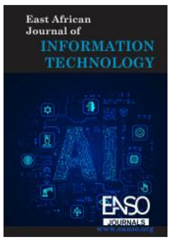


East African Journal of Information Technology
eajit.eanso.org
Volume 6, Issue 1, 2023
Print ISSN: 2707-5346 | Online ISSN: 2707-5354
Title DOI: https://doi.org/10.37284/2707-5354

EANSO
EAST AFRICAN NATURE & SCIENCE ORGANIZATION

Original Article

Development of Pi Sigma Neural Network Model for the Prediction of Software Reliability Using 5 NASA Failure Datasets

Barka Piyinkir Ndahi[12*], *Dr. Opeyemi Aderike Abisoye, PhD*[1], *Dr. Hamzat Olanrewaju Aliyu, PhD*[1] & *Dr. Oluwaseun Adeniyi Ojerinde, PhD*[1]

[1] Federal University of Technology, P. M. B. 65, Minna, Niger State, Nigeria.
[2] University of Maiduguri, P.M.B 1069, Maiduguri, Borno State, Nigeria.
* Correspondance ORCID ID: https://orcid.org/0000-0002-0632-7368; Email: ndahibarka@yahoo.com

Article DOI: https://doi.org/10.37284/eajit.6.1.1366

Date Published:

13 August 2023

Keywords:

Software Reliability, Feedforward Neural Networks, Pi Sigma Neural Networks, Software Fault, Soft Computing, Ensembles.

ABSTRACT

Software reliability models are usually used to model the failure of software systems and prediction of its reliability potential. These models are however plagued with less accuracy, efficiency, and resource-effectiveness. Some soft computing methods have not yet been implemented to investigate their effectiveness and robustness for software fault prediction. Pi Sigma Neural Network (PSNN) software reliability prediction model was developed in this study for a better understanding of the modelling of software systems defects and reliability validated on 5 NASA promise datasets after carrying out data analysis using Seaborn on Python, working with raw data, pre-processed data with min-max normalization, Synthetic Minority Oversampling Technique (SMOTE) to overcome class imbalance problem between defective and non-defective modules, and then correlational analysis with varying thresholds (0.8, 0.85, 0.9 and 0.95) to reduce noise and get key features. The results obtained using the PSNN model showed for all the datasets good average performance for recall being highest at 79.8% based on no threshold, precision at 76.2% on 0.9 threshold, f1-score with 75.6% on 0.95 threshold and accuracy at 74.8% with the same 0.95 threshold. A model based on recall is good at fault finding. Modifying the structure and architecture of the PSNN, like using a voting ensemble algorithm of varied combinations of PSNNs and using a firefly algorithm to optimize in the future, will improve the Neural Network technique.

APA CITATION

Ndahi, B. P., Abisoye, O. A., Aliyu, H. O. & Ojerinde, O. A. (2023). Development of Pi Sigma Neural Network Model for the Prediction of Software Reliability Using 5 NASA Failure Datasets. *East African Journal of Information Technology*, *6*(1), 135-148. https://doi.org/10.37284/eajit.6.1.1366

CHICAGO CITATION

Ndahi, Barka Piyinkir, Opeyemi Aderike Abisoye, Hamzat Olanrewaju Aliyu and Oluwaseun Adeniyi Ojerinde. 2023. “Development of Pi Sigma Neural Network Model for the Prediction of Software Reliability Using 5 NASA Failure Datasets”. *East African Journal of Information Technology* 6 (1), 135-148. https://doi.org/10.37284/eajit.6.1.1366.

HARVARD CITATION

Ndahi, B. P., Abisoye, O. A., Aliyu, H. O. & Ojerinde, O. A. (2023) “Development of Pi Sigma Neural Network Model for the Prediction of Software Reliability Using 5 NASA Failure Datasets”, *East African Journal of Information Technology*, 6(1), pp. 135-148. doi: 10.37284/eajit.6.1.1366.

IEEE CITATION

B. P. Ndahi, O. A. Abisoye, H. O. Aliyu & O. A. Ojerinde, “Development of Pi Sigma Neural Network Model for the Prediction of Software Reliability Using 5 NASA Failure Datasets”, *EAJIT*, vol. 6, no. 1, pp. 135-148, Aug. 2023.

MLA CITATION

Ndahi, Barka Piyinkir, Opeyemi Aderike Abisoye, Hamzat Olanrewaju Aliyu & Oluwaseun Adeniyi Ojerinde. “Development of Pi Sigma Neural Network Model for the Prediction of Software Reliability Using 5 NASA Failure Datasets”. *East African Journal of Education Studies*, Vol. 6, no. 1, Aug. 2023, pp. 135-148, doi:10.37284/eajit.6.1.1366.

INTRODUCTION

Millions of computing devices keep on getting sold on a yearly basis (Alsop, 2021), and the number continues to rise, especially in the mobile industry, with a rate higher than laptop computers (O’Dea, 2021). There are millions of software/applications in stores and other places ready for the population to embrace, with Android users alone being capable of choosing out of 3.48 million apps as of the first quarter of 2021, and more are being developed (Ceci, 2021). Software and computing devices are prone to failure over time, especially as they are released and updated and errors or faults are in them (Bharany et al., 2022).

Software reliability can be defined as the probability of failure-free software operation for a specified period of time in a specified environment (Sahu et al., 2021). Software reliability assessment is a means of mitigating software failure in order to safeguard devices and their contents by making them robust (Sahu et al., 2021). Cases of software failure have been discussed over time, like the problems with the Nigerian Independent National Electoral Commission Election Result Viewing Portal failure, which affected the credibility of Nigerian February 2023 presidential election, various failures in SpaceX rockets exploding when mastering reusable spaceship/rockets, high impact bugs which affect vehicles/tools and ships causing accidents and losses of lives/properties and affecting financial systems, enterprise resource planning system which contributed to $160 million loss for Hewlett-Packard Co. in 2004, and Advanced Automation System cancelled after $2.6 billion is spent by U.S. Federal Aviation Administration in year 2004 (Acheampong, 2023; Minow, 2023; Wu et al., 2021; Charette, 2005). Due to software failure capabilities and effects, finding the reliability of software is a major research problem globally (Khan *et al*., 2021). Software failure poses challenging risks to users of computers; thus, it is imperative to offer a safe platform for users by providing reliable methods against failures. Software reliability is very important because quite a number of systems, such as cars and nuclear plants, nowadays rely on software and for them to be resilient and reliable, it is of great necessity. It is important to minimize failure as much as possible, and this is achievable with the use of soft computing techniques.

Software reliability is a key part of software quality (Sahu et al., 2021). Assumptions and abstractions must be made to simplify the problem, such as by using factors like failure rate or line of code, amongst others (Kather et al., 2021; Prasad & Sangeetha, 2012). Software reliability modelling has matured to the point that meaningful results can be obtained by applying suitable models to the problem (Sahu et al., 2021). Reliability metrics are used to quantitatively express the reliability of the software product. The choice of which metric to adopt depends upon the type of system to which it applies and the requirements of the application domain (Kather et al., 2021; Kaur & Bahl, 2014).

