

Figure 4: bar graph representation of the performance of the CLNI techniques for NASA MDP dataset

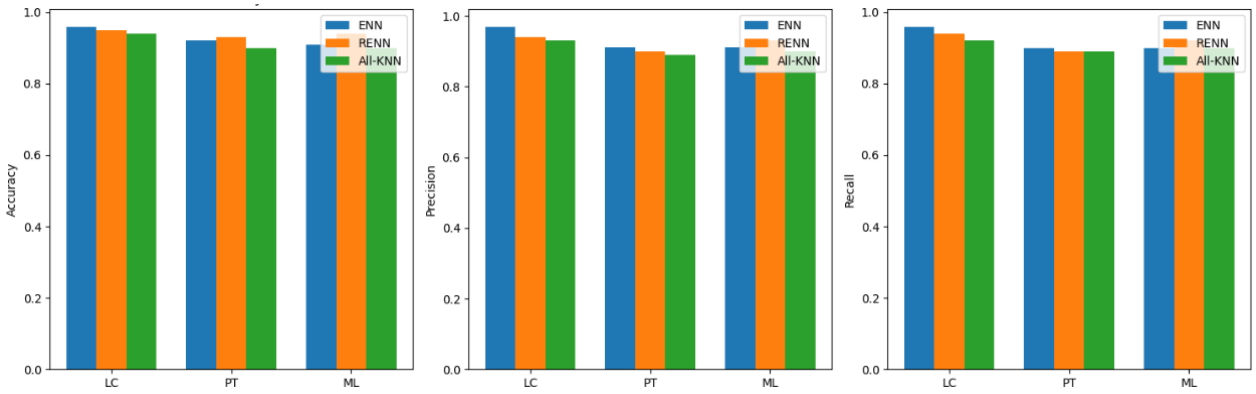

Figure 5: bar graph representation of the performance of the CLNI techniques for AEEEM dataset

