Study of Software Defect Prediction using Forward Pass RNN with Hyperbolic Tangent Function

Swati Rai[1], Dr. Kirti Jain[2]

[1]Research Scholar, [2]Associate Professor,
[1, 2]SSOAC, SAGE University, Bhopal, Madhya Pradesh, India

ABSTRACT
For the IT sector and software specialists, software failure prediction and proneness have long been seen as crucial issues. Conventional methods need prior knowledge of errors or malfunctioning modules in order to identify software flaws inside an application. By using machine learning approaches, automated software fault recovery models allow the program to substantially forecast and recover from software problems. This feature helps the program operate more efficiently and lowers errors, time, and expense. Using machine learning methods, a software fault prediction development model was presented, which might allow the program to continue working on its intended mission. Additionally, we assessed the model's performance using a variety of optimization assessment benchmarks, including accuracy, f1-measure, precision, recall, and specificity. Convolutional neural networks and its hyperbolic tangent functions are the basis of the deep learning prediction model FPRNN-HTF (Forward Pass RNN with Hyperbolic Tangent Function) technique. The assessment procedure demonstrated the high accuracy rate and effective application of CNN algorithms. Moreover, a comparative measure is used to evaluate the suggested prediction model against other methodologies. The gathered data demonstrated the superior performance of the FPRNN-HTF technique.

KEYWORDS: *FPRNN-HTF (Forward Pass RNN with Hyperbolic Tangent Function), precision, recall, specificity, F1-measure, and accuracy*

How to cite this paper: Swati Rai | Dr. Kirti Jain "Study of Software Defect Prediction using Forward Pass RNN with Hyperbolic Tangent Function" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-7 | Issue-6, December 2023, pp.268-273, URL: www.ijtsrd.com/papers/ijtsrd60159.pdf

IJTSRD60159

1. INTRODUCTION

The presence of flaws in software significantly impacts its dependability, quality, and upkeep expenses. Even with diligent application, it might be difficult to get bug-free software since most defects are buried. A significant issue in software engineering is also creating a software bug prediction model that might identify problematic modules early on. Predicting software bugs is a crucial task in software development. This is so that user happiness and overall program performance may be increased by anticipating the problematic modules before software is deployed. Additionally, early software problem prediction enhances software adaptability to various situations and maximizes resource efficiency.

Numerous research have been conducted on the prediction of software bugs with machine learning methods. Take the linear Auto-Regression (AR) technique, for instance, to forecast the defective modules. Based on previous data on software accumulated flaws, the research forecasts future software errors. The research also used the Root Mean Square Error (RMSE) method to assess and compare the AR model with the Known Power Model (POWM). Three datasets were also included in the research for assessment, and the outcomes looked good. The research examined the suitability of several machine learning techniques for defect prediction. The key earlier studies on each machine learning approach and the most recent developments in machine learning-based software bug prediction.

2. BACKGROUND

Robotic programming deformity expectation (SDP) tactics are gradually used, sometimes with the use of artificial intelligence (AI) processes, according to Görkem Giray et al. [1]. But existing machine learning methods need physically removed highlights, which are cumbersome, time-consuming, and only partially capture the semantic information disclosed in bug describing equipment. Professionals have the invaluable opportunity to extract and benefit from more complex and multi-layered knowledge as a result of profound learning (DL) techniques.

According to Iqra Batool et al. [2], programming engineers may identify problematic builds-like modules or classes-early in the product advancement life cycle with the use of programming issue/deformity expectation. Information mining, artificial intelligence, and deep learning techniques are used to program expectations that are not satisfied.

Heterogeneous deformity expectation (HDP), as introduced by Haowen Chen et al. [3], refers to the imperfection prediction amongst projects with different measurements. The majority of current HDP methods severely limit their interpretability by mapping source and target data into a conventional measurement space where each element has no real significance. Furthermore, HDP often faces the problem of class inequality.

According to Cagatay Catal et al. [4], phishing attacks aim to steal personal information by using sophisticated techniques, tools, and tactics. Some examples of these are happy infusion, social engineering, online forums, and mobile apps. A number of phishing location techniques were developed in order to prevent and lessen the risks of these attacks; deep learning calculations proved to be one of the most effective.

Xieling and others [5], A number of open-source and endeavor-supported information diagrams have emerged in recent years, marking a remarkable advancement in the application of information portrayal and thinking into a variety of domains, including computer vision and natural language processing. The goal of this research is to thoroughly examine the current state and trends of information diagrams, with a focus on the topical examination structure.

Cagatay et al. (2006) Innovative techniques are put forward for identifying and eliminating the various types of malware, with deep learning computations playing a crucial role. Even while the development of DL-based portable malware detection techniques has received a great deal of attention, it hasn't been thoroughly examined yet. The objective of this effort is to identify, compile, and review the published publications related to the use of deep learning techniques to the detection of portable malware.

One of the major challenges in programming advancement and programming language research for further enhancing programming quality and dependability is ality expectation (Akimova et al., Deform et al., 2007). The problem in this area is to accurately and very precisely identify the corrupted source code. Developing a prediction model with shortcomings is a challenging problem for which several approaches have been put out throughout history.

