JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Software Fault Prediction Using Deep Neural Networks

Satish Dekka[1], S Deepika[2], P Krishna Sai[3], K Gowthama Laxmi[4], S K Afreen Begam[5]

[1,2,3,4,5] Department of Computer Science and Engineering, Lendi Institute of Engineering and Technology, AP, INDIA.

ABSTRACT

Software systems have become larger and more complex than ever. Such characteristics make it be very challengeable to prevent software defects. Therefore, automatically predicting the number of defects in software modules is necessary and may help developers efficiently to allocate limited resources. Various approaches have been proposed to identify and fix such defects at minimal cost. However, the performances of these approaches require significant improvement. Therefore, we propose a novel approach that leverages deep learning techniques to predict the number of defects in software systems. First, we preprocess a publicly available dataset, including log transformation and data normalization. Second, we perform data modelling to prepare the data input for the deep learning model. Third, we pass the modelled data to a specially designed deep neural network-based model to predict the number of defects. We also evaluate the proposed approach on four well-known datasets. The evaluation results illustrate that the proposed approach is accurate and can improve upon the machine learning approach.

Keywords

Machine Learning, Deep Learning, Fault, Minimal cost, Predictive modelling

1.INTRODUCTION

A software fault is an error, bug, defect, flaw, malfunction or mistake in software that causes it to create a wrong or unexpected outcome. A software bug is an issue that prevents a system from carrying out its intended purpose. Software defects can cause different problems. Manual testing and code review are common methods for identifying software flaws. The primary disadvantage of these techniques is their high time and effort costs. The automatic approaches to the Software Fault Prediction would allow one to reduce the costs and improve quality of the software projects. Thus, Software Fault Prediction is an important problem in the fields of the software engineering and programming language research. The Finding the faulty code with great precision is the task. Software Fault Prediction: A approach called "software fault prediction" makes use of a model to identify likely defect-containing code sections. Predicting software faults is a crucial and advantageous technique for enhancing software dependability and quality. Fault prediction is one of the key challenges in software development. Software Fault Prediction is a critical problem in software development and maintenance procedures that affects the overall success of the product. Predicting and finding the bugs in the earlier phase in SDLC makes the software more reliable,

efficient and better quality when compared with finding bugs in the later stages. However, developing a software defect prediction model is not an easy task and manynew tools and methods are introducing in the deep learning for better performance. on the fault of defect-prone software

modules as compared to other techniques. Software with poor quality might lead to unexpected and incorrect outcomes. These days, it is harder togenerate high-quality software at a reduced cost due to the increasing complexity of current software systems.is issue can be solved by using the techniques of software fault prediction. This activity can improve the software quality by indicating the software modules in advance where faults are more likelyto occur. Prediction and prevention of software faults at the initial stages of software development canreduce the overall development time and cost by limiting the testing efforts. Artificial Neural Network (ANN) is a widely accepted supervised learning approach to deal with the prediction problems in multiple domains of software engineering such as effort estimation, cost estimation, and defect prediction. Three layers make up an ANN's structure: the input layer, the output layer, and the hidden layer (s). A connection exists from the nodes in the input layer with the nodes in the hidden layer and then from the nodes of the hidden layer with the nodes of the output layer. Through the input layer, input data are fed into the neural network. Classifiers in the supervised machine learning category require a training set of data (called training data) with a predetermined output class in order to function data set includes various features which are categorized as dependent and independent features.dependent feature is one which is going to be predicted, also known as the output class. Independent features are those that do not include the output class. The supervised classifier creates a classification model during training by identifying hidden patterns and relationships between the dependent and independent data. After training, the classifier is given test data, which has an unknown output class. Based on patterns and rules extracted from the training data, the classification model predicts the class of the test data. Software fault prediction is accomplished by classifying a specific software instance (method, class, module, file, andpackage) as either defective or non-defective data set used for softwarefault prediction includes the historical software defect data collected from previous releases of same project (or in some cases from other projects), which is made up of several qualities or characteristics, often known as metrics. Predictive techniques can assist the development team in making efficient use of their testing resources, hence reducing expenses and time.One widely adopted approach in software fault prediction is the use of Artificial Neural Networks (ANNs), a supervised learning method capable of handling prediction problems across various domainsof software engineering. ANNs consist of three layers: the input layer, the output layer, and one or morehidden layers. These layers are interconnected, allowing input data to be processed through the networkto generate predictions. During training, ANNs require a dataset containing both dependent (output class) and independent features, enabling the classifier to learn patterns and relationships.