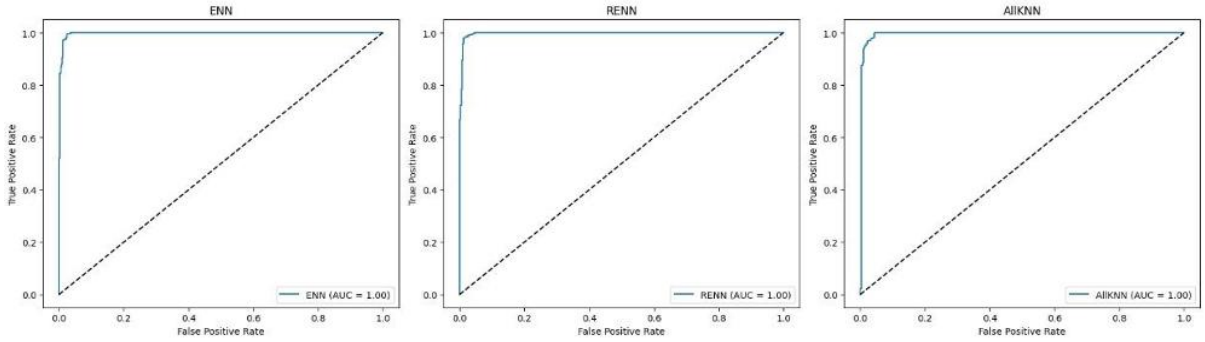

Figure 6: ROC-AUC curve for MC1 dataset

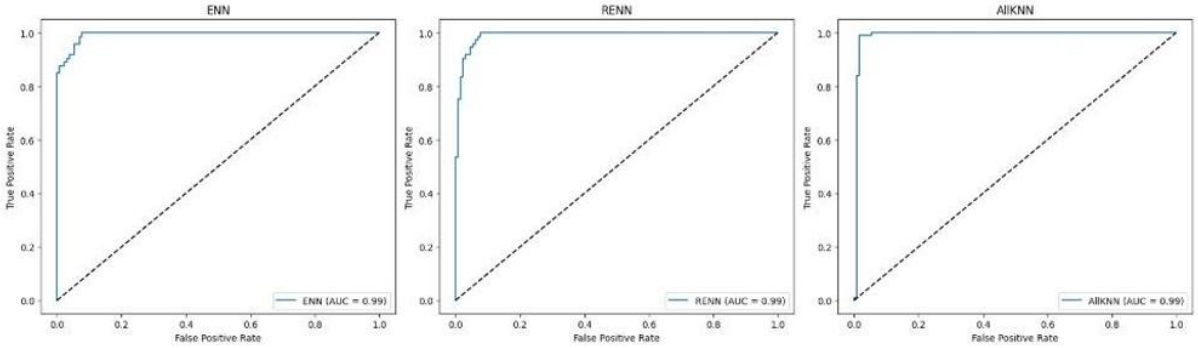

Figure 7: ROC-AUC curve for PC1 dataset

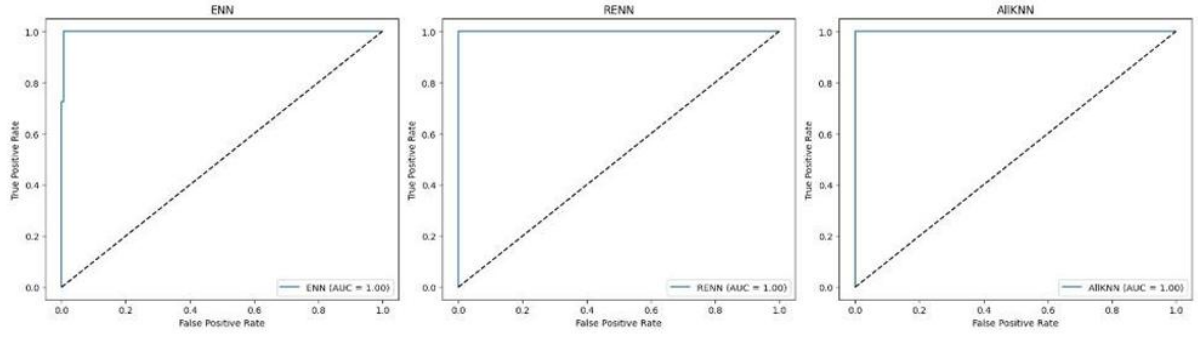

Figure 8: ROC-AUC curve for PC2 dataset

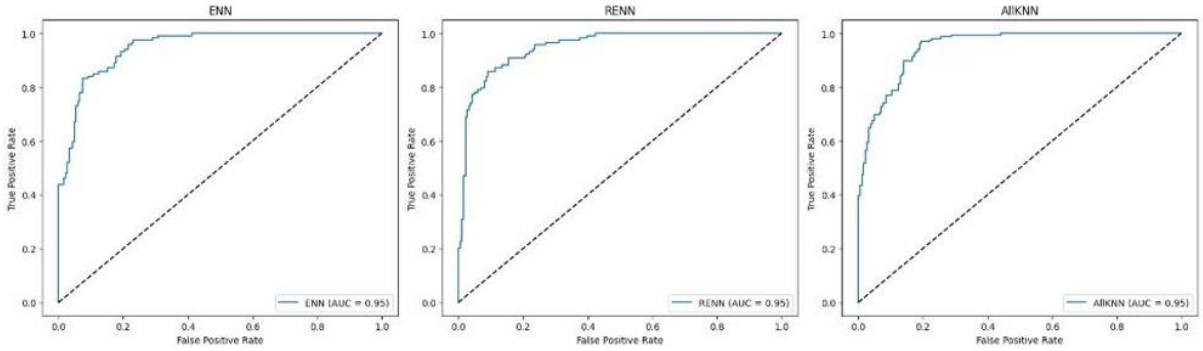

Figure 9: ROC-AUC curve for PC3 dataset

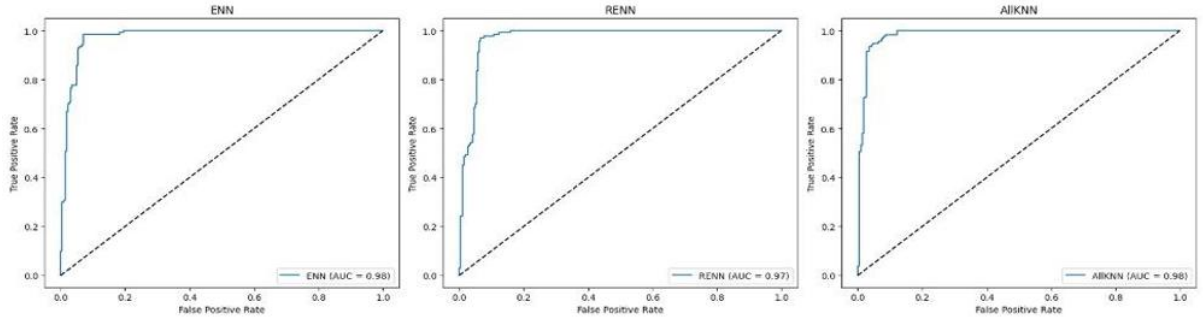

Figure 10: ROC-AUC curve for PC4 dataset

We further analyzed our results in terms of ROC-AUC curve. Figure 6 to Figure 10, all three methods, ENN, RENN, and All-KNN - show excellent performance on all datasets. For MC1 and PC2, the models reach a perfect score (AUC = 1.00), meaning they correctly separate the two classes every time, with no mistakes. The curves stay very close to the top-left corner, showing high accuracy with almost no false alarms. On the PC1 dataset, all methods score slightly below perfect (AUC = 0.99), still showing great results. ENN and RENN have smooth curves with very few errors, while All-KNN rises quickly, showing it catches most of the correct cases early. For PC3, the AUC drops to (0.95) for all three. This is still strong, but not as perfect as the others. It means the models are good but make a few more mistakes. All methods perform about the same here. The PC4 dataset has slightly lower scores: ENN (0.98), RENN (0.97), and All-KNN (0.96). ENN performs the best, while All-KNN is a bit lower, because it removes too much data when cleaning. In summary, all methods work very well, especially on MC1 and PC2. ENN is slightly more consistent across all datasets, making it a reliable choice.

