Global Scientific JOURNALS

GSJ: Volume 12, Issue 11, November 2024, Online: ISSN 2320-9186
www.globalscientificjournal.com

SOFTWARE ERROR PREDICTION MODEL USING DEEP NEURAL NETWORK

Salman Ibrahim Marafa[1,*], Wurie Barrie Amadu[2] and Funsho Ahmed Dauda [2]

[1] Department of software engineering, Nankai University, Tianjin, China; marafasalman@gmail.com
[2] Department of software engineering, Nankai University, Tianjin, China; codewithbawurie@gmail.com
[3] Networking department, the Nigerian television Authority, Ilorin, Nigeria; fard4trust2002@gmail.com
* Marafa S.I, marafasalman@gmail.com; Tel.: (+86-13752034504)

Article information

Keywords:
ANN, RF, SDP, SVM, RF, DS

Abstract

Software error Prediction is an important aspect in software development and maintenance processes with an objective of locating and fixing defects ahead of schedule that could be expected under diverse circumstances. Predicting software errors especially in earlier phase improves the software quality, reliability, efficiency also the software testing cost. Many software development activities are performed by various individuals, which may lead to occurrence of different software errors, causing disappointments later during use Various software defect prediction (SDP) approaches that rely on software metrics have been proposed in the last two decades. Bagging, support vector machines (SVM), decision tree (DS), and random forest (RF) classifiers are known to perform well to predict defects. This paper presents a software error prediction model based on Artificial Neural Network algorithm (ANN). The evaluation process showed that error back propagation algorithm can be used effectively with high accuracy, precision and f-score rate. Furthermore, a comparison measure is applied to compare the proposed prediction model with other approach. The collected results showed that the (ANN) approach has a better performance.

1. Introduction

Software defect prediction plays a crucial role in software engineering by enhancing quality and ensuring timely, cost-effective development. It is applied before the testing phase of the software development life cycle. Numerous researchers have focused on developing various models, both within and across projects, to improve software quality and monitoring efficiency. There are two approaches that can be used to build a software defect prediction model that is supervised learning and unsupervised learning. Supervised learning has the problem that, to train the software defect prediction model, there is need for a historical data or some known results, while on the other hand, unsupervised learning analyze and cluster unlabeled data sets and discover a hidden pattern or data groupings. There are many public datasets

which are available, free for the researchers like PROMISE, Eclipse and Apache to overcome the challenging problem of training data performed on new project (Rakesh Rana., 2015)

Software defect prediction methods leverage past software metrics and fault data to identify faulty modules in upcoming software releases. To meet software quality assurance goals, quality models serve as valuable tools for detecting flawed program modules. A software defect is a coding error that can cause the software to behave unexpectedly, potentially leading to errors or failures. These defects, often caused by programming mistakes, can sometimes result in recoverable faults.

Once such faults are detected, they are sent to appropriate handler for performing required steps. Defects prediction will give an opportunity to the team to retest again the modules or files which are faulty or for which the probability of defectiveness is more. Focusing more on defective modules rather than non-defective ones ensures efficient use of resources, making project maintenance significantly easier such that, it is beneficial for both software users as well as project owners. The exact prediction of where the defects are probable to occur in code can help directly to test effort, enhance the software quality and reduce costs. A fault susceptible module is the one in which the quantity of faults is higher than selected threshold. Today the greatest challenge in the software industry is to make any application or software completely fault free to achieve the best software. As the defected modules are recognized, it is easier for the experts to focus only on the development work.

The different types of software metrics like class level, method level, file level, process level are used to find fault in the software, before the testing of software at early stage of software developments life cycle. There are different methods which are used to find the software defects, the methods are statistical method, machine learning method, and expert analysis system (Iker Gondra, 2008). The number of faults in the software causes problems in system performance and the software containing many faults delivered to the user. So, there is need for an automated model which takes less time and predict the approximate faults existing in the system

1.1 Artificial Neural Network

ANN is a machine learning method based on a model that can be used for classification. An ANN model is composed of multiple layers of interconnected units known as neuron. Training the ANN model with a set of data with known labels, the ANN model can learn to predict the values of unknown data. An Artificial neural network has mainly three layers: an input layer, an output layer, and an intermediate or hidden layer. The input layer neurons distribute the input signals x_1 to neurons in the hidden layer(s). Each hidden layer neuron j sums up its input signals x_i after weighing them with the strengths of the respective connections X_{ji} from the input layer.

