**SOFTWARE DEFECT PREDICTION: AN EMPRICAL STUDY OF**

**DEEP LEARNING TECHNIQUES.**

Milan Bins Mathew1, Surya Manjunathan G2, Gokul B3, Mohana Ganesh G4

1 Vellore Institute of Technology, Chennai 600127, India.

2 Vellore Institute of Technology, Chennai 600127, India.

3 Vellore Institute of Technology, Chennai 600127, India.

4 Vellore Institute of Technology, Chennai 600127, India.

**ABSTRACT**

Software Defect Prediction (SDP) is a very critical task and highly important in the Software Development Cycle. In today’s world every software is composed of hundreds of thousands of lines of code. In some environments such as self-driving cars we are looking at millions of lines of code to ensure safety and reliability of the machines. Every software is split into several modules for easier and more efficient programming. Identifying and correcting these errors in the initial stages of development is highly essential as later this process can be very expensive and may potentially cause hazards. It is very hard for humans to find these defects in software and though SDP makes this easier, it is not always easy to predict defective models. Today we have several methods, mostly Machine Learning (ML) techniques combined with other Statistical techniques that are used in SDP environments to predict defects in software. This study aims to utilize state of the art neural network models to use in SDP with the help of 4 datasets. The techniques used here include the Artificial Neural Network (ANN), Recurrent Neural Network (RNN), Convolutional Neural Network (CNN) and a hybrid model of ANN with Support Vector Machine. The performance of each of these models has been evaluated using different metrics such as RMSE (Root Mean Square Error), AUC ROC (Area Under ROC Curve), F1 Score and Accuracy.

**Keywords:** Deep Learning, Software defect, Neural Network Models, datasets, performance metrics.

1. **Introduction**

Software Engineering is the systematic application of engineering approaches to software. The various cycles in the software development cycle include designing, developing, maintaining, testing and performance evaluation. The quality of any piece of software is determined upon by factors such as reliability, flexibility and being error-free. A major step towards this goal is to avoid any defects in the software that can result in any kind of failure or inconsistencies to the system. Therefore, in the domain of Software Engineering, SDP is of utmost significance and must go hand-in-hand with the entire software development process. A software defect is defined as a flaw in the code that causes a deviation from the desired/expected output. This deviation may or may not cause serious damages but always impacts the Quality Assurance (QA) of the software thereby, reducing the reliability and credibility on its developers. Hence, every company understands the importance of proper SDP systems.

Software Defect Prediction as such was not a major step in earlier days when the computing systems were simpler and task-specific. It was conducted during the testing stages of the Development Cycle. But with the modern innovations and computers being used to drive cars, pilot airplanes, perform surgeries the systems as well as the software that runs on it has become extremely complex and have become a grim challenge for software engineers to manually test and predict the defects in the software. It is inefficient and expensive for manual testing and SDP in the primary period of the software development cycle is measured as an important aspect of Software Quality Assurance.

Thus, forecasting the software defects in any software has become very prominent in the recent decades as it helps in reducing the overall expenses while constantly maintaining the quality of the software. It is one of the most assistive operations in the software industry. Machine Learning (ML) techniques were a major driving force in this prediction of software defects and was considered an industry standard. But SDP has always been hard as there is no definite heuristic that can be used to identify defects in software. Now with the growing efficiency of deep learning models the ML techniques are now slowly being replaced. A major reason being that Deep Learning Techniques can identify deeper features from the dataset and perform better analysis than ML techniques.

This study focuses on the empirical analysis of four Deep Learning (DL) techniques used in SDP of which some are potentially new solutions as well. The DL techniques used here include thee Artificial Neural Network (ANN), Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), and a hybrid solution using ANN for feature extraction and Support Vector Machine (SVM) for classification. These techniques are employed on four different datasets JM1, KC2, AR1, MC1.