Architecture of Neural Networks:

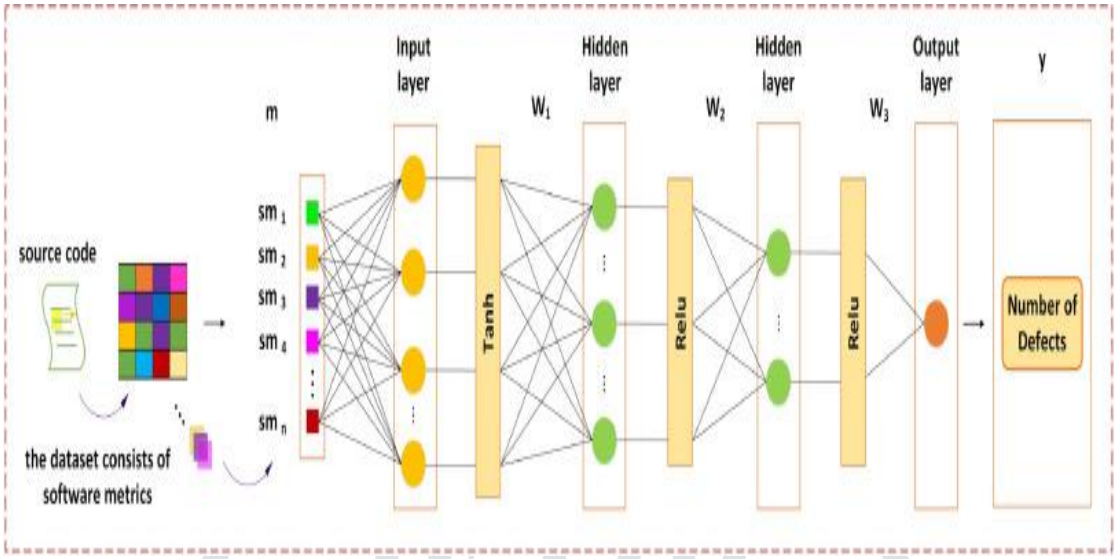

Fig.1 ***Architecture of Neural Networks***

The dataset used for software fault prediction typically comprises historical software defect data collected from previous releases of the same project or, in some cases, from other projects. This datasetincludes various metrics or qualities associated with software modules, such as methods, classes, modules, files, or packages. By leveraging predictive techniques, development teams can efficiently allocate testing resources, thereby reducing expenses and development time. In practice, software fault prediction involves classifying specific software instances as either defectiveor non-defective based on the learned patterns and rules extracted during training. By identifying potentially faulty modules early in the development process, teams can prioritize testing efforts and implement preventive measures to address potential issues before they manifest into critical defects. Overall, software fault prediction contributes to enhancing software quality, reducing development costs, and ensuring the reliability and stability of software systems in increasingly complex environments

2.LITERATURE SURVEY

In recent years, wide variety of deep learning models have been proposed and applied to different domains by researchers. However, in defect prediction context, to the best of our knowledge few works have been done which we will review in this section. .[1] “established a Selection of features for apps scheme the unequal dataset of software defects. Next, the selection with attributes built on the wrapper is implemented, culminating in the collection of subsets of attributes.[2] have “proposed a study for the prediction of software defects using a machine learning method focused on the neural network. Github databases are regarded in this work for the study of defect prediction

Machine learning is a powerful methodology for prediction, software defect prediction model proposed by Wang et al. [3] for increasing the quantity of application software systems. Databases of defective software comprise of unbalanced data which produces random patterns. This problem encourages the creation of an effective and reliable classifier of situations for academic and industrial applications. Xu et al. [4] researched “software defect prediction

strategies and hypothesized that traditional techniques use vectorization and feature selection" framework to minimize trivial features, but still exclude other essential features resulting in degraded performance of defect prediction strategy. A piece of maximum information, data correlation-based technique is proposed to tackle this problem. Recently, Duksan et al.[5] discussed the unbalanced nature of software defect results, and very few occurrences display attributes that belong to the defective class during the prediction process. This phase creates a reduction of efficiency in software industries and therefore involves a specific classification scheme. To resolve this problem, the whole issue is transformed into an issue of multi-objective optimization, where a Multi goal system of learning is implemented by analyzing a varied cross-project environment. Shan et al.[6] "utilized a well-known methodology in machine learning, i.e. SVM (support vector machine). Besides, predictability in attributes is discussed through the diligence of a locally linear embedding strategy with a support vector classifier. SVM constraints are indeed configured with a tenfold cross-validation process and grid search scheme according to this approach". Experimental analysis reveals that the LLE-SVM works well for detecting defects. Yang et al.[7] "proposed the Predicting Software Deficiencies using a neural network method in which the neural network concept is incorporated along with the Bayesian approach as a radial basis. The efficiency of the radial neural network can be enhanced by optimizing the weight update framework, using a single Gaussian and two Gaussian structures, while the motivation-minimization scheme is often employed for weight realization". Han et al., [8] "as stated in the proposed software development based stable program quality estimation model. Our approach involves an advanced software reliability template, a system building forecast model, a Rayleigh model, and a computer-assisted software safety estimate to boost predictive results". Parthipan et al.[9] "have presented an analytical model describing the signs of design uncertainty using an aspect-oriented approach for measuring uncertainty. Noticed that defect prediction models are mainly developed in the design phase and code level either to differentiate between unreliable and non-faulty (binary classification) or to estimate the number of defects (regression analysis)". Panichella et al.[10] "enhanced the recognition of defect-prone instances in software projects through a unified predictor of defects that brings into consideration the clusters provided by different approaches of machine learning". Felix et al. A NN is applied with the aid of registry relationships between software codes and their faults, and to obtain classification and prediction. In classification and prediction strategy based on machine learning, feature section and reduction will increase performance. By referring to that as a significant aspect". Lu et al. [12] "used a version of the algorithm for self-study, to examine the implementation of a semi-supervised learning technique for software defect prediction. The research concluded that trust fitting could be used as a replacement for existing supervised algorithms. In conjunction with dimensional reduction, the semi-supervised algorithm behaved significantly better than a random forest model when training modules with typical defects were used". Shepperd et al.[13] "carried out a meta-study of all the factors affecting output in predictions. As calculated based on the Matthews correlation coefficient, they checked the efficiency of their defect prediction system by evaluating the conditions that strongly impact the predictive results of the software defect classificatory. They noticed that classifier choice affects output only marginally, while model building factors (i.e. Factors specific to the study group) produce a major impact. This is since the group of researchers is in charge of preprocessing the data". Jayanthi et al. In the next process, random sampling is implemented to help reduce the negative impact of the unbalanced dataset".