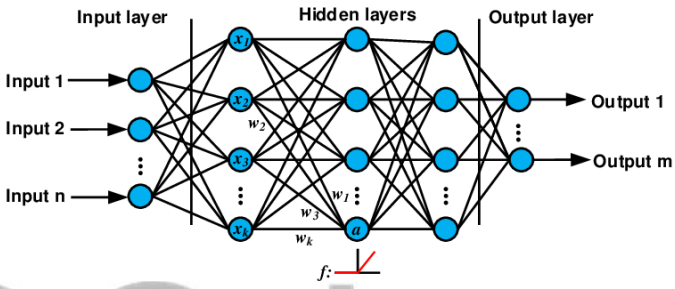

Fig 1. An image showing an Artificial Neural Network (ANN) with several intermediate and hidden layers respectively

In Figure 1, If x and w are the activation and the learned weight of the previous layer that are connected to a node, the output (activation) of that node would be $a = f\left(\sum_{j=1}^{k} w_j x_j + b\right)$. In this equation, b is a learned constant bias and f is a non-linear function e.g. ReLU, sigmoid.

1.2 Statement of problem

As the dependence on software for managing and executing daily operations continues to rise, the emphasis on software quality becomes increasingly crucial. This is especially relevant as many organizations are now aligning their business goals with enhancing customer satisfaction and fostering profitable growth through expanded software utilization. In this context, even a minor flaw or inefficiency in the software can lead to substantial financial losses, potentially in the millions of naira and damage customer retention, highlighting the critical need for maintaining strong software quality.

2.0 Related work

The common software defect prediction process follows both statistical and machine learning approach. But recently, studies have focused more on machine learning and its distinctive performance as compared to statistical approach.

According to (Xin Fan, Liangjue Lian, Li Yu, Wei Zheng, Yun Ge, 2024), it was opined that target projects often lack sufficient data, which affects the performance of the transfer learning model and also the presence of uncorrelated features between projects can lead to a reduction in the prediction accuracy of the transfer learning model. A software defect prediction system based on stable learning (SDP-SL) that combines residual networks and code visualization techniques was introduced to address these issues, using code visualization techniques, this method first converts code files into code images, which are then used to build a defect prediction mode. Finally, the performance evaluation showed that it outperformed other within-project and cross-project models, therefore, can effectively enhance defect prediction.

In (Misbah Ali, Tehseen Mazhar, Yasir Arif, Shaha Alotaibi et al., 2024), an ensemble-based prediction model for software defects that integrates various classifiers was developed, the model used a two-step prediction procedure to identify faulty modules. In the first stage, Random Forest, Support Vector Machine, Naïve Bayes, and Artificial Neural Network were the four supervised machine learning algorithms used. To attain the maximum accuracy feasible, these algorithms underwent an iterative parameter optimization. Furthermore, in the second stage, the predictive accuracy of the individual classifiers was integrated into a voting ensemble to make the final predictions. In conclusion, a comparative analysis with twenty state-of-the-art techniques was carried out to establish the effectiveness of the proposed framework and the result demonstrated that a remarkable accuracy was achieved as compared to the aforementioned state-of-the-art techniques.

The study in (Bhaskar Marapelli, Anil Carie, and Sardar Islam, 2023) assessed the existing dataset used for software prediction models and suggested that there was problem of noisy attributes and class imbalance. To solve this, ROS-KPCA-SG model (Random Over Sampling-Kernel Principal Component Analysis Staked Generalization Model) model was proposed to solve the problem. The performance of the model was compared with individual models with different combinations of sampling techniques. The results showed the proposed ROS-KPCA-SG model solves the problems and gave better performance than other models. For instance, it was observed that, the AUC-ROC score is between 0.9 and 1 for the ROS-KPCA-SG model on all the datasets, and the accuracy is near to 90% and above which was a higher value than other models compared.

(Bahman Arasteh, Sahar Golshan, Shiva Shami, and Farzad Kiani. Sahand, 2024) Discussed the concept of reducing the time needed to create software defect predictor. By using a hybrid method combining the auto-encoder and the K-means algorithm clustering error and time was lowered. In order to choose the training dataset's effective attributes and subsequently minimize its size, the auto-encoder algorithm was employed as a pre-processor and consequently led to a reduction in size of the datasets. In addition, Tests carried out on the standard NASA PROMISE datasets show that the suggested fault predictor has an improved accuracy of (96%) and precision of (93%), after eliminating the inefficient elements from the training data set. The suggested method yields a recall criterion of about 87%.

It is believed that some machine learning algorithms perform poorly with a small size of dataset, as an example, Naive Bayes's algorithm achieved a higher accuracy value when tested on a larger dataset as compared to a small dataset. (Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, and Fatima Alsarayrah, 2018) Confirmed this assertion by analyzing three supervised ML algorithms to predict software defects based on historical data. These classifiers which are; Naïve Bayes (NB), Decision Tree (DT) and Artificial Neural Networks (ANNs).ANN, however had the highest accuracy value, followed by DT and lastly NB

3.1 *Approach to Research*

This stage involves acquiring appropriate datasets that will be used in this research. Therefore, the datasets will be acquired from http://bug.inf.usi.ch/download.php which is publicly accessible for use.