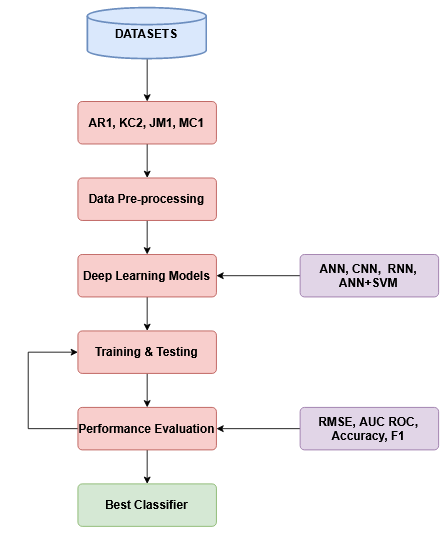
1. **Literature Survey:**

Scientists have been trying to find an accurate model to predict software defects for a long time now. Khan et.al. [1] in their study have performed the performance analysis of Machine Learning (ML) techniques for SDP on various datasets such as the AR1, AR3, CM1, JM1, KC2, KC3, and MC1. Different ML algorithms used here are support vector machine (SVM), Decision tree, Random Forest (RF), KNN, Multilayer perceptron (MLP), Naive Bayes, Credal Decision tree, Radial Basis function, and Average one dependency estimator. The experimental results have shown that the Random Forest algorithm performed better with better accuracy, recall, RMSE values. Iqbal [2] performed comparative exploration of several ML techniques for SDP on twelve NASA datasets such as MW1, CM1, JM1, PC1, PC2, PC3, PC4, PC5, KC1, KC3, MC1, and MC2, while the classification techniques include one rule (OneR), NB, MLP, DT, RBF, kStar (K∗), SVM, KNN, PART, and RF. Bashir et.al. tried to improve the SDP model using Ranker feature selection (RFS), data sampling (DS), and iterative partition filter (IPF) techniques to conquest class imbalance, noisy correspondingly, and high dimensionality. Seven ML techniques including NB, RF, KNN, MLP, SVM, J48, and decision stump are employed on CM1, JM1, KC2, MC1, PC1, and PC5 datasets for evaluations. Overall experimental outcomes of the proposed model outperformed other models. Czibula et.al. proposed a novel classification model based on relational association rule mining. This method outperformed most of the common ML techniques used at that time. Kamei et.al [5] made a detailed report regarding the accomplishments in the software defect prediction field and the future challenges. The paper covers all the changes in the industry over the past two decades. Bayesian networks have also been used to predict software defects and were considered effective along with data mining techniques [6, 7]. Li et.al. [8] proposed a technique using convolutional neural networks (CNN) for this purpose. CNNs are commonly used for speech recognition and image classification [9, 10]. This proves that CNN models are effective in identifying local patterns. Jayanthi et.al. [11] proposed a prediction technique using neural networks. These were some of the first fronts on solving the SDP problem. Xu et al. [12] studied software defect prediction techniques and concluded that conventional techniques use preprocessing and feature selection schemes for reducing irrelevant features but this process still ignores some important features as well. The rest of the paper discusses our novel approach to solving this problem and is organized as follows. In Section 3, we present the details of the methodology and different techniques used by us. Section 4 contains our experimental results and discussions. In Section 5, we provide our conclusion and vision for the future.

1. **Methodology and Techniques**

The aim of this study is to present the performance analysis of deep learning models for SDP on datasets such as JM1, AR1, KC2, and MC1. The datasets have been made available to public use by the University of California, Irvine. The datasets contain numerical data regarding the software under consideration. We have used the following performance metrics for evaluating the model: Root Mean Square Error (RMSE), Area Under the Curve of Receiver Characteristic Operation (AUC ROC), and Accuracy and Mean Absolute Error (MAE).

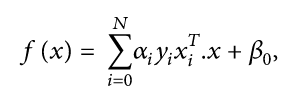
Fig 1. represents the experimental setup for the study.