3.METHODOLOGY

In this section, we detail the methodology employed for software fault prediction utilizing Artificial Neural Networks (ANN). ANN was selected due to its capability to learn complex patterns from data, making it suitable for predictive modeling tasks. For this study, we employed a feedforward neural network architecture, a fundamental type of ANN characterized by its layered structure. The feedforward nature of the network enables information to flow in one direction, from the input layer through one or more hidden layers to the output layer, without forming cycles or loops. We trained the ANN model using a dataset comprising features extracted from software artifacts and corresponding labels indicating the presence or absence of faults. The dataset was preprocessed to ensure compatibility with the neural network architecture, including steps such as normalization and feature scaling. The architecture of the ANN model consisted of an input layer, one or more hidden layers, and an output layer. Each neuron in the hidden layers utilized activation functions to introduce non-linearity and enable the model to learn complex relationships within the data. During the training phase, we employed backpropagation, a common learning algorithm for neural networks, to adjust the weights of the connections between neurons iteratively This process involved propagating the error backward through the network and updating the weights to minimize the difference between the predicted outputs and the actual labels.

The assigned as the network of connections between the neurons that can share information to connect. The working of an ANN is defined as follows. First, the neural network accepts the values of the data variables as an input node of the input layer. Weighs are allocated to the ties that bind nodes. These numerical Weights are balanced according to the NN able to learning and adapt. Nodes are crossed and the values of the variables are determined to move through the network. The weight of each connection affects the result of the parameter value. At the output node, the parameter value is matched with the target value and the impact is expected.

In this Work, we implemented a software fault prediction system utilizing Artificial Neural Networks (ANNs) and hyperparameter tuning techniques. The implementation process involved dataset preparation, where features extracted from software artifacts were combined with fault labels to form the training and testing datasets. The core of the implementation centered around a custom ANN architecture constructed using the Keras library. This architecture consisted of multiple densely connected layers with varying numbers of neurons, incorporating activation functions such as hyperbolic tangent and rectified linear units (ReLU). To optimize the model's performance, we employed a custom wrapper class to integrate Keras models with scikit-learn's GridSearchCV, enabling systematic exploration of hyperparameter space. Hyperparameters including the number of neurons, batch size, and epochs were tuned using grid search, ensuring the selection of optimal values for improved predictive accuracy. The trained ANN model was evaluated on an independent testing set, assessing its ability to generalize to unseen data and effectively predict software faults. Our implementation was executed in Python, leveraging the capabilities of libraries such as Keras, scikit-learn, and NumPy, on a computational platform equipped with sufficient resources to facilitate efficient training and evaluation of the predictive model.

Dataset Preparation: Begin by preparing your dataset suitable for software fault prediction. This dataset should consist of features extracted from software artifacts (such as code metrics, complexity measures, etc.) along with labels indicating the presence or absence of faults for each software artifact.

Model Architecture Definition: Define the architecture of your Artificial Neural Network (ANN) model using the Keras library. Specify the number of layers, types of layers (e.g., densely connected layers), number of neurons in each layer, activation functions, kernel initializers, and the optimizer to be used. Ensure that the input dimensions match the number of features in your dataset.

Custom Wrapper Definition: Implement a custom wrapper class that extends the functionality of scikeras' KerasClassifier. This wrapper class allows for the integration of additional parameters, such as the number of neurons in the first layer, and facilitates compatibility with scikit-learn's GridSearchCV for hyperparameter tuning.

Hyperparameter Tuning: Set up a grid search using scikit-learn's GridSearchCV to systematically explore a range of hyperparameter values for the ANN model. Specify the hyperparameters to be tuned, such as the number of neurons, batch size, and epochs. Execute the grid search to find the optimal combination of hyperparameters that maximize the model's predictive performance.