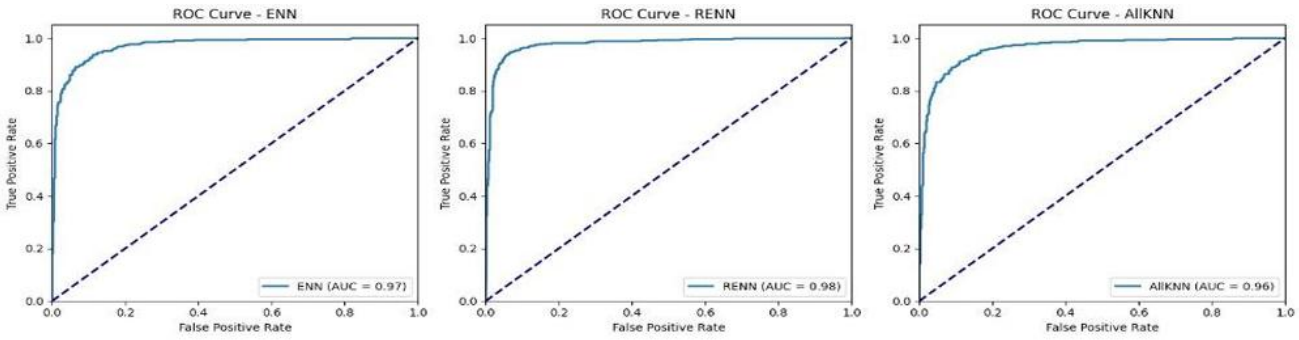

Figure 11: ROC-AUC curve for ML dataset

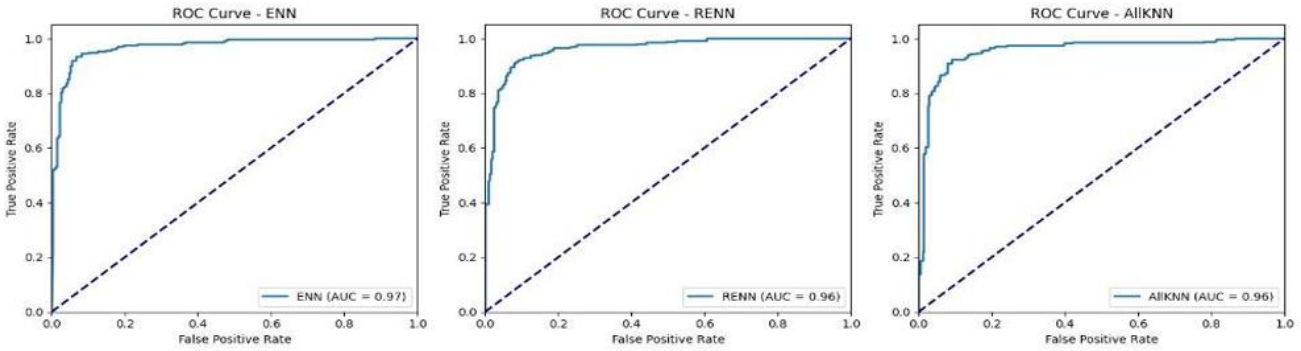

Figure 12: ROC-AUC curve for JDT dataset

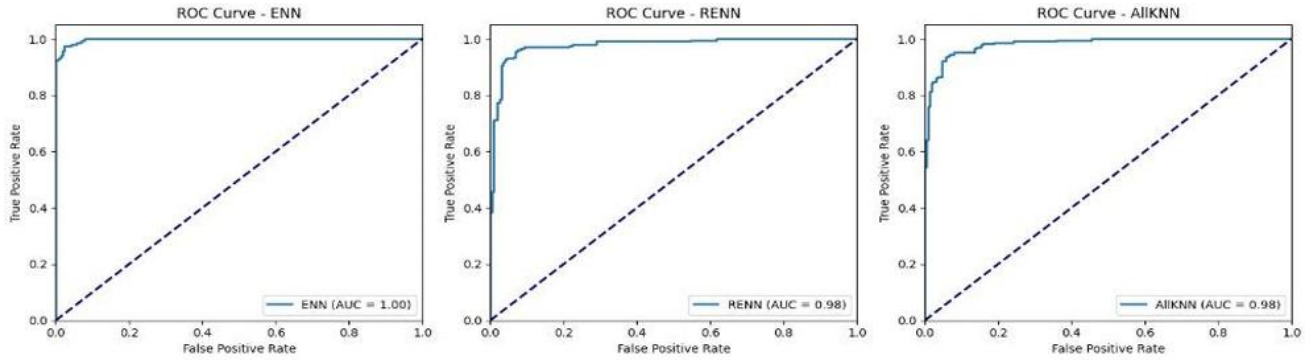

Figure 13: ROC-AUC curve for LC dataset

By analyzing the ROC-AUC curve for ML, JDT and LC dataset, it is observed that ENN consistently demonstrates a slight edge in performance, particularly on the LC and JDT datasets. On the LC dataset, ENN stands out with a perfect AUC of (1.00), clearly outperforming both RENN and All-KNN, which still achieve excellent scores of (0.98). In contrast, the ML dataset sees RENN take the lead with an AUC of (0.98), narrowly outperforming ENN (0.97) and All-KNN (0.96). For the JDT dataset, ENN once again leads at (0.97), while RENN and All-KNN both follow closely at (0.96). Through the performance gaps across methods are relatively small, ENN shows more consistent top-tier results across datasets.

4.3 Comparison with existing works

Table 8 presents a comparative analysis of various classification techniques applied to multiple datasets, focusing on the impact of balancing techniques, CLNI methods, and feature selection strategies. The results indicate that these preprocessing strategies contribute significantly in improving classification accuracy across different datasets.