*Algorithm (Fig.1)*

**3.1 Support Vector Machine (SVM)**

Support Vector Machine or SVM is a very popular and commonly used ML technique that has numerous uses in the field of classification, photonics, pattern recognition etc. SVM solves problems by mapping the training examples to points in space so as to maximize the width of the gap between any two different categories. Though initially developed for binary classification, SVM can now also efficiently perform multiclass-classification. Equation (1) represents the function used in linear SVM.

 (1)

**3.2 Artificial Neural network (ANN)**

Artificial Neural Network or ANN is a deep learning approach inspired by the biological neural networks in animal brains. The learning is done through updating of weights associated with neurons and edges. The weight increases or decreases the strength of the signal at a connection. Here, we have used the ANN for two different models, one with a simple linear classifier and one with the machine learning classifier SVM and is used to identify if the software is defective or not.

**3.3 Convolutional Neural Networks (CNN)**

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural networks, most commonly applied to analyze visual imagery. Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons are mathematical functions that calculate the weighted sum of multiple inputs and gives an activation value as output. When input is an image in a ConvNet, each layer generates several activation functions that are passed onto the next layer. Though the dataset is a set of numerical values, the data is reshaped into a 3-d array to feed it as an input to the model of CNN.

**3.4 Recurrent Neural Networks (RNN):**

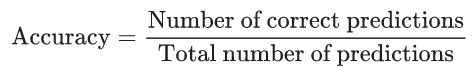
Recurrent Neural Networks is the special class of neural networks that stores the previous input and uses that information along with other information it has to predict the output for the current input. It is a very powerful neural network model for large datasets and for applications requiring high computational power as it is the only neural network with internal memory. In this project we had used 2 Long Short-Term Memory (LSTM) layers as the feedback RNN layers to our model.LSTM is an extension for recurrent neural networks, which basically extends the memory. Therefore, it is well suited to learn from important experiences that have very longtime lags in between. LSTM has 3 important gates: input, forget and output which determines which information to be stored which to be discarded and which is the best information to pass on to the next layer. In addition, we have used batch normalization, added drop out layers, applied kernel and bias regularizes to optimize the overall performance of the model.

1. **Experimental Results:**

The experiment has been compiled and run on a laptop with GPU and has been trained for 200 epochs. The datasets have been split into 80% for training and 20% for validation. The results have been evaluated and compiled using different performance metrics and has been given below.

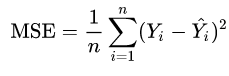
**4.1 Accuracy**

Accuracy is a commonly used performance metric that represents the fraction of the predictions that have been correctly classified by a model. Thus, accuracy can be mathematically represented as given in equation (2).

 (2)

**4.2 Mean Absolute Error**

Mean Squared Error or MSE of an estimator measures the average of the squares of errors. That is the average squared difference between the actual values and predicted values. It can be mathematically represented as in equation (3)

 (3)

**4.3 Area Under Receiving Operating Characteristics Curve**

Area Under Receiving Operating Characteristics Curve, or AUC ROC, is one of the most important evaluation metrics for evaluating performance of classification models. It tells us how well the model can distinguish between different classes and the value ranges from 0 to 1.

**4.4 Root Mean Squared Error**

Root Mean Square Error (RMSE) or Root Mean Squared Deviation (RMSD) is the standard deviation of the residuals (difference of actual value and predicted value). It is a measure of how far the deviation is from expected values and can be represented using equation (4).

 (4)

**ANN model**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Datasets** | **RMSE** | **AUC ROC** | **Accuracy** | **MAE** |
| JM1 | 0.17 | 0.53 | 82 | 0.182 |
| KC2 | 0.18 | 0.60 | 81 | 0.142 |
| AR1 | 0.04 | 0.50 | 96 | 0.003 |
| MC1 | 0.003 | 0.699 | 99 | 0.0001 |