The following stage is the feature selection stage which will be achieved by using Principal Component Analysis (PCA) in order to extract the relevant features from the datasets that will be collected in the first stage.

In this third stage, classification of the data will take place, the extracted features will be classified using error back propagation neural Network learning machine classifier. Finally, evaluation of the results will be done using precision, recall, accuracy, and f –score.

3.1.1 *Data Collection*

Software defect datasets often highlight specific components or modules within a system that are prone to defects. These defects can vary in granularity and manifest at different levels of the software architecture. Such datasets primarily derive information from software metrics and developer-related details. Software metrics are typically classified into three categories: requirements metrics, product metrics, and process metrics. Examples of metrics used in software defect analysis include measures of software engineer productivity, as well as aspects of software design and maintenance. Some metrics are directly measurable, while others require analytical methods. For instance, in object-oriented (OO) software, design coupling metrics such as Coupling Between Objects (CBO), Response for a Class (RFC), Message Passing Coupling (MPC), and Information-Flow-Based Coupling (ICP) are critical for defect prediction. Additional important software metrics include Lines of Code (LOC), McCabe Cyclomatic Complexity (MCC), McCabe Essential Complexity (MEC), McCabe Module Design Complexity (MMDC), and Halstead metrics, (McCabee T, 1996). These metrics provide valuable insights into various aspects of software quality and defect likelihood.

The proposed methodology is applied on Aerospace Engineering and Engineering Mechanics (AEEM) suite dataset which is a software data repository collected by [2] and is comprised of metric and bugged data from four different open source projects (JDT: Eclipse JDT core, PDE: Eclipse PDE UI, EQ: Equinox Framework, Luc: Apache Lucence). Each dataset contains 62 software metrics including OO, previous defects, and change metrics. This dataset was created under a distinct setting compared to the NASA dataset and focuses on defect prediction at the class level. The Aerospace Engineering and Engineering Mechanics (AEEM) dataset includes features such as source code metrics, change metrics, entropy of source code metrics, and churn of source code metrics. The datasets could be acquired from http://bug.inf.usi.ch/download.php which is publicly accessible for use, additionally, the dataset is free of noise, and this noise refers to the incorrect software data caused by some unexpected reasons, such as unintentional errors in collecting or transferring the values of software features. The first data collection phase is to find defects from the repository and collect all valuable pieces of information.

3.2 *Feature Selection*

The research applies Principal Component Analysis (PCA) using a wrapper method for linear dimensionality reduction, which minimizes the number of features by identifying principal components (PCs) with the most significant variance. The goal of PCA is to retain as much information as possible while reducing dimensionality. The M is a t-dimensional data set. The n principal axes G1, G2... Gn here $1 \leq n \leq t$, are orthonormal axes onto which the retained variance is maximum in the projected space (] Peter N. Belhumeur, João P. Hespanha, and David J. Kriegman, 1996). Commonly G1, G2... Gn can be given by the n leading eigenvectors of the sample covariance matrix: These axes correspond to the leading eigenvectors of the sample covariance matrix C.

$$C=\left(1/l\right)\sum_{k=1}^{l}(x_k-\bar{x})^T(x_k-\bar{x}) \qquad 3.1$$

Here $x_k \in M$, $\bar{x}$ is the mean of samples, l is the number of samples. According to this:

$$UG_K = V_k G_k, K \in 1, \ldots\ldots\ldots\ldots n, \qquad 3.2$$

Here V_k is the k_{th} largest eigenvalue of U. The n principal components of a given observation vector $x_k \in M$ are given as below:

$$Q= [q_1, q_2, q_3.. q_n] = [[G_1^T x, G_2^T x, \ldots\ldots\ldots G_n^T x]=G^T x] \qquad 3.3$$

There, q is the n principal components of x

PCA is favored for its low sensitivity to noise, reduced memory requirements, elimination of data redundancy through orthogonal components, and improved processing time and classification accuracy. Additionally, PCA is an unsupervised method, making it adaptable for various applications.