Soft computing is a form of techniques and algorithms that deal with situations where there are uncertainties, partial truth, and ambiguity, and helps in forecasting, optimizing, and decision-making in real-life situations (Pandey et al., 2021; Burney *et al*., 2017). Soft computing is used in software engineering to predict and build models that enhance software reliability (Dhavakumar *et al*., 2018).

Optimization algorithms are used in minimizing loss functions to get desired output. Some popular optimization techniques include gradient descent, particle swarm optimization, firefly algorithm, and genetic algorithm. Combining soft computing models and optimization techniques optimally gives better and more capable models. When constructing models like Artificial Neural Networks, the number of neural nets and how they are linked could impact positively and, at times, even negatively. Using the right number of mini-batches and learning rate contributes to giving models that are efficient and gives output at a rate that is desired with a high level of accuracy. Optimization is usually started by defining the loss/cost function and ends with minimizing it using optimization techniques. The choice of an optimization algorithm can make a difference between getting good prediction accuracy in hours or days. The applications of optimization are limitless and are widely researched topics in industry as well as academia (Diwekar, 2020; Gill *et al*., 2019).

Arasteh (2018) proposed a combined method that includes Neural Network and Naïve Bayes algorithm to build a software fault prediction model. The paper used five traditional fault datasets from NASA via the Promise repository to construct and evaluate the prediction model. The result of the experiments shows higher prediction accuracy and precision than other methods. On average, the accuracy of the constructed prediction models by Naive Bayes, ANN, SVM, and the proposed method is 84.16%, 90.79%, 87.46%, and 96.91%, respectively. The average precision of the constructed prediction models by Naive Bayes, ANN, SVM, and the proposed method are 0.72, 0.74, 0.50, and 0.99. However, the paper recommends a combination of other learning algorithms to build an efficient prediction model.

Kaur and Sharma (2019) employed an ANN-based software fault prediction (SFP) model to classify the fault-prone modules in the work, an ANN-based approach for SFP using object-oriented metrics. The proposed model used the Levenberg Marquardt algorithm for the learning process. ANN was the reputable method for defect module prediction based on the results of the systems proposed. The experiments were performed on 18 public datasets from the PROMISE repository. Receiver operating characteristic curve, accuracy, and mean squared error (MSE) were taken as performance parameters for the prediction task. Results of the proposed systems signified that ANN provides significant results in terms of accuracy and error rate. In this study, values of ROC-AUC varied for different datasets. The average ROC-AUC for this study was approximately 0.92, which showed the usefulness of the proposed ANN prediction model. By the rate of error minimization, the accuracy was increased of the SFP technique. With great accuracy, the technique of ANN proposed can carry out defect module prediction after comparison of the method proposed with existing approaches.

This research, therefore, proposes a modified pi sigma neural network to be combined with the Firefly algorithm to predict software reliability.

Statement of the Problem

Real-time systems tasks require robust and accurate software that will eradicate losses of lives, environmental disasters, and huge financial losses. Such domain includes space exploration life-critical systems like support systems in health care. Soft computing algorithms for software reliability are plagued with less accuracy, efficiency, and resource effectiveness (Son *et al.*, 2019). In addition, there is a need for a combination of learning algorithms to build an efficient and robust software reliability prediction model (Arasteh, 2018). Some of the machine learning techniques have never been implemented for software fault prediction (Pandey *et al.*, 2021). Extensive comparisons between search-based techniques, machine learning techniques, and statistical techniques have not been carried out, and few search-based techniques like Artificial Immune Recognition System (AIRS) and Genetic Programming (GP) have good performance in predicting defects, but a number of studies that support this finding are very small (Son *et al.*, 2019). Additionally, machine learning and statistical techniques, on the other hand, have been evaluated extensively. To reach conclusions, it is important that search-based techniques are also extensively evaluated (Son *et al.*, 2019; Iftikhar *et al.*, 2018).

To address the aforementioned limitations and challenges, this research aims to address the combination of a modified pi sigma neural network with firefly algorithm to predict software reliability. This approach would aim to improve the accuracy, precision, recall, and f1-score (Son *et al*., 2019) of soft computing.

METHODOLOGY

The field of medicine, security, the stock market, pattern recognition, image compression etc., has all seen the application of neural networks in their domain to assist in solving a range of problems, and many have seen great successes (Sharma & Chandra, 2019). This work makes efforts to develop better models in performance for software reliability prediction using neural networks.

For classifying faulty or non-faulty modules, we use neural network classifier to handle such data. However, in multilayered perceptron, slow training is observed due to errors propagated backwardly, while quick learning with little functionality is observed in a neural network that is fed forward due to linear threshold unit (Nayak *et al*., 2020; Nayak *et al*., 2016). Additionally, Pi sigma neural networks (PSNN) minimize this problem and allow for processing information in a faster manner, and they are usually computationally less intensive.

This research is experimental in nature and quantitative in approach. The area of target and sample size is the NASA metrics data program of promise repository found on the internet consisting of datasets with sizes of roughly 500 to 10,000 plus. Data analysis methods include a relplot used for understanding the relationship between variables, bar charts, countplots, a histogram used across variables that are categorical and a boxplot for sketching graphs to show groups of numerical data based on their quartiles; using seaborn in Python.

On such basis, we organize our research in four stages: (1) developing a Pi Sigma Neural Network Software Reliability Prediction Model, (2) developing a voting technique based on the combination of varied pi sigma neural networks, (3) optimizing voting technique based on the combination of the varied pi sigma neural networks using firefly algorithm, (4) evaluating the model.

The final step will give a good solution for software reliability modelling based on the pi sigma neural network. All models are evaluated on data coming from the Promise repository (Shirabad & Menzies, 2005).

Data Collection

To understand the applicability of our model, we have already performed an initial validation of the pi sigma neural network model on CM1, PC1, KC1, KC2, and JM1. Data was collected from the internet; the Promise dataset (Shirabad & Menzies, 2005) was downloaded from the NASA repository. This is a PROMISE Software Engineering Repository data set made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering. The data was used for software defect prediction. The dataset was created by NASA, then the NASA Metrics Data Program in the year 2004.

Table 1: Dataset description

Dataset	Language	Instances	Attributes	Faulty modules (%)
CM1	C	498	21	9.83
PC1	C	1109	21	6.94
KC1	C++	2109	21	15.45
KC2	C++	522	21	20.49
JM1	C	10885	21	19.35

Article DOI: https://doi.org/10.37284/eajit.6.1.1366

The dataset represents defects occurring in the software system. Accuracy, precision, recall, and f1-score were used for measurement.