Table 8
Comparison of Different Techniques on Various Datasets

Ref	Balancing technique	CLNI technique	Dataset	Repository	Feature Selection Technique	Model	Result (Accuracy)
Ali et al. (2024)	-	-	CM1	NASA MDP	-	Voting Ensemble	86.87%
			JM1				79.12%
			MC2				68.42%
			MW1				99.33%
			PC1				82.16%
			PC3				87.17%
Proposed work	SMOTE	ENN	MC1	NASA MDP	Correlation Mutual Information Variance Threshold Chi-Square	Dense Neural Network	99.02%
		All-KNN	PC1				97.42%
		All-KNN	PC2				99.63%
		ENN	PC3				88.58%
		RENN	PC4				95.34%
		ENN	ML	AEEEM			91.42%
		All-KNN	JDT				94.53%
		All-KNN	LC				95.43%

9. CONCLUSIONS

Software defect prediction focuses on identifying faulty modules at the early stages of the software development life cycle. Developing an efficient and effective defect prediction model is crucial for reducing maintenance costs and improving software quality. In this research, we proposed a software defect prediction approach that integrates SMOTE with CLNI to achieve a more balanced dataset and improved prediction accuracy. The datasets are collected from NASA MDP and AEEEM. The model was evaluated using eight datasets with relevant features to assess its effectiveness. The results demonstrate that the proposed classification model yields promising performance compared to traditional approaches. However, when applied to different datasets, it is observed that the PC3 dataset underperforms due to its limited number of samples, highlighting the importance of having a sufficient amount of training data. Relying on a predefined feature selection method may limit the model's adaptability across diverse datasets. In future work, we aim to overcome this limitation by exploring uniform and adaptive feature selection strategies and evaluating the approach on a broader range of datasets.

ACKNOWLEDGMENTS

The authors express their heartfelt gratitude to Bangladesh Army University of Science and Technology, Bangladesh; Military Institute of Science and Technology, Bangladesh; East West University, Bangladesh; Daffodil International University, Bangladesh, and Quantum Robotics and Automation Research Group (QRARG), Bangladesh.

DATA AVAILABILITY STATEMENT

Datasets generated during the current study are available from the corresponding author upon reasonable request.

FUNDING DECLARATION

This research was self-funded.

ETHICS APPROVAL

This study is an engineering experimental investigation. The MIJST Research Ethics Committee has confirmed that formal ethical approval was not required.

ETHICS, CONSENT TO PARTICIPATE, AND CONSENT TO PUBLISH

Not applicable.

COMPETING INTERESTS

The authors declare that they have no competing interests.

AUTHOR CONTRIBUTIONS

Author 1: Ahmmed Bin Ashfaque- Writing: Original draft, Review and editing, Formal analysis, Software, Validation, Visualization, Supervision

Author 2: Abdus Sattar2- Writing: Original draft, Review and editing, Formal analysis, Software, Validation, Visualization, Supervision

Author 3: Hosney Jahan- Writing: Original draft, Review and editing, Formal analysis, Software, Validation, Visualization, Supervision

Author 4: M. Akhtaruzzaman- Writing: Original draft, Review and editing, Formal analysis, Software, Validation, Visualization, Supervision

Author 5: Fernaz Narin Nur- Writing: Original draft, Review and editing, Formal analysis, Software, Validation, Visualization, Supervision

ARTIFICIAL INTELLIGENCE ASSISTANCE STATEMENT

Portions of this manuscript were assisted by an artificial intelligence language model (ChatGPT, OpenAI). The tool was used solely for language editing, text refinement, and clarity improvement. All content, data interpretation, analysis, conclusions, and final decisions were generated, verified, and approved by the authors. The authors take full responsibility for the accuracy and integrity of the manuscript.

CONFLICT OF INTEREST DECLARATION

The authors declare that they have no conflicts of interest.

REFERENCES

Akintola, A. G., Balogun, A. O., Lafenwa-Balogun, F., & Mojeed, H. A. (2018). Comparative analysis of selected heterogeneous classifiers for software defects prediction using filter-based feature selection methods. *FUOYE Journal of Engineering and Technology, 3*, 134–137.