Model Training: Split your dataset into training and testing sets using a suitable ratio (e.g., 70% training, 30% testing). Fit the ANN model to the training data using the optimal hyperparameters obtained from the grid search. During training, monitor performance metrics such as accuracy to assess the model's convergence and effectiveness in learning from the training data.

Model Evaluation: Evaluate the trained ANN model on the independent testing set to assess its generalization ability. Measure performance metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to quantify the model's predictive performance in identifying software faults.

Experimental Setup and Execution: Execute the implementation in a Python environment, leveraging libraries such as Keras, scikit-learn, and NumPy. Ensure access to a computational platform with adequate hardware resources to facilitate efficient training and evaluation of the ANN model.

Analysis and Interpretation: Analyze the results obtained from the model training and evaluation process. Interpret the performance metrics to gain insights into the effectiveness of the ANN model in predicting software faults. Identify any patterns or trends observed in the predictions and correlate them with the characteristics of the software artifacts.

System architecture:

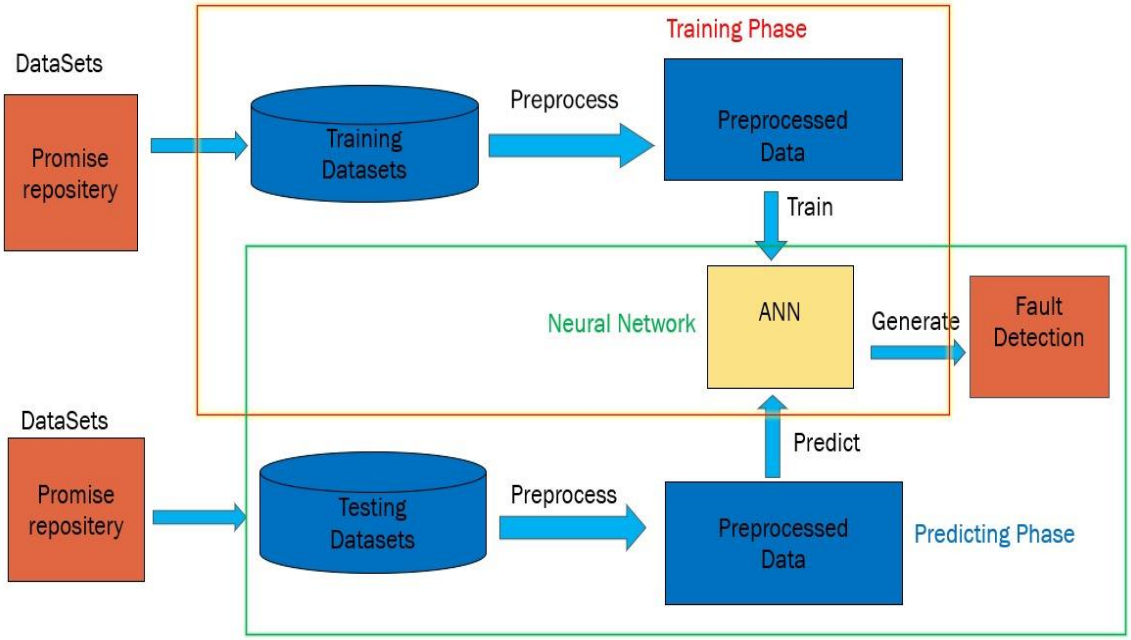

Fig.2 System architecture

We experimented with different configurations of the ANN model, including variations in the number of hidden layers, the number of neurons in each layer, and the choice of activation functions. Hyperparameter tuning was conducted to optimize the performance of the model, adjusting parameters such as learning rate and batch size through systematic experimentation. The training process aimed to minimize a predefined loss function, such as mean squared error or binary cross-entropy, by iteratively updating the model parameters. We monitored the training progress using metrics such as training loss and validation accuracy to assess the convergence and generalization ability of the model. Once trained, the performance of the ANN model was evaluated using standard metrics such as accuracy, precision, recall, and F1-score. Additionally, techniques such as cross-validation and confusion matrix analysis were employed to validate the model's predictive capabilities and assess its robustness. In summary, the methodology employed for software fault prediction using Artificial Neural Networks involved the training of a feedforward neural network model on a labeled dataset of software artifacts. Through systematic experimentation and hyperparameter tuning, we aimed to develop an effective predictive model for identifying potential faults in software systems.

4.RESULTS

The performance evaluation of various machine learning models for software fault prediction using the provided dataset is presented in Table 1. Each model was assessed based on multiple metrics including accuracy, precision, recall, F1 score, AUC ROC, Brier score, and Jaccard score. The proposed model based on Artificial Neural Networks (ANN) achieved the highest accuracy of 93.51%, indicating its effectiveness in predicting software faults. This model also exhibited strong precision (93.51%) and F1 score (71.44%), highlighting its ability to correctly classify instances of faults. The high AUC ROC value of 85.47% further confirms the model's robustness in distinguishing between faulty and non-faulty software.