Data comes from McCabe and Halstead features extractors of source code. These features were defined in the 70s in an attempt to objectively characterize code features that are associated with software quality. The McCabe and Halstead measures are "module"-based where a "module" is the smallest unit of functionality. In C or Smalltalk, "modules" would be called "function" or "method", respectively. CM1 is a NASA spacecraft instrument that characterizes code features that are associated with software quality and are written in "C". PC1 is flight software for an Earth-orbiting satellite that characterizes code features that are associated with software quality and are written in "C". KC1 is a "C++" system implementing storage management for receiving and processing ground data that characterizes code features that are associated with software quality. KC2 has data from C++ functions. Science data processing is another part of the same project as KC1; different personnel than KC1. Shared some third-party software libraries with KC1, but no other software overlap. It characterizes code features that are associated with software quality. JM1 is written in "C" and is a real-time predictive ground system: that uses simulations to generate predictions and characterizes code features that are associated with software quality. *Table 2* below shows the data structure of the datasets.

Table 2: Data structure of the datasets.

Attribute Name	Attribute description
Loc	McCabe's line count of code
v(g)	McCabe's "cyclomatic complexity."
ev(g)	McCabe's "essential complexity."
iv(g)	McCabe's "design complexity."
N	Halstead total operators + operands
V	Halstead "volume"
L	Halstead "program length."
D	Halstead "difficulty"
I	Halstead "intelligence"
E	Halstead "effort"
B	Halstead
T	Halstead's time estimator
lOCode	Halstead's line count
lOComment	Halstead's count of lines of comments
lOBlank	Halstead's count of blank lines
lOCodeAndComment	
uniq_Op	unique operators
uniq_Opnd	unique operands
total_Op	total operators
total_Opnd	total operands
branchCount	of the flow graph
Defects	module has/has not one or more reported defects

Data Analysis

From *Figure 1*, it was found that the data have a common/similar distribution for all the independent features plotted against each other except when an independent feature is plotted against 'l' like in Fig. 2 or against 'locCodeAndComment' like in *Figure 3*. *Figure 2* shows there is an inverse relationship between 'l' and Figure all the other independent variables except when plotted against 'locCodeAndComment' like in *Figure 3*.

Figure 1: Relationship between 'v(g)' and 'loc'

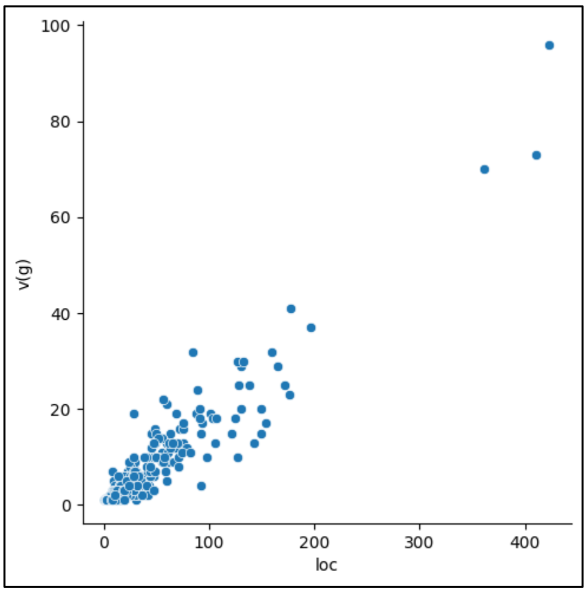


Figure 2: Relationship between 'l' and 'loc'

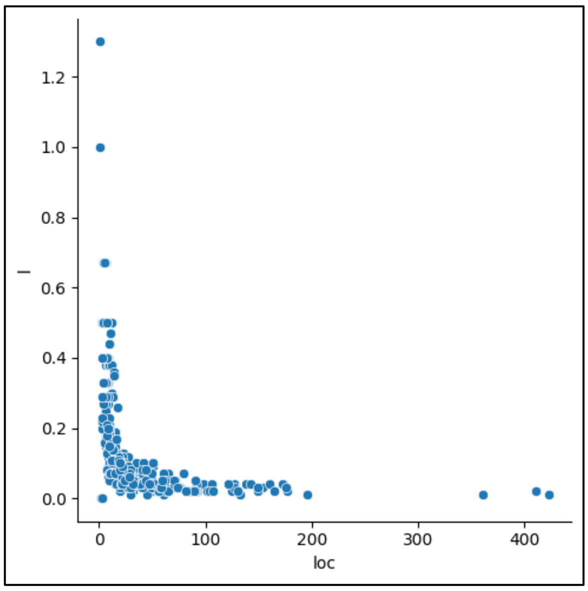


Figure 3: 'l' vs 'locCodeAndComment'

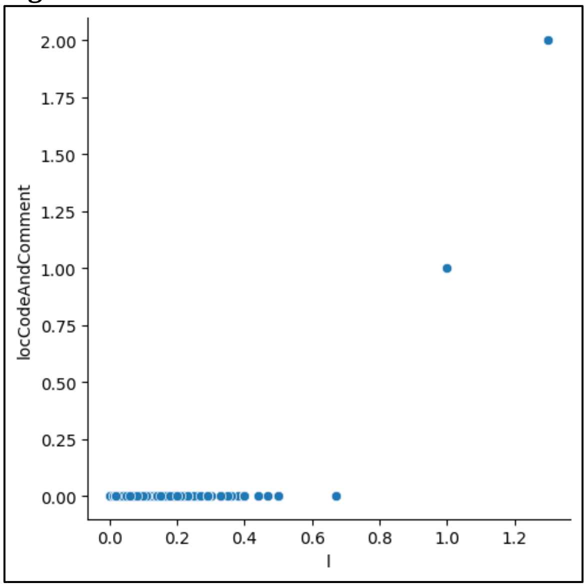


Figure 4: 'loc' vs 'locCodeAndComment'

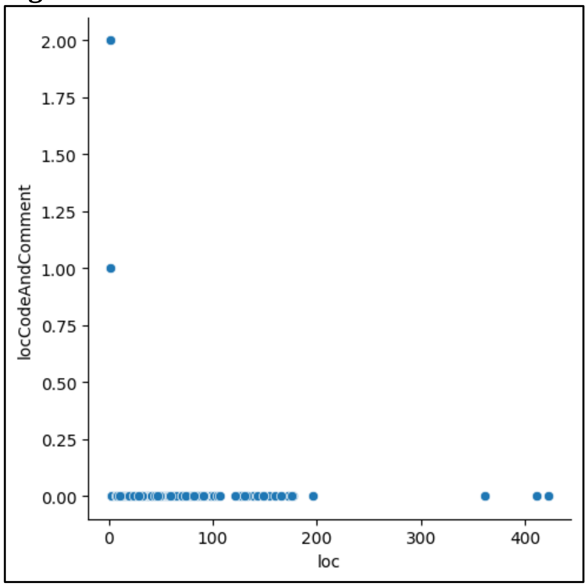


Figure 1 shows that relationships between independent variables and other independent variables, except for the relationship with 'l' and 'locCodeAndComment', have a direct/proportional relationship. It shows that in the relationship of most of the independent features, with the exception of 'l' and 'locCodeAndComment', when one increases, the others also increase. *Figure 4* shows that when 'loc' or other independent variables is close to 0 on the x-axis, 'locCodeAndComment' has most of its values on the line relative to the independent variable or very close to it, with a few exceptions that are scattered.