*Performance of ANN Model (Table 1).*

**CNN model**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Datasets** | **RMSE** | **AUC ROC** | **Accuracy** | **MAE** |
| JM1 | 0.436 | 0.514 | 80.98 | 0.189 |
| KC2 | 0.425 | 0.5 | 81.90 | 0.180 |
| AR1 | 0.282 | 0.5 | 92.00 | 0.08 |
| MC1 | 0.0859 | 0.5 | 99.26 | 0.007 |

*Performance of CNN Model (Table 2).*

**ANN + SVM model**

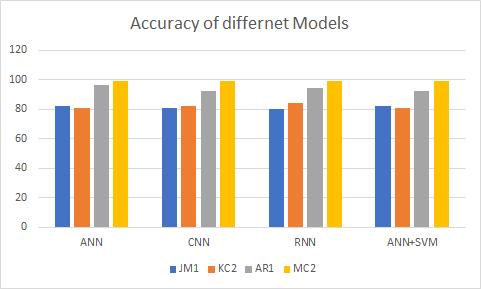
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Datasets** | **RMSE** | **AUC ROC** | **Accuracy** | **MAE** |
| JM1 | 0.42 | 0.57 | 82 | 0.184 |
| KC2 | 0.43 | 0.61 | 81 | 0.2 |
| AR1 | 0.28 | 0.5 | 92 | 0.08 |
| MC1 | 0.08 | 0.5 | 99 | 0.007 |

*Performance of Hybrid Model (Table 3).*

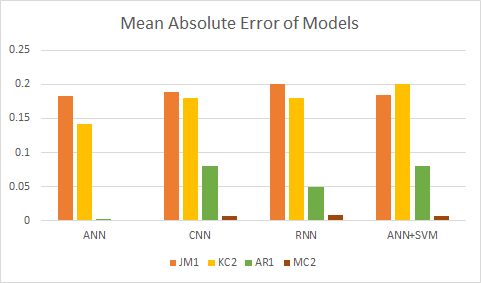
**RNN model**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Datasets** | **RMSE** | **AUC ROC** | **Accuracy** | **MAE** |
| JM1 | 0.44 | 0.55 | 80 | 0.201 |
| KC2 | 0.40 | 0.66 | 84 | 0.18 |
| AR1 | 0.22 | 0.5 | 94 | 0.05 |
| MC1 | 0.09 | 0.53 | 99 | 0.009 |

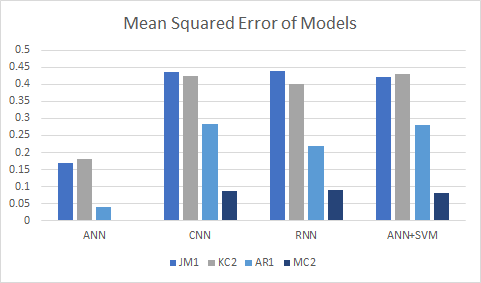
*Performance of RNN Model (Table 4).*

****

*Accuracy Plot (Fig.2)*

**

*Mean Absolute Error of Different Models (Fig.3)*

**

*Mean Squared Error of Different Models (Fig.4).*

The performance of different models can be seen in above. The models have been evaluated using performance metrics such as Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Accuracy and Area Under the Curve of Receiver Characteristic Operation (AUC ROC). The comparative results of the different models with respect to their accuracy, mean absolute error and mean squared error can be seen in Figures 2, 3 and 4 respectively. From the generated plots we can see that the RNN and CNN models are able to identify the features more effectively and are able to consistently and almost same accuracy values. The ANN and the Hybrid model show a slightly dropped accuracy in the KC2 dataset with the former having the highest accuracy in the AR1 dataset. The ANN model on the other hand shows no error in the MC2 dataset which indicates an overfitting of the model. Therefore, from the above plots we can conclude that the Recurrent Neural Network Model (RNN) shows the highest consistency in its results and lower residuals when compared to other models.