3.3 *Methods for Updating the Weight Vector*

Levenberg Marquardt (LM) Method: The Levenberg-Marquardt (LM) method iteratively identifies the minimum of a multivariate function, expressed as the sum of squares of nonlinear real-valued functions. This approach is utilized for updating weights during the learning process. Compared to the gradient descent method, the LM method offers faster and more stable execution, as it combines the principles of steepest descent and Gauss-Newton methods. In LM method, weight vector W is updated as follows:

$$W_{k+1} = W_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \qquad 3.4$$

where W_{k+1} is the updated weight, W_k is the current weight, J is Jacobian matrix, and μ is combination coefficient; that is, when μ is very small then It functions as the Gauss-Newton method when μ is small, and as the Gradient Descent method when μ is significantly large. Jacobian matrix is calculated as follows:

$$J = \begin{bmatrix} \frac{\partial d}{\partial dW_1}(E_{1,1}) & \frac{\partial d}{\partial dW_2}(E_{1,1}) & \cdots & \frac{\partial d}{\partial W_N}(E_{1,1}) \\ \frac{\partial d}{\partial dW_1}(E_{1,1}) & \frac{\partial d}{\partial dW_2}(E_{1,2}) & \cdots & \frac{\partial d}{\partial W_N}(E_{1,2}) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d}{\partial dW_1}(E_{p,M}) & \frac{\partial d}{\partial dW_2}(E_{p,M}) & \cdots & \frac{\partial d}{\partial W_N}(E_{p,M}) \end{bmatrix}$$

3.3 *Cross Validation*

This technique is highly effective for assessing different combinations of feature selection, dimensionality reduction, and learning algorithms. A popularly used validation method is k-fold cross-validation. In this approach, the original training dataset is split into k separate subsets, referred to as "folds". One fold is set aside as the test set, while the remaining k-1 folds are used to train the model. For example, if k is set to 4 (i.e., 4 folds), three of the subsets would be used for training, and the fourth would be used for evaluation. Once all four iterations are complete, the average error rate and standard deviation are calculated, offering insights into the model's generalization performance.

3.4 *Performance Evaluation Measures*

The performance of software defect prediction models is evaluated using various measures, including;

True Positive (TP): TP represents the count of defective software instances that have been accurately identified as defective.

True negative (TN): TN is the number of clean software instances that are correctly classified as clean.

False Positive (FP): Refers to the number of non-defective software instances incorrectly classified as defective.

False Negative (FN): Refers to the number of defective software instances incorrectly classified as non-defective.

One of the simplest and most commonly used metrics for assessing the performance of predictive models is classification accuracy, also known as the correct classification rate. It measures the proportion of correctly classified instances relative to the total number of instances. Another important metric is precision, calculated as the ratio of correctly classified defective instances (TP) to the total instances classified as defective (TP + FP). Additionally, recall measures the proportion of correctly identified defective instances (TP) to the total defective instances (TP + FN). The F-score, which is the harmonic mean of precision and recall, is widely employed in literatures also as an evaluation metric.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \qquad 3.5$$

Accuracy takes into account both true positives and true negatives across all instances. In other words, it represents the proportion of instances that are correctly classified

$$Precision = \frac{TP}{TP+FP} \qquad 3.6$$

It is the ratio of the number of instances correctly identified as defective (TP) to the total number of instances classified as defective (TP + FP).

$$Recall = \frac{TP}{TP+FN} \qquad 3.17$$

It is the proportion of instances correctly identified as defective (TP) relative to the total number of faulty instances (TP + FN)

$$F - sccore = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \quad 3.15$$

The harmonic mean of precision and recall

3.5 *Evaluation of Classification Quality*

In this research, we adopted binary classification, which means that we predict whether a component/file has a defect or not. Table 1 below presents the confusion matrix, which is commonly used to evaluate the performance of binary classification models.

Table 1. Confusion matrix

		Defects are observed	
		TRUE	FALSE
Predicted Condition	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

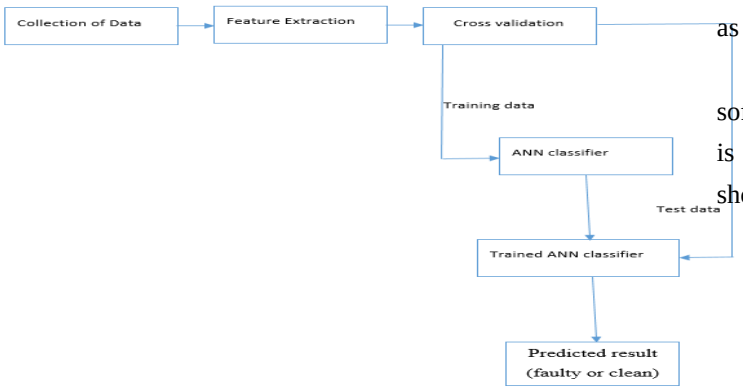

Figure 2 Architecture of the model

3.6 *Error Back-Propagation*

Error back-propagation learning involves two distinct phases: a forward pass and a backward pass. In the forward pass, the input is fed into the neural network and propagated through each layer, while keeping the network weights unchanged. During this phase, the network produces an output. In the backward pass, the error is calculated as the difference between the desired output and the actual output from the network. This error is then propagated backward through the network to update and adjust the weights based on the error values. This adjustment follows a systematic process governed by a learning rule. The training process starts with randomly initialized weights, and the objective is to fine-tune these weights to minimize the error. The adjustment process continues iteratively until the performance of the network reaches an acceptable level. The activation function used in the neurons of artificial neural networks that employ the back-propagation algorithm is typically a weighted sum of the inputs X_i, where each input is multiplied by its corresponding weight W_{ji}. This process ensures the gradual optimization of the network to improve predictive accuracy.