In a similar manner, using a bar chart, it was revealed that defect modules had a higher count compared to all the other independent features like in *Figure 5*, except when plotted against 'l' like in *Figure 6*. *Figure 5* shows how the defective modules have higher values in all the independent features (except for 'l') compared to non-defective modules. Countplot showed in *Figure 7* there is a class imbalance, with non-defective modules being a lot higher than defective modules.

Figure 5: Bar chart of 'loc' and defects

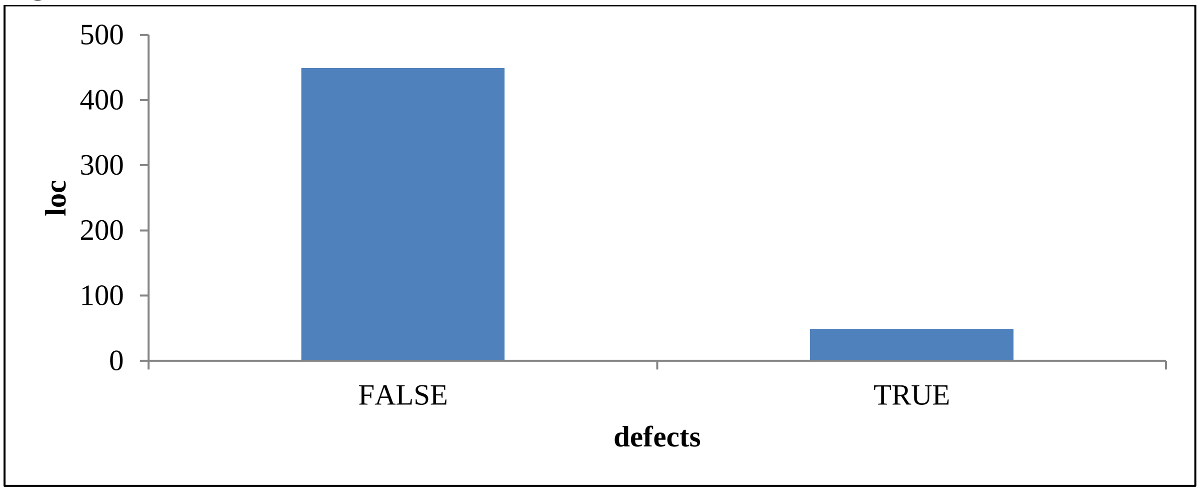


Figure 6: Bar chart of 'l' and defects

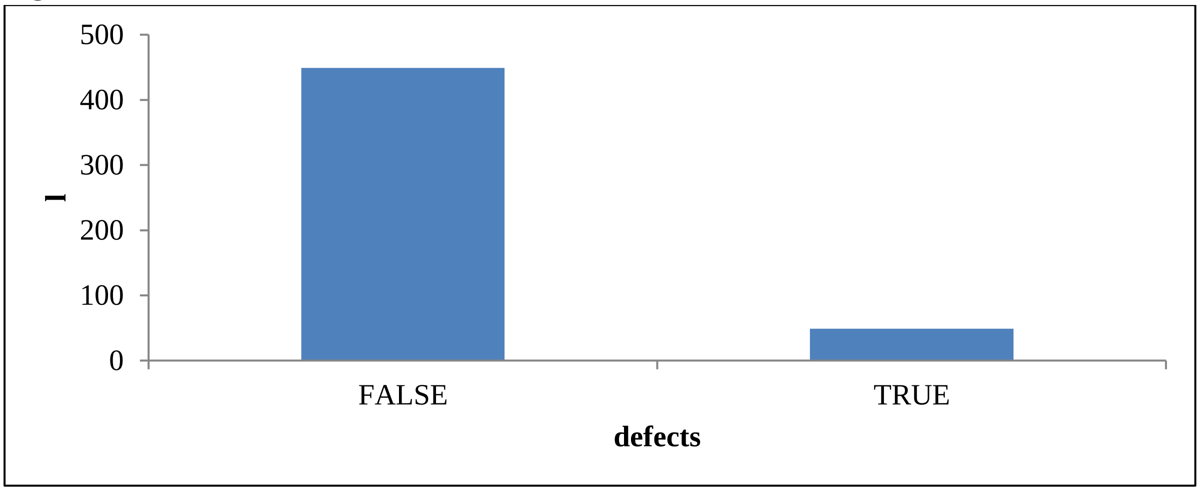
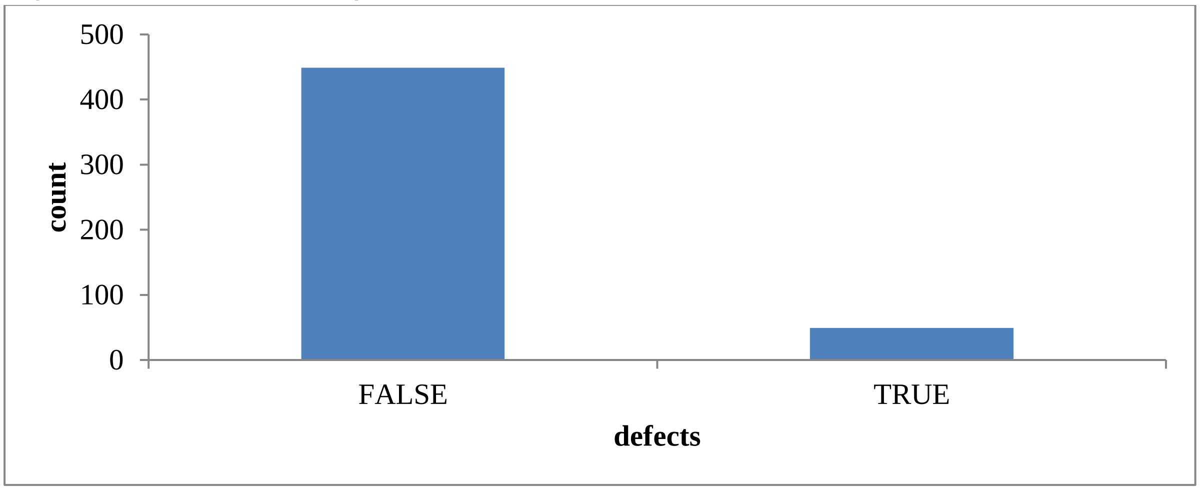


Figure 7: Countplot showing number of non-defective modules compared to defective ones



Boxplot, on the other hand, showed that the median, interquartile range and total distribution of quantitative data within whiskers of defective modules in relationship to other features is higher/highest in all (like in *Figure 8*): this indicates higher values of independent features usually result in faults. Exceptions are of features' l' (see *Figure 9*), which was the opposite because of the inverse relationship, and 'locCodeAndComment' (*Figure 10*), which had equal measures, which shows most of the values are at 0 except outliers. Additionally, the boxplots have wide outliers.