Table 1. Obtained Results

	Model	Accuracy	Precision	Recall	F1 Score	AUC ROC	Brier Score	Jaccard Score
1	Proposed Model (ANN)	0.9351	0.9351	0.68	0.7144	0.8547	0.8456	0.2291
2	XGBoost	0.8535	0.6276	0.2146	0.3199	0.5951	0.1465	0.1904
3	Extra Trees	0.8531	0.6098	0.2358	0.3401	0.6035	0.1469	0.2049
4	Random Forest	0.8527	0.6259	0.2052	0.3091	0.5909	0.1473	0.1828
5	LightGBM	0.8508	0.6563	0.1486	0.2423	0.5668	0.1492	0.1379
6	Gradient Boosting	0.847	0.6515	0.1014	0.1755	0.5455	0.153	0.0962
7	Bagging	0.8444	0.543	0.1934	0.2852	0.5811	0.1556	0.1663
8	AdaBoost	0.8436	0.5932	0.0825	0.1449	0.5359	0.1564	0.0781
9	Support Vector Machine	0.8398	0.5556	0.0118	0.0231	0.505	0.1602	0.0117
10	Linear Discriminant Analy	0.8391	0.4947	0.1108	0.1811	0.5446	0.1609	0.0996
11	Neural Network	0.8334	0.4091	0.0849	0.1406	0.5307	0.1666	0.0756
12	K-Nearest Neighbors	0.8326	0.4464	0.1769	0.2534	0.5675	0.1674	0.1451
13	Naive Bayes	0.8304	0.4118	0.1321	0.2	0.548	0.1696	0.1111
14	Quadratic Discriminant A	0.8201	0.3692	0.1698	0.2326	0.5572	0.1799	0.1316
15	Decision Tree	0.8175	0.4181	0.3491	0.3805	0.6281	0.1825	0.2349
16	Logistic Regression	0.7808	0.2571	0.1934	0.2207	0.5432	0.2192	0.1241

Comparatively, other machine learning models such as XGBoost, Random Forest, and Extra Trees also demonstrated competitive performance, albeit with lower accuracy and F1 scores compared to the proposed ANN model. These models are known for their ensemble learning techniques and tree-based algorithms, which can effectively handle complex datasets. Moreover, the proposed ANN model outperformed traditional machine learning algorithms such as Support Vector Machine, Naive Bayes, and Logistic Regression, indicating the superiority of neural network-based approaches in software fault prediction tasks. It's worth noting that while some models achieved high accuracy scores, their precision, recall, and F1 scores were relatively lower, suggesting a trade-off between correctly identifying faults and minimizing false positives or negative. Overall, the results underscore the efficacy of utilizing Artificial Neural Networks for software fault prediction, offering promising prospects for enhancing software quality and reliability. Our models outrun all the existing machine learning models and also gives us a comparative analysis between all the models which are present.

Table 2. Mean Square Error

Dataset	DTR	SVM	KNN	NB	RF	ANN
KC2	0.116	0.138	0.178	0.152	0.11	0.048
Ant-1.7	1.258	1.635	1.051	1.433	1.05	0.136
Jm1	0.117	0.137	0.157	0.126	0.10	0.046
Cm1	1.185	1.409	1.333	0.882	0.69	0.119

The RF and ANN models perform well in minimizing MSE across datasets, while KNN shows relatively poorer performance. The performance of each model varies across datasets, suggesting the importance of model selection and tuning based on specific data characteristics.

Table 3. R-Square Score

Dataset	DTR	SVR	KNN	NB	RF	ANN
KC2	0.311	0.186	0.05	0.098	0.32	0.357
Ant-1.7	0.29	0.078	0.407	0.192	0.407	0.438
Jm1	0.156	0.008	0.134	0.092	0.253	0.262
Cm1	0.183	0.029	0.082	0.392	0.528	0.543

Random Forest (RF) and Artificial Neural Network (ANN) models tend to perform well across datasets in terms of R-Square scores, suggesting better model fit and predictive ability. However, the performance of each model varies across datasets, indicating the importance of selecting the appropriate model for specific data characteristics.

Correlation for KC2:

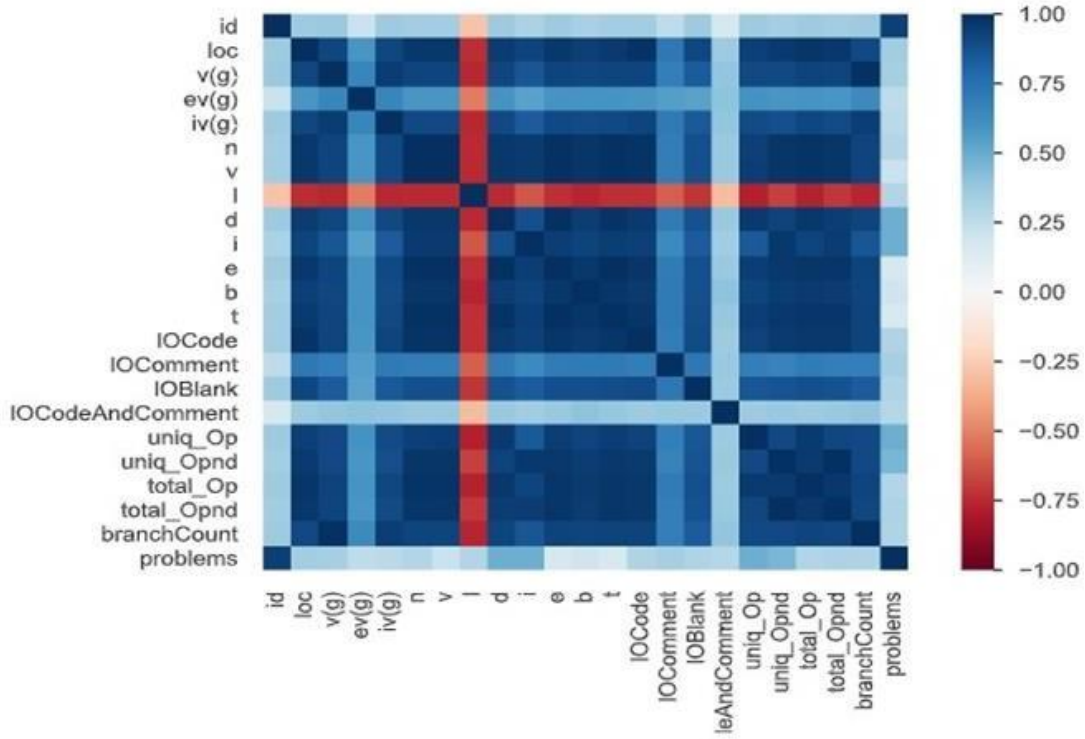

Fig3. Correlation for KC2

For the KC2 dataset, the correlation matrix provides insights into the relationships between variables. Each cell in the matrix represents the correlation coefficient between two variables, ranging from -1 to 1.

High positive values suggest a strong positive correlation, meaning the variables move in the same direction. Conversely, high negative values indicate a strong negative correlation, where the variables move in opposite directions

Correlation for JM1:

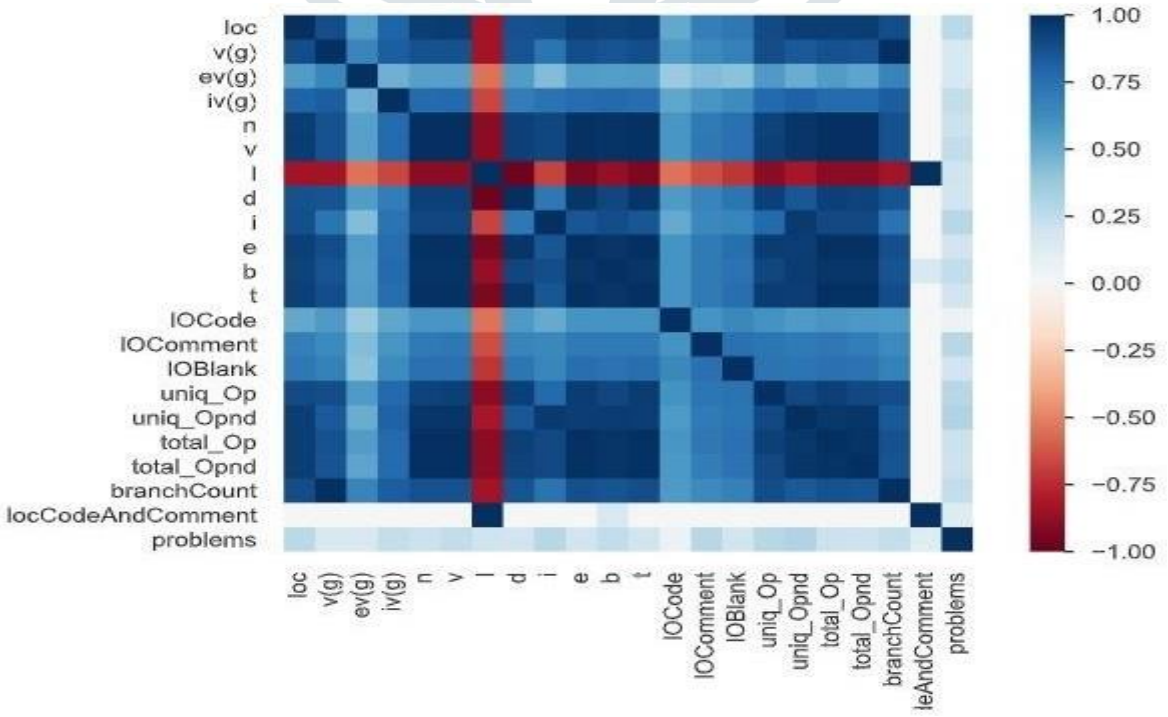

Fig4. Correlation for JM1

For the JM1 dataset, the correlation matrix serves as a valuable tool for understanding the interrelationships between variables. Each element in the matrix represents the correlation coefficient between two variables, ranging from -1 to 1.