4.0 *Result and Discussion*

The experiment was conducted by extracting relevant features from the datasets, as outlined in the methodology, using principal component analysis. This step significantly impacts system error prediction, as iterative feature additions and modifications in software tend to increase system complexity. However, the machine learning algorithm used to build the software prediction system in this research is Artificial Neural Network (Levenberg Marquardt algorithm) classifier and the performance evaluation were determined. MATLAB (R2018a) was used as the development environment using statistical toolkit.

Moreover, the "weighted-ent" dataset has 22 software metrics with varying amount of instances. These is evident in the GUI representation of each data set as shown in fig 3, 4, 5 and 6.

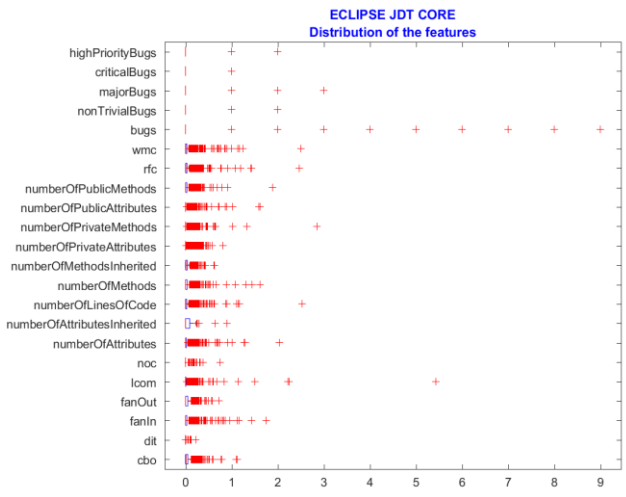

Fig 3 Distribution of eclipse JDT CORE showing 22 features and 9 instances

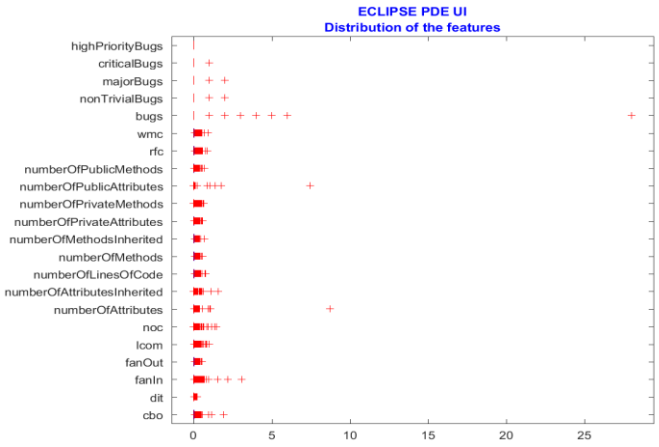

Fig. 4 The distribution of Eclipse PDE UI with a similar number of 22 features and 25 instances

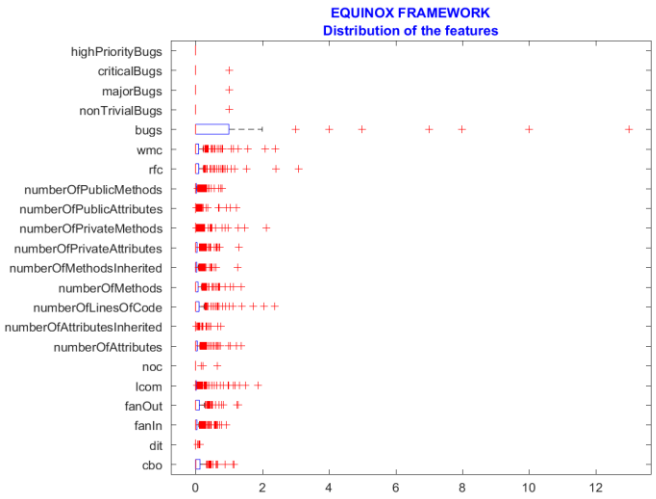

Fig. 5 The distribution of Equinox framework with 22 features and 12 instances

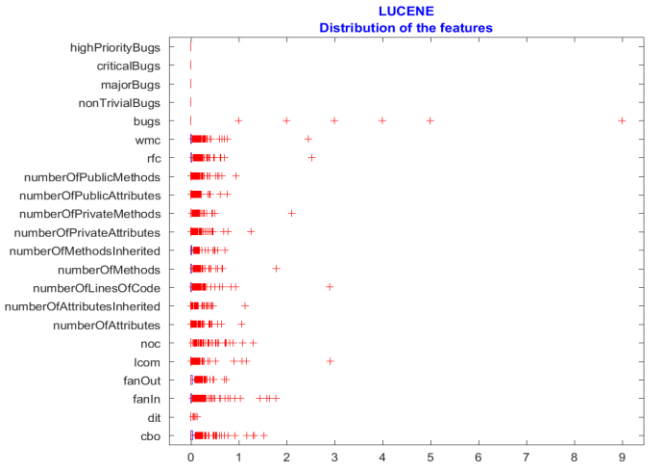

Fig. 6 The distribution of LUCENE with a similar number of 22 features and 9 instances