Figure 8: Boxplot of 'loc' against defects

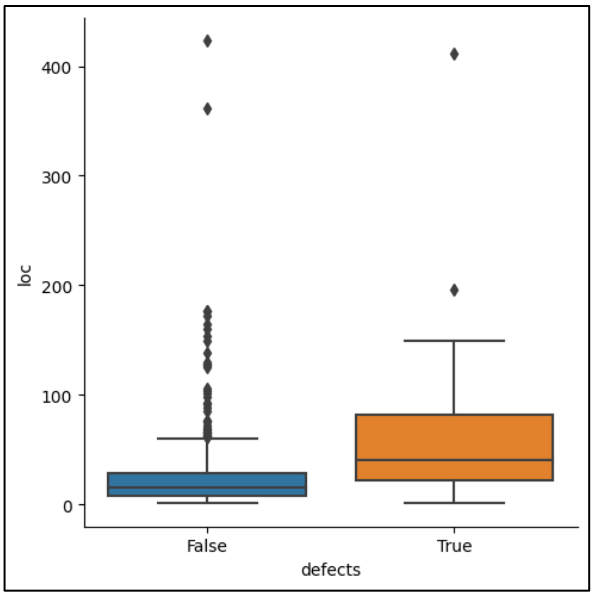


Figure 9: Boxplot of 'l' against defects

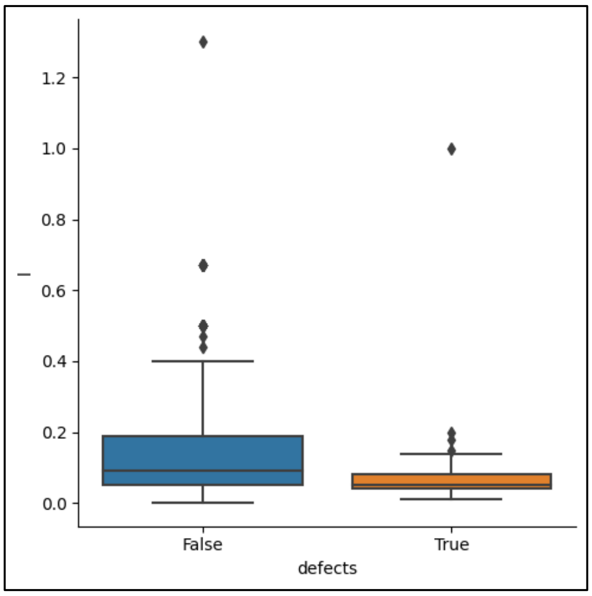
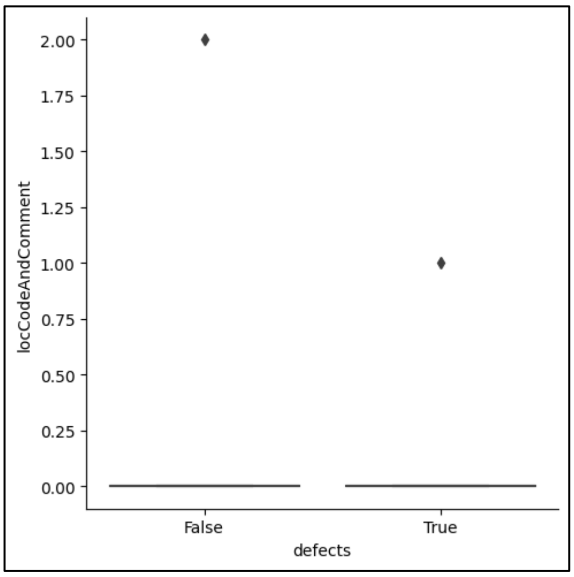


Figure 10: Boxplot of 'locCodeAndComment' against defects



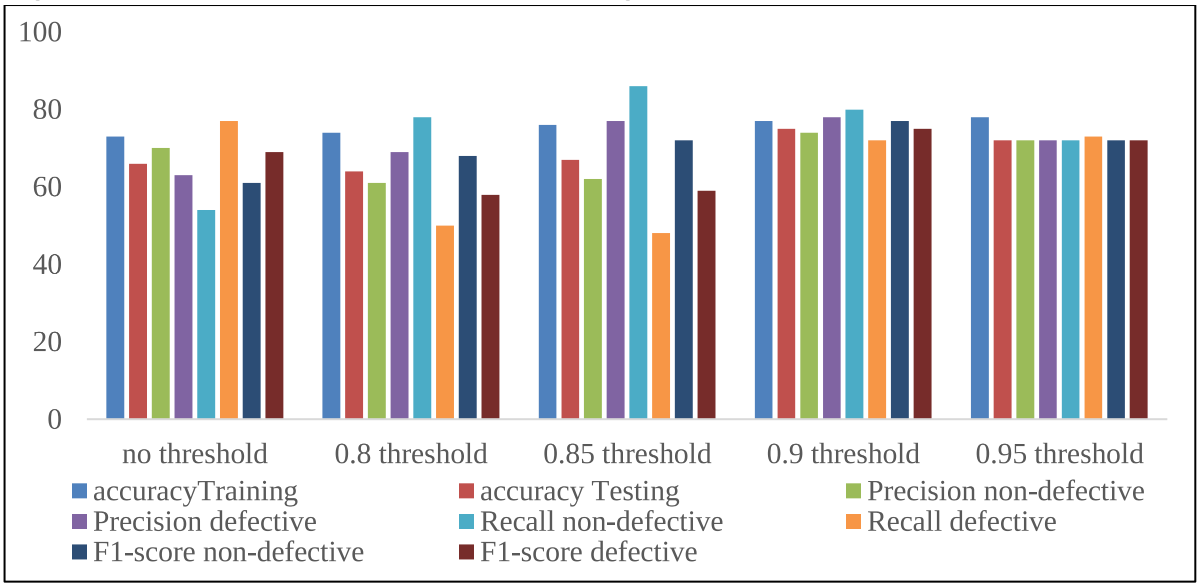
SIMULATION AND RESULTS

Comparison of raw data with pre-processed data was carried out while using CM1 dataset with PSNN using back propagation gradient descent (Shin & Ghosh, 1991; Nayak *et al*., 2016; Nayak *et al*., 2020) for training with epoch of 1001 with linear activation for 3 hidden neurons and sigmoid as activation for the output in this research. Data was split with 70% for training and 30% for testing, while MSE was used for training the model. With raw data, we got roughly 10% accuracy for both training and testing; while with pre-processed data using min-max normalization (Henderi et al., 2021), the accuracy rose to 90% roughly for both the training and testing set, but it was observed that other performance metrics (precision, recall, f1-score) reflected imbalance based on reporting and the report for the defective module was poor.

To solve the imbalance problem, SMOTE oversampling (Prasetiyo et al., 2021) was used to oversample the defective instances to balance up with the non-defective module. Upon simulation, accuracy for the training set was 73%, while for the testing set, 65%. Additionally, there was a great improvement in the other performance metrics.

To further improve accuracy and the other performance metrics, correlational analysis (Son et al., 2019) was performed using thresholds of 0.8, 0.85, 0.9 and 0.95. *Figure 11* shows the results for CM1.