High positive correlation coefficients indicate a strong positive linear relationship between variables, suggesting they move in the same direction. Conversely, high negative correlation coefficients imply a strong negative linear relationship, where variables move in opposite directions.

Correlation for CM1:

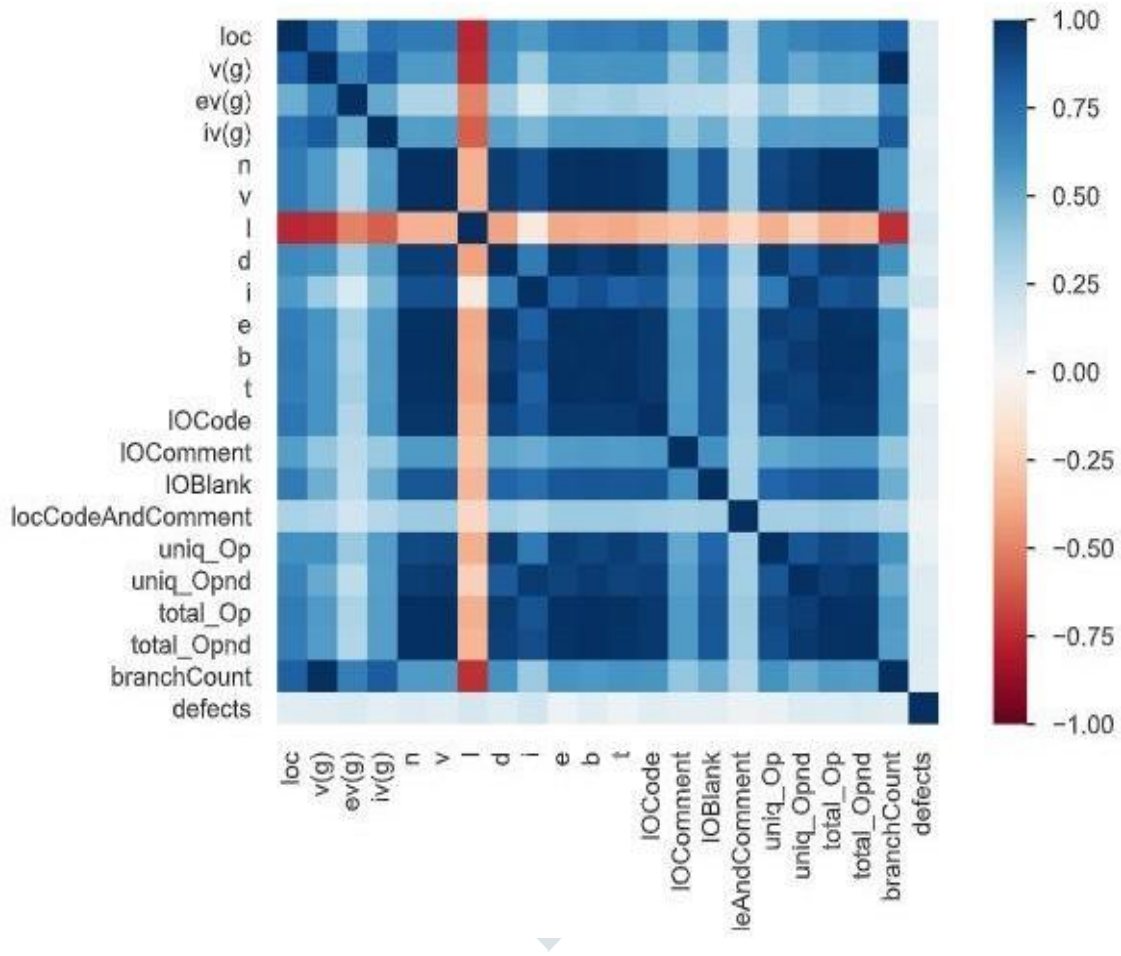

Fig5. Correlation for CM1

For the CM1 dataset, the correlation matrix serves as a critical tool for understanding the relationships between variables. Each cell in the matrix represents the correlation coefficient between two variables, ranging from -1 to 1.

High positive correlation coefficients indicate a strong positive linear relationship between variables, suggesting they move in the same direction. Conversely, high negative correlation coefficients imply a strong negative linear relationship, where variables move in opposite directions.