In order to eliminate the possibility of over fitting, feature selection was employed. PCA was used to extract the features that most affect the output which is the number of bugs in this case.

Four times 4-fold cross validation was performed when evaluating the prediction system. The datasets were separated into four equal parts. Three parts out of four are used as training data and the feature selection process while the forth part is used for testing. In order for every instances of the datasets to be used as testing and training, this process was repeated four times. Cross-validation was used due to the limited amount of data, offering an advantage over the holdout method. In the holdout method, the dataset is divided into two parts: one is used for training, and the other for testing. In this research, the solution to the bias idea was adopted using cross validation where all the instances were used one time for testing and training. This simply means that, instead of conducting four folds, a total of 16 folds is generated and the error estimate is therefore more reliable.

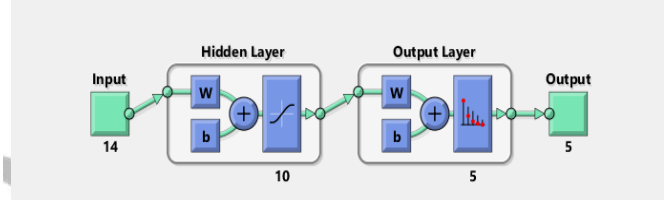

Figure 7 Architecture of Error back propagation neural network

Fig. 7 shows that the architecture used for error-back propagation algorithm with 14 neurons (i.e. features of dataset) in input layer. Hidden layer uses 10 neurons and 5 neurons in its output layer and eventual output respectively.

4.1 *Confusion Matrix*

Confusion matrix also referred to as an error matrix which reports the number of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) where TP is the correct positive prediction, FP is the incorrect positive prediction, TN is the correct negative prediction and FN is the incorrect negative prediction. Having conducted the experiments, the confusion matrix where the true conditions are the actual results, that is the original result from the file and predicted condition are the result from the predictors which are presented in Table 2.