Figure 11: Results of various thresholds indicating performance measure for CM1 (in %)

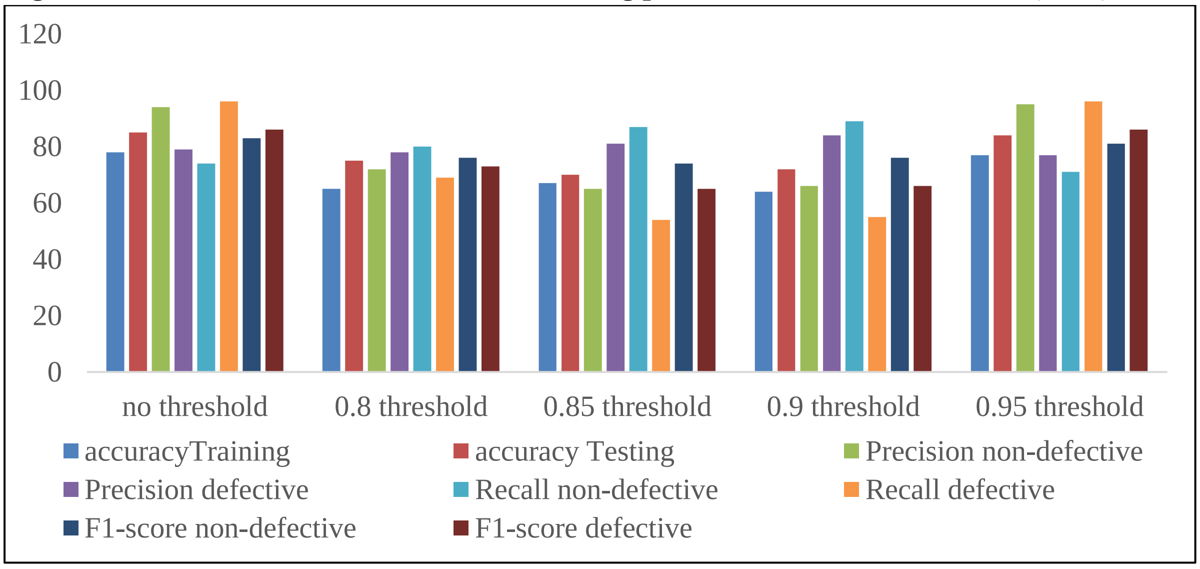


Accuracy at the 0.9 threshold had the highest value at 75%, while precision at the 0.9 threshold had the highest value at 78%. Recall at no threshold had the highest value at 77%, while the F1 score at 0.9 threshold had the highest value at 75%.

Similar data analysis for CM1 was carried out for PC1, KC1, KC2 and JM1, and it was found they shared a common distribution with that of CM1.

In likewise the same manner, similar validation was carried out using PC1, as can be seen *in Figure 12*.

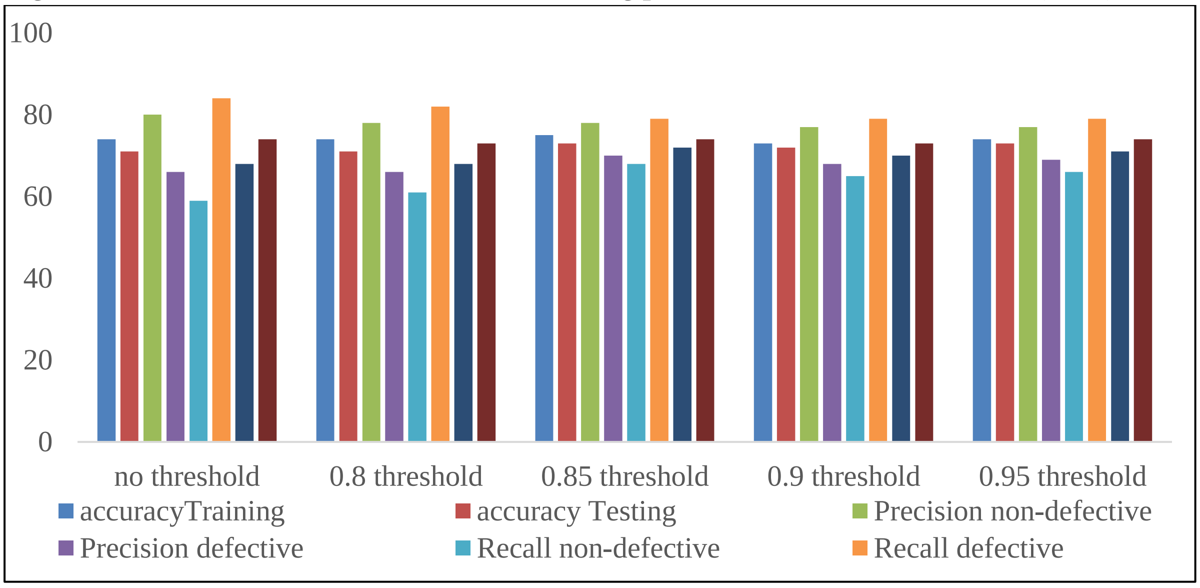
Figure 12: Results of various thresholds indicating performance measure for PC1 (in %)



Accuracy for no threshold has a maximum value of 85%; Precision has the highest score of 84% for the threshold of 0.9. The recall had the highest score of 96% for both no threshold and 0.95. On the other hand, the F1 score at no threshold and 0.95 gave the best value of 86%.

Similar validation was carried out using KC1, as can be seen in *Figure 13*.

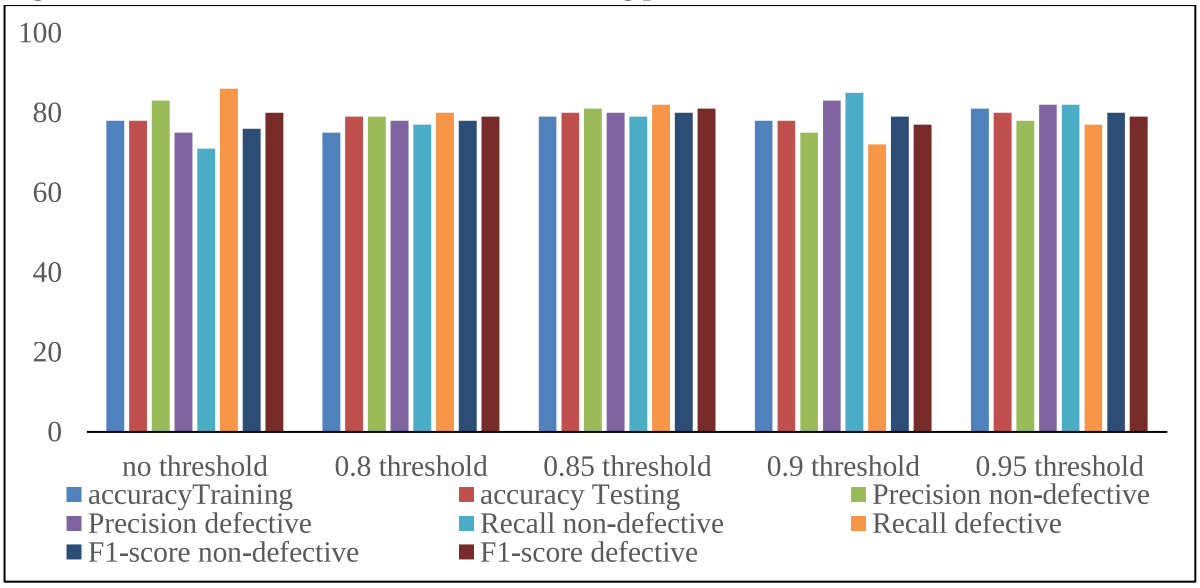
Figure 13: Results of various thresholds indicating performance measure for KC1 (in %)