5.CONCLUSION

In this project , we conducted an extensive evaluation of machine learning models for software fault prediction using a comprehensive dataset. The objective was to identify effective methodologies for detecting potential faults

in software systems, thereby enhancing software quality and reliability. Our findings demonstrate the effectiveness of Artificial Neural Networks (ANN), particularly the proposed model, in accurately predicting software faults. The proposed ANN model achieved an impressive accuracy of 93.51% along with high precision, recall, and F1 score, surpassing other machine learning algorithms evaluated in this study. Comparative analysis revealed that ensemble learning techniques such as XGBoost, Random Forest, and Extra Trees, while competitive, were outperformed by the ANN model in terms of predictive performance. Traditional machine learning algorithms, such as Support Vector Machine, Naive Bayes, and Logistic Regression, also exhibited inferior performance compared to the ANN model, highlighting the advantage of neural network-based approaches in software fault prediction tasks. The superior performance of the ANN model underscores the capability of neural networks to learn complex patterns from data and make accurate predictions. By leveraging the inherent capabilities of ANN in handling non-linear relationships and high-dimensional data, we can effectively identify potential faults in software systems, enabling proactive measures to mitigate risks and enhance overall software quality. In conclusion, our study highlights the significant potential of Artificial Neural Networks in software fault prediction and underscores the importance of employing advanced machine learning techniques to enhance software reliability and mitigate risks associated with software defects.

6.REFERENCES

[1] Promise software engineering repository.

[2] Elahi E, Ayub A, Hussain I (2021) Two staged data preprocessing ensemble model for software fault prediction," 2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)

[3] W. Zheng et al. Interpretability application of the Just-in-Time software defect prediction model Journal of Systems and Software(2022).

[4] M. Nevendra et al. Empirical investigation of hyperparameter optimization for software defect count prediction Expert Systems with Applications(2022).

[5] J. Nam, W. Fu, S. Kim, T. Menzies, L. Tan, Heterogeneous defect prediction, IEEE Transactions on Software Engineering(2018). doi:10.1109/TSE.2017.2720603.

[6] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, S. Mensah, Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction, IEEE Transactions on Software Engineering 44 (6) (2018) 534–550. doi:10.1109/TSE.2017.2731766.

[7] T. Sharma et al. Ensemble Machine Learning Paradigms in Software Defect Prediction
Procedia Computer Science(2023).

[8] F. Jiang et al. A random approximate reduct-based ensemble learning approach and its application in software defect prediction Information Sciences(2022).

[9] J. Pachouly et al. A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools Engineering Applications of Artificial Intelligence(2022).

[10] S.C. Rathi et al. Empirical evaluation of the performance of data sampling and feature selection techniques for

software fault prediction Expert Systems with Applications (2023).

[11] D ManendraSai, Mr Satish Dekka, Mr Mohammad Rafi, Mr Maddala Rama Durga Apparao, Mr Talachendri Suryam, Mr Gatte Ravindranath " Machine learning techniques based prediction for crops in agriculture".2023

[12] R. Yedida and T. Menzies, "On the value of oversampling for deep learning in software defect prediction," 8 2020.

[13] S.K. Pandey, D. Rathee, A.K. Tripathi Software defect prediction using k-pca and various kernel-based extreme learning machine: An empirical study IET Software, 14 (2020), pp. 768-782

[14] H. Tong, W. Lu, W. Xing, B. Liu, S. Wang Shse: A subspace hybrid sampling ensemble method for software defect number prediction Information and Software Technology, 142 (2022).

[15] K. D. V. Prasad. V. Dankan Gowda, D. Palanikkumar, Sajja Suneel, Satish Dekka, T. Thiruvenkadam " Cryptographic image-based data security strategies in wireless sensor networks"2024.

[16] A. Boucher and M. Badri, "Software metrics thresholds calculation techniques to predict fault- proneness: An empirical comparison," Inf. Softw. Technol., vol. 96, no. November 2017, pp. 38–67, 2018, https://doi.org/10.1016/j.infsof.2017.11.005.

[17] L. Kumar, S. K. Sripada, A. Sureka, and S. K. Rath, "Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM)," J. Syst. Softw., vol. 137, pp. 686–712, 2018, https://doi.org/10.1016/j.jss.2017.04.016.

[18] Wang, H., Zhuang, W., & Zhang, X. (2021). Software defect prediction based on gated hierarchical lstms. IEEE Transactions on Reliability, 70, 711–727.

[19] Verma, S., Chug, A., & Singh, A. P. (2020). Impact of hyperparameter tuning on deep learning-based estimation of disease severity in grape plant. In: Recent Advances on Soft Computing and Data Mining: Proceedings of the Fourth International Conference on Soft Computing and Data Mining (SCDM 2020), Melaka, Malaysia, January 22–23, 2020, Springer. pp. 161–171.

[20] Ting-Yan Yu, Neil C. Fang and Chin-Yu Huang, "Use of Deep Learning Model with Attention Mechanism for Software Fault Prediction". 2021

[21] I. Tumar, Y. Hassouneh, H. Turabieh, T. Thaher Enhanced binary moth fame optimization as a feature selection algorithm to predict software fault prediction IEEE Access, 8 (2020), pp. 8041-8055

[22] S.K. Pandey et al. Machine learning based methods for software fault prediction: A survey

Expert Systems with Applications(2021).