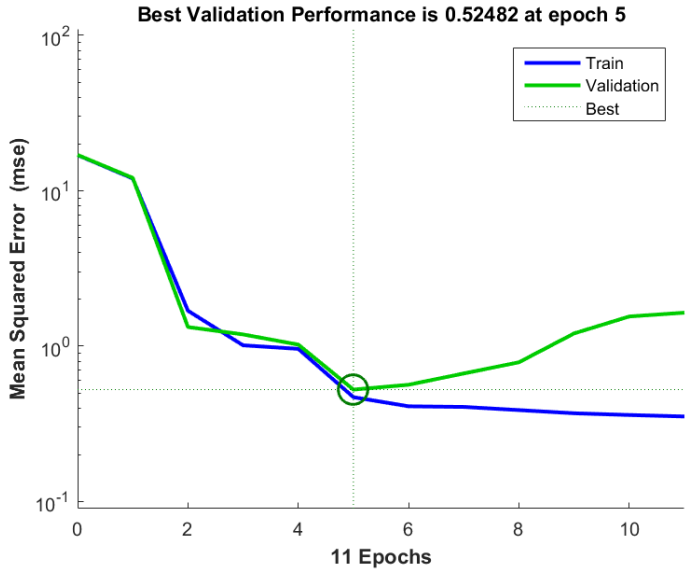

Fig. 8 Training and Validation for Eclipse JDT Core

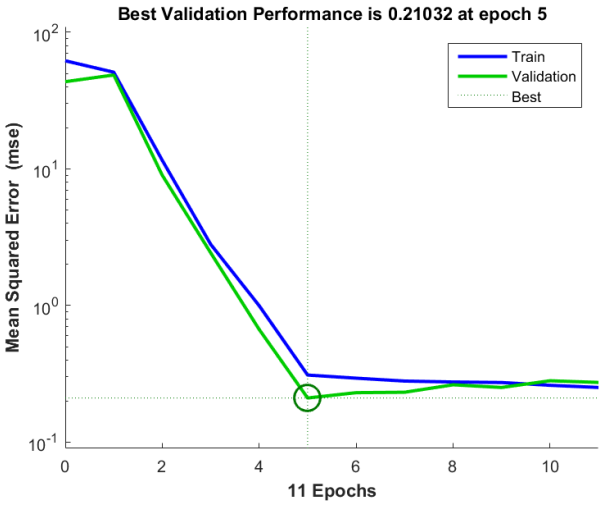

Fig. 9 Training and Validation for Eclipse PDE UI

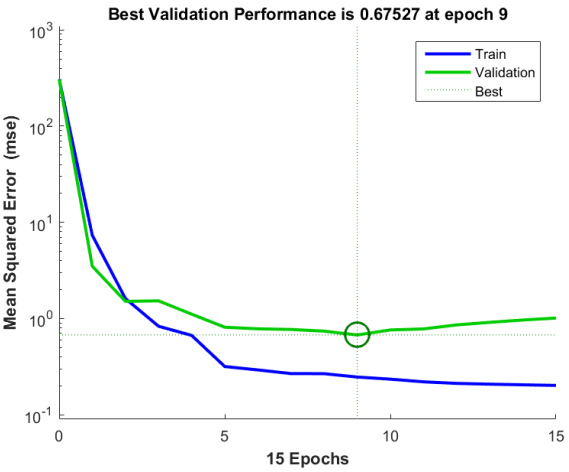

Fig. 10 Training and Validation for Equinox

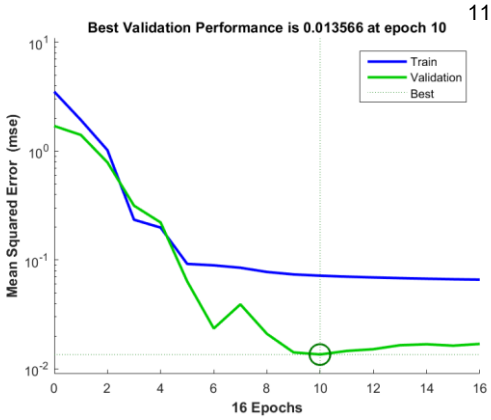

Fig. 11 Training and Validation for Lucene

4.2 *System Evaluation*

The system was measured in terms of accuracy, precision, recall and f-score. As shown in Table 1, accuracy is described as the ratio of all correctly classified instances. Precision measures the fraction of faulty modules that are actually faulty. It is a measure of how good the prediction system is at identifying actual faulty files. Recall measures the percentage of fault-prone modules that are classified correctly.

According to the conducted experiments the percentage of the True Positive Rate (TPR) and True Negative Rate (TNR) of the datasets used in this research work; ECLIPSE JDT CORE, ECLIPSE PDE UI, EQUINOX FRAMEWORK, and LUCENE are (79.31% and 88.24%), (45.45% and 87.97%), (65.22% and 73.81%) and (50.00% and 93.85%) respectively. The training and validation for the datasets ECLIPSE JDT CORE, ECLIPSE PDE UI, EQUINOX FRAMEWORK, and LUCENE was conducted. However, the best validation performance is 0.52482 at epoch 5, 0.21032 at epoch 5, 0.67527 at epoch 9 and 0.01356 at epoch 10 respectively.

Table 1 System Results of the Accuracy, Precision, Recall, and F-Score

Name	Accuracy	Precision	Recall	TPR	TNR	Balanced Accuracy	F1 Score
ECLIPSE JDT CORE	86.93%	53.49%	79.31%	79.31%	88.24%	83.78%	63.89%
ECLIPSE PDE UI	83.28%	31.91%	45.45%	45.45%	87.97%	66.71%	37.50%

EQUINOX FRAMEWORK	70.77%	57.69%	65.22%	65.22%	73.81%	69.51%	61.22%
LUCENE	91.30%	33.33%	50.00%	50.00%	93.85%	71.93%	40.00%

Table 2 Confusion matrix results

ECLIPSE JDT CORE			
	True Conditions		
Predicted Condition		True	False
	True	23 (TP)	20 (FP)
	False	6 (FN)	150 (TN)

ECLIPSE PDE UI			
	True Conditions		
Predicted Condition		True	False
	True	15	32
	False	18	234

EQUINOX FRAMEWORK			
	True Conditions		
Predicted Condition		True	False
	True	15	11
	False	18	31

LUCENE			
	True Conditions		
Predicted Condition		True	False
	True	4	8
	False	4	122

4.4 Comparison to Related Study

In Table 3, the implemented result of the proposed model is being illustrated and compared with those of classifiers such as in Rizwan *et al*. (2017) which made use of similar performance metrics. That is accuracy, f1-score, precision. It can be seen that the proposed model performed better than the model been compared with. It is further compared to the average accuracy result of the related study. Rizwan *et al.* (2017) proposed a system to predict defects specific to concurrent programs by combining both static and dynamic program metrics.