Accuracy at 0.85 and 0.95 thresholds have the best value of 73%, while precision at 0.85 thresholds has the best value of 70%. Recall, on the other hand, no threshold has the best value of 84%, and lastly, F1 score at no threshold, 0.85 and 0.95 thresholds give the maximum value of 74%.

In likewise the same manner, a similar simulation was carried out using KC2, as can be seen in *Figure 14*.

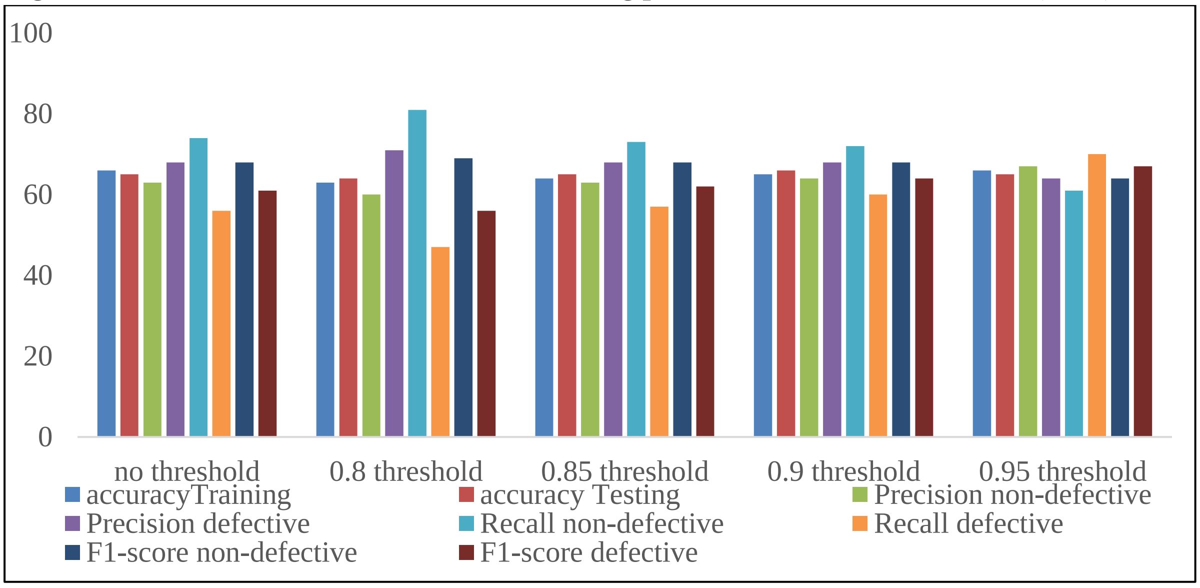
Figure 14: Results of various thresholds indicating performance measure for KC2 (in %)



Accuracy at 0.85 and 0.95 thresholds give the best value of 80%; Precision max value is 83% at the 0.9 threshold. Recall on the other hand at no threshold has the highest value at 86% while F1 score at .85 threshold give best value of 81%.

In the same manner, similar validation was carried out using JM1, as can be seen in *Figure 15*.

Figure 15: Results of various thresholds indicating performance measure for JM1 (in %)



Accuracy at 0.9 threshold gives the best value of 66% while precision is max at 0.8 threshold having 71%. Lastly, recall has 70% for a 0.95 threshold, while F1 score at 0.95 threshold gives the best value of 67%.

Performance Metrics

Recall happens to be the most popular performance measure (Son et al., 2019) for software defect prediction. Recall helps the model to evaluate how many defects or non-defects are predicted correctly out of the total number of defects or non-defects. Precision is also one of the most popular performance measures, followed by accuracy and f1-score (Son et al., 2019). Precision helps a model evaluate how many defects or non-defects are correctly predicted out of a total number of predicted defects or non-defects. Accuracy tells us how correctly a model predicts the overall defective and non-defective models in relationship to all instances (Son et al., 2019). Lastly, F1-score gives the harmonic mean based on precision and recall.

DISCUSSION

Considering recall, the most popular performance metric (Son et al., 2019) in software fault prediction, no threshold performed best for the four datasets (value of 77% for CM1, 84% for KC1, 86% for KC2, 96% for PC1) with one out the four being a tie with .95 threshold (PC1) while for the last dataset JM1, it performed the 4th best out of all the thresholds with a value of 56% unlike the higher values gotten for the other datasets. This could be because JM1 is a huge dataset consisting of roughly 11,000 samples while the rest consist of samples less than 2200. This shows that recall gives us a good and useful model. The average value for the recall is 79.8%.

On the other hand, considering precision, the 0.9 threshold performed the best, having maximum value for three datasets with values of 78% for CM1, 83% for KC2 and, 84% for PC1, 68% for both KC1 and JM1. In likewise the same manner, precision gives us a good and useful model. The average is 76.2%.

Accuracy for the testing set had maximum value for CM1 at 75% and 0.9 threshold, PC1 at 85% with no threshold, KC1 at 73% with 0.85 and 0.95 threshold, KC2 at 80% with 0.85 and 0.95 threshold while JM1 at 66% with 0.9 threshold. Since 0.85 and 0.95 are the most re-occurring, they are going to be further considered. Using 0.85 threshold for CM1, accuracy is 67%, PC1 is 70%, KC1 is 73%, KC2 is 80% and JM1 is 65%. While using threshold of 0.95, CM1 accuracy is

72%, PC1 is 84%, KC1 is 73%, KC2 is 80%, and JM1 is 65%. From analysis, it can be concluded that 0.95 threshold gives the best predictive model based on accuracy. The average is 74.8%.

Lastly, considering f1 score, CM1 at 0.9 threshold gives 75%, PC1 at no threshold and 0.95 threshold gives 86%, KC1 at no threshold, 0.85 and 0.95 threshold gives 74%, KC2 at 0.85 threshold gives 81% while JM1 at 0.95 threshold gives 67%. As the 0.95 threshold is the max appearing in three datasets for PC1 at 86%, KC1 at 74% and JM1 at 67%, based on analysis, this threshold gives the best predictive model based on f1-score. Additionally, for the same threshold, CM1 has a value of 72%, while KC2 is 79%. The average is 75.6%.