1. **Conclusions:**

The identification of software defects at the primary phase of SDLC is a challenging task, as well it can subsidize the provision of high-quality software systems. So in this paper we compared the performance of widely used deep learning models like ANN, RNN, CNN, ANN+SVM based on the performance metrics like accuracy, AUC ROC score, MAE, RMSE values. The goal of this project is to find the best deep learning model that can predict software defects efficiently. Based on the results generated and the following evaluation we can infer that the RNN model performs best as it gives consistently better results and lower residuals than other models. It also does not show the problem of overfitting as it was seen in the ANN models. Though it performs as good as the CNN model it has lower error rates than CNN. This is an important characteristic as Accuracy is not an ideal metric in the field of SDP as this has given rise to a lot of false positives during prediction that is why an equal if not greater importance has been given to the error values as well.

For future work an important task is to better fit the model to the different datasets. As we have arrived at the conclusion that RNN is a better choice, the next step will be to identify other modern architectures and techniques which use the Recurrent Network as its base, such as bi-directional LSTM, and GRU models and other novel approaches to identify the ideal technique to perform Software Defect Prediction.

1. **Data Availability:**

The datasets used in this research are taken from UCI ML Learning Repository available at <https://archive.ics.uci.edu/>.

1. **Compliance with Ethical Standards**

**Conflict of interest** the authors declare no conflict of interest.

**Ethical approval** This article does not contain any studies with human

participants performed by any of the authors. This article does not

contain any studies with animals performed by any of the authors. This

article does not contain any studies with human participants or animals

performed by any of the authors.

**Informed consent** There is no individual participant included in the

1. **References**
2. Bilal Khan, Rashid Naseem, Muhammad Arif Shah, Karzan Wakil, Atif Khan, M. Irfan Uddin, Marwan Mahmoud, *"Software Defect Prediction for Healthcare Big Data: An Empirical Evaluation of Machine Learning Techniques",* Journal of Healthcare Engineering, vol. 2021.
3. A. Iqbal et al., “Performance analysis of machine learning techniques on software defect prediction using NASA datasets,” *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, pp. 300–308, 2019.
4. K. Bashir, T. Li, C. W. Yohannese, and Y. Mahama, *“Enhancing software defect prediction using supervised-learning based framework,”* in Proceedings of the 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Nanjing, China, November 2017.
5. Czibula, Gabriela; Marian, Zsuzsanna; Czibula, Istvan Gergely (2014). *Software defect prediction using relational association rule mining.* Information Sciences, 264(), 260–278.
6. Kamei, Yasutaka; Shihab, Emad (2016). [IEEE 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) - Suita, Osaka, Japan (2016.3.14-2016.3.18)] 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) - Defect Prediction: Accomplishments and Future Challenges. , (), 33–45.
7. Okutan, Ahmet; Yıldız, Olcay Taner (2014). *Software defect prediction using Bayesian networks*. Empirical Software Engineering, 19(1), 154–181.
8. Yuan Chen, ; Xiang-heng Shen, ; Peng Du, ; Bing Ge, (2010). *[IEEE 2nd International Conference on Computer and Automation Engineering (ICCAE 2010) - Singapore (2010.02.26-2010.02.28)] 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE) - Research on software defect prediction based on data mining. , (), 563–567.*
9. Li, Jian; He, Pinjia; Zhu, Jieming; Lyu, Michael R. (2017). *[IEEE 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) - Prague, Czech Republic (2017.7.25-2017.7.29)] 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) - Software Defect Prediction via Convolutional Neural Network. , (), 318–328.*
10. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn*, “Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition,” in ICASSP’12: Proc. of the International Conference on Acoustics, Speech and Signal Processing, 2012.*
11. Krizhevsky, I. Sutskever, and G. E. Hinton, *“Imagenet classification with deep convolutional neural networks,” in NIPS’12: Proc. of the Advances in Neural Information Processing Systems, 2012.*
12. Jayanthi, R.; Florence, Lilly (2018)*. Software defect prediction techniques using metrics based on neural network classifier. Cluster Computing,*
13. Xu, Z., Xuan, J., Liu, J., Cui, X.: MICHAC: *defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, pp. 370–381 (2016)*