Ali, M., Mazhar, T., Al-Rasheed, A., Shahzad, T., Yazeed Yasin Ghadi, & Muhammad Amir Khan. (2024). Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning. *PeerJ Computer Science, 10*, e1860–e1860. https://doi.org/10.7717/peerj-cs.1860

Ali, M., Mazhar, T., Arif, Y., Shaha Al-Otaibi, Yazeed Yasin Ghadi, Shahzad, T., Muhammad Amir Khan, & Habib Hamam. (2024). Software defect prediction using an intelligent ensemble-based model. *IEEE Access*, 1–1. https://doi.org/10.1109/access.2024.3358201

Alkhawaldeh, I. M., Albalkhi, I., & Naswhan, A. J. (2023). Challenges and limitations of synthetic minority oversampling techniques in machine learning. *World Journal of Methodology, 13*(5), 373–378. https://doi.org/10.5662/wjm.v13.i5.373

Cetiner, M., & Sahingoz, O. K. (2020, July 1). A comparative analysis for machine learning based software defect prediction systems. *IEEE Xplore*. https://doi.org/10.1109/ICCCNT49239.2020.9225352

Elreedy, D., & Atiya, A. F. (2019). A comprehensive analysis of synthetic minority oversampling technique (SMOTE) for handling class imbalance. Information Sciences, 505, 32–64.

Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., & Zhang, M. (2021). COSTE: Complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction.

Information and Software Technology, 129, 106432. https://doi.org/10.1016/j.infsof.2020.106432

Gupta, M., Rajnish, K., & Bhattacharjee, V. (2023). Software fault prediction with imbalanced datasets using SMOTE-Tomek sampling technique and genetic algorithm models. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-023-16788-7

J, A. A., & Judith, J. E. (2023). Enhanced deep learning approach for software defect forecasting. 1–7. https://doi.org/10.1109/aicera/icis59538.2023.10419998

Khleel, N. A. A., & Nehéz, K. (2024). Software defect prediction using a bidirectional LSTM network combined with oversampling techniques. *Cluster Computing, 27*(3), 3615–3638. https://doi.org/10.1007/s10586-023-04955-9

Mafarja, M., Thaher, T., Al-Betar, M. A., Too, J., Awadallah, M. A., Abu Doush, I., & Turabieh, H. (2023). Classification framework for faulty-software using enhanced exploratory whale optimizer-based feature selection scheme and random forest ensemble learning. *Applied Intelligence*. Advance online publication. https://doi.org/10.1007/s10489-022-04427-x

McHugh, M. L. (2008). The Chi-square test: An introduction. *Biochemia Medica, 18*(2), 112–118. https://www.researchgate.net/publication/5856449_The_Chi-square_test_an_introduction

Rathore, S. S., Chouhan, S. S., Jain, D. K., & Vachhani, A. G. (2022). Generative oversampling methods for handling imbalanced data in software fault prediction. *IEEE Transactions on Reliability, 71*(2), 747–762. https://doi.org/10.1109/TR.2022.3158949

Schober, P., Boer, C., & Schwarte, L. A. (2018). Correlation coefficients: Appropriate use and interpretation. *Anesthesia & Analgesia, 126*(5), 1763–1768. https://doi.org/10.1213/ANE.0000000000002864

Venkatesh, B., & Anuradha, J. (2019). A review of feature selection and its methods. *Cybernetics and Information Technologies, 19*(1), 3–26. https://doi.org/10.2478/cait-2019-0001

Vuttipittayamongkol, P., & Elyan, E. (2020). Neighbourhood-based undersampling approach for handling imbalanced and overlapped data. *Information Sciences, 509*, 47–70. https://doi.org/10.1016/j.ins.2019.08.062

Zhao, L., Shang, Z., Zhao, L., Qin, A., & Tang, Y. Y. (2018). Siamese dense neural network for software defect prediction with small data. *IEEE Access, 7*, 7663–7677. https://doi.org/10.1109/access.2018.2889061