But these results are from the simple pi sigma neural network model without any significant modification to its structure/nature, unlike the next stages.

CONCLUSION

This paper showed the development of a software reliability model using a neural network approach, in particular, the pi sigma neural network. Results show that this model has moderate to good performance.

The average values as reported for recall is 79.8%, precision is 76.2%, f1-score is 75.6% and lastly, accuracy is 74.8%. Based on the value, theory, data and desires of the software project, the best predictive model that can show a good measure of faults in the system is recall, followed by the others based on their value and needs of software developers as reported. The developed reliability prediction model for software helps the software testers to focus their effort on the error-prone modules instead of the modules as a whole. With cost minimal, the developers of the software can build projects that are even better in reliability.

Future Work

The first and simple stage has shown moderate to good results in prediction abilities. The second stage is developing a voting technique based on the varied combination of pi-sigma neural networks, which is going to be applied to the software reliability prediction modelling. The third stage is optimizing the voting technique based on the varied combination of pi-sigma neural networks using the Firefly algorithm. The firefly algorithm has shown good results on optimization problems in the literature. The fourth stage is evaluating the model. The combination of these approaches and the developed model, theoretically, will give better results on the above-mentioned dataset and other similar ones from NASA/promise repository.

REFERENCES

Acheampong, M. (2023). Overpromising and Underdelivering? Digital Technology in Nigeria's 2023 Presidential Elections.

Alsop, T. (2021). Global PC unit shipments 2006-2020. https://www.statista.com

Arasteh, B. (2018). Software Fault-Prediction using Combination of Neural Network and Naive Bayes Algorithm. *Journal of Networking Technology*, *9*(3), 94-101. DOI: 10.6025/jnt/2018/9/3/94-101

Bharany, S., Sharma, S., Khalaf, O. I., Abdulsahib, G. M., Al Humaimeedy, A. S., Aldhyani, T. H., ... & Alkahtani, H. (2022). A systematic survey on energy-efficient techniques in sustainable cloud computing. *Sustainability*, *14*(10), 6256.

Burney, S. A., Ali, S. M., and Burney, S. (2017). A survey of soft computing applications for decision making in supply chain management. In *IEEE 3rd International Conference* on *Engineering Technologies and Social Sciences* (pp. 1–6). https://doi.org/10.1109/ICETSS.2017.8324158

Ceci, L. (2021). Number of apps available in leading app stores 2021. https://www.statista.com/

Charette, R. N. (2005). Why software fails. *IEEE spectrum*, 42(9), 36.

Dhavakumar, P., Shankar, S., Vikram, P. M. (2018, April). Soft computing techniques for enhancing software reliability. *International Journal of Latest Trends in Engineering and Technology*, 133-140. https://www.ijltet.org

Diwekar, U. M. (2020). *Introduction to applied optimization* (Vol. 22). Springer Nature.

Gill, P. E., Murray, W., & Wright, M. H. (2019). Practical optimization. Society for Industrial and Applied Mathematics.

Henderi, H., Wahyuningsih, T., & Rahwanto, E. (2021). Comparison of Min-Max normalization and Z-Score Normalization in the K-nearest neighbor (kNN) Algorithm to Test the Accuracy of Types of Breast Cancer. *International Journal of Informatics and Information Systems*, *4*(1), 13-20.

Iftikhar, A., Musa, S., Alam, M., Su'ud, M. M., Ali, S. M. (2018, October). Application of Soft Computing Techniques in Global Software Development: state-of-the-art Review. *International Journal of Engineering & Technology*, *7*(4.15), 304-310. DOI: 10.14419/ijet.v7i4.15.23015

Kather, P., Duran, R., & Vahrenhold, J. (2021). Through (tracking) their eyes: Abstraction and complexity in program comprehension. *ACM Transactions on Computing Education (TOCE)*, *22*(2), 1-33.

Kaur, G., & Bahl, K. (2014, May). Software Reliability, Metrics, Reliability Improvement Using Agile Process. *IJISET - International Journal of Innovative Science, Engineering & Technology*, *1*(3). http://www.ijiset.com

Kaur, R., & Sharma, S. (2018, July). An ANN based approach for software fault prediction using object-oriented metrics. In *International Conference on Advanced Informatics for Computing Research* (pp. 341-354). Springer, Singapore.

Khan, A. W., Hussain, I., & Zamir, M. (2021). Analytic hierarchy process-based prioritization framework for vendor's reliability challenges in global software development. Journal of Software: Evolution and Process, 33(3), e2310.

Minow, J. I. Spacecraft Anomalies and Failures Workshop 2023: NASA Introductory Comments. In *Spacecraft Anomalies and Failures 2023 Workshop*.

Nayak, J., Naik, B., & Behera, H. S. (2016). A novel nature inspired firefly algorithm with higher order neural network: performance analysis. *Engineering Science and Technology, an International Journal*, *19*(1), 197-211.

Nayak, J., Naik, B., Pelusi, D., & Krishna, A. V. (2020). A comprehensive review and performance analysis of firefly algorithm for artificial neural networks. *Nature-Inspired Computation in Data Mining and Machine Learning*, 137-159.

O'Dea, S. (2021). Global smartphone sales to end users 2007-2021. Retrieved from https://www.statista.com

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2021). Machine learning based methods for software fault prediction: A survey. *Expert Systems with Applications*, *172*, 114595.

Prasad, R. S., & Sangeetha, Y. (2012). SPC for Software Reliability using Inflection S-Shaped Model. *International Journal of Computer Applications*, *60*(2).

Prasetiyo, B., Muslim, M. A., & Baroroh, N. (2021, June). Evaluation performance recall and F2 score of credit card fraud detection unbalanced dataset using SMOTE oversampling technique. In *Journal of Physics: Conference Series* (Vol. 1918, No. 4, p. 042002). IOP Publishing.

Sahu, K., Alzahrani, F. A., Srivastava, R. K., & Kumar, R. (2021). Evaluating the impact of prediction techniques: Software reliability perspective. *Computers, Materials & Continua*, *67*(2), 1471-1488.

Shirabad, J. S., & Menzies, T. J. (2005). The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. http://promise.site.uottawa.ca/SERepository

Sharma, D., & Chandra, P. (2019). A comparative analysis of soft computing techniques in software fault prediction model development. *International Journal of Information Technology*, *11*(1), 37-46.

Shin, Y., & Ghosh, J. (1991, July). The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *IJCNN-91-Seattle international joint conference on neural networks* (Vol. 1, pp. 13-18). IEEE.

Son, L. H., Pritam, N., Khari, M., Kumar, R., Phuong, P. T. M., & Thong, P. H. (2019). Empirical study of software defect prediction: a systematic mapping. *Symmetry*, *11*(2), 212.

Wu, X., Zheng, W., Chen, X., Zhao, Y., Yu, T., & Mu, D. (2021). Improving high-impact bug report prediction with combination of interactive machine learning and active learning. *Information and Software Technology*, *133*, 106530.