Table 3 Comparison of Rizwan's model against the Developed Model

Name	Accuracy	Precision	Recall	F-Score
Random Forest	18.70%	59.74%	37.16%	43.68%
Naïve Bayes	17.10%	47.04%	45.54%	43.62%
J48	43.58	49.38%	46.70%	12.20%
Proposed Model	83.07%	44.10%	59.99%	50.65%

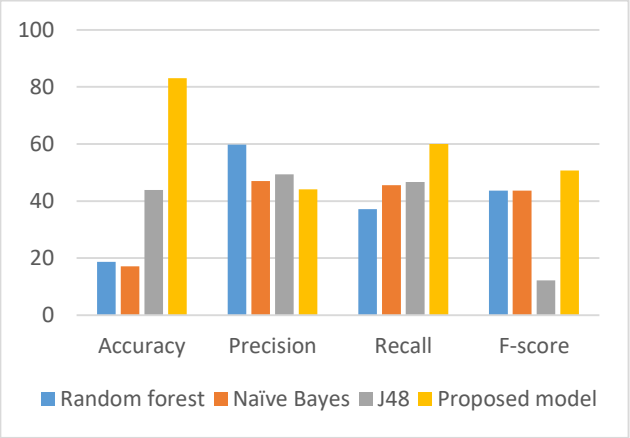

Figure 12 Comparison of Rizwan's model against the Developed Model

5.0 Summary and Conclusion

This research focused primarily on prediction of software error or bugs on one the most well-known machine learning algorithms that are used to predict software defects. The performances of the model algorithms was evaluated using classification accuracy, precision, recall and F-score. The cross-validation strategy was employed to deal with the bias issue. The outcomes of the conducted experiment showed that the proposed model performed well in terms of accuracy. However, there is room for more extensive research and study in this domain because, software development is ever evolving with new approaches and frameworks.so, and this may give rise to new errors in the future and thus a need for continuous optimization.

6.0 *Acknowledgment*

Special thanks to God almighty for the inspiration of this project and to colleagues and friends who supported us physically and emotionally when needed.

References

Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, and Fatima Alsarayrah Software bug prediction using machine learning approach. International journal of advanced computer science and applications, 9(2), 2018.

Bacchelli A., D'Ambros M., and Lanza M. (2010) Are popular classes more defect prone? In Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering, FASE'10, pages 59–73.

Bahman Arasteh, Sahar Golshan, Shiva Shami, and Farzad Kiani. Sahand: A software fault-prediction method using auto-encoder neural network and k-means algorithm. Journal of Electronic Testing, pages 1–15, 04 2024.

Bhaskar Marapelli, Anil Carie, and Sardar Islam. Software Defect Prediction Using ROS-KPCA Stacked Generalization Model, pages 587–597. 04 2023.

Iker Gondra. Applying machine learning to software fault-proneness prediction. Journal of Systems and Software, 81(2):186–195, 2008. Model-Based Software Testing.

McCabe T. and C. Butler, "Design Complexity Measurement and Testing," Communi-
cations of the ACM, December 1989

Misbah Ali, Tehseen Mazhar, Yasir Arif, Shaha Alotaibi, Yazeed Ghadi, Tariq Shahzad, Muhammad Khan, and Habib Hamam. Software defect prediction using an intelligent ensemble-based model. IEEE Access, PP: 1–1, 01 2024.

Peter N. Belhumeur, João P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. In Bernard Buxton and Roberto Cipolla, editors, Computer Vision — ECCV '96, pages 43–58, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

Rakesh Rana. Software Defect Prediction Techniques in Automotive Domain: Evaluation, Selection and Adoption. PhD thesis, 02 2015.

Rizwan, Syed & Wang, Tiantian & Xiaohong, Su & Shaikh, Salahuddin. (2017). Empirical Study on Software Bug Prediction. 55-59. 10.1145/3178212.3178221.

Xin Fan, Liangjue Lian, Li Yu, Wei Zheng, Yun Ge, et al. Software defect prediction method based on stable learning. Computers, Materials & Continua, 78(1), 2024.

Author's brief profile

Marafa .S. I (B.Tech) is currently a master's student at the department of software engineering, Nankai University, Tianjin China. His research areas include fault tolerance, Machine Learning, Optimization and software engineering. He can be reached by phone on (+86)-13752034504 and via email-marafasalman@gmail.com.

Amadu W.B (B.Sc.) is currently a master's student at the department of software engineering, Nankai University, Tianjin China. His research areas includes human computer interaction and software engineering. He can be reached by phone on (+86)-15922144231 and via email-codewithbawurie@gmail.com

Dauda .F.A (M.Tech) is an engineer at the Nigerian Television Authority, His research focus spans along machine learning, cyber security and optimization of telecommunication networks. He can be reached via (+234)-8085818815 and by email-fard4trust@gmail.com