3. PROBLEM IDENTIFICATION

It is typical development practice to verify and examine source codes using analytical techniques. This procedure may be carried either automatically or manually with the use of tools for dynamic and static code analysis, among other things. Static code analysis has seen a recent surge in tool development, offering really useful, added-value solutions to many of the issues that software development companies encounter. However, these techniques are difficult to utilize in real-world scenarios due to a large number of false positive and false negative outcomes. Therefore, another technique or approach-such as Machine Learning (ML) algorithms-must be discovered for static code analysis.

The problems that have been identified based on previous studies are listed below:

- It is not always possible to identify relevant software flaws.
- A software bug's recovery is not entirely recognized.
- Because of its poor precision, the unnamed software problem may be detected.

4. RESEARCH OBJECTIVES

The objectives of the proposed work:

- To increase accuracy for flawless software bug retrieval.
- To increase recall for software faults that are absolutely applicable throughout the retrieval process.
- To increase the precision of software bug detection.

5. METHODOLOGY

The Algorithm of proposed methodology FPRNN-HTF (Forward Pass RNN with Hyperbolic Tangent Function) is as follows

I = Number of input layers

H = Number of hidden layers

O = Number of output layers

S = Number of data set instances

Step 1: for i = 1 to H

Step 2: for j = 1 to S

calculating the forward for the forward hidden layers with activation function

$$h_t^f = \tanh\left(W_h^f h_{t-1}^f + W_x^f x_t + b_h^f\right)$$

end for

Step 3: for j=S to 1

calculating the backward pass for the backward hidden layer's activation function

$$h_t^b = \tanh\left(W_h^b h_{t-1}^b + W_x^b x_t + b_h^b\right)$$

end for

end for

Step 4: for i =1 to O

calculating the forward pass for the output layer using the previous stored activation function

$$P\left(y_t \middle| \{x_i\}_{i \neq t}\right) = \sigma\left(W_y^f h_t^f + W_y^b h_t^b + b_y\right)$$

W_y is the weight matrix connecting the hidden layer to output layer,

W_h is the weight matrix that connects hidden to hidden layer,

and W_x is the weight matrix that connects input layer to hidden layer.

b_y is the output layer bias vectors, and b_h is the hidden layer bias vectors.

For the final nonlinearity r, and use tanh as an activation function for classification. According to this form, the RNN will evaluate the output y_t according to the information propagated through the hidden layer regardless of whether it depends directly or indirectly on the values $\{x_i\}_{i=1}^t = \{x_1, x_2, ..., x_t\}$.

end for

end for

The Architecture of proposed methodology FPRNN-HTF (Forward Pass RNN with Hyperbolic Tangent Function) is as follows

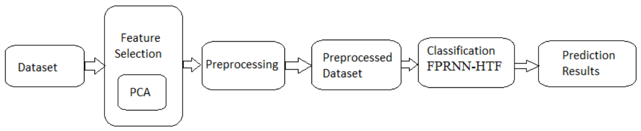
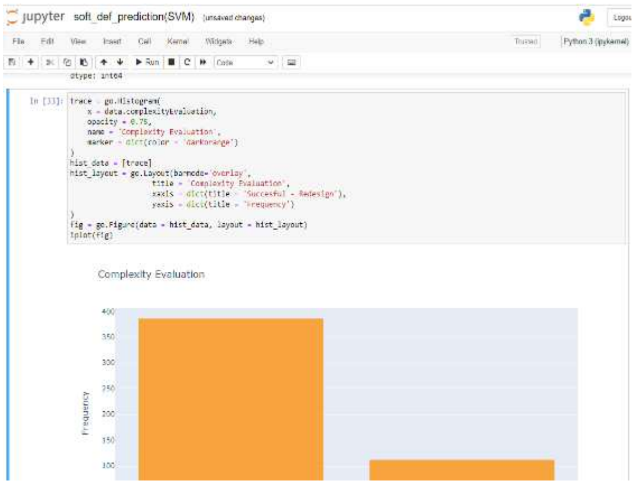


Figure 1: Process of proposed work

6. RESULTS AND ANALYSIS

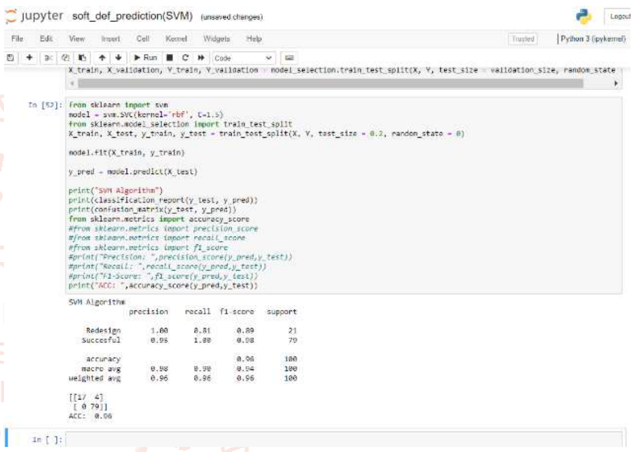
Python 3.11.1 on Anaconda Navigator and a Jupyter notebook are used to take the following metrics. The computation of precision, recall, F1-Score, and accuracy is determined by using the recommended FPRNN-HTF method on CS1.csv data from the PROMISE dataset.


jupyter soft_def_prediction(SVM) (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Complexity Evaluation
Frequency
400
350
300
250
200
150
100

```
dtype: int64

In [33]: trace = go.Histogram(
             x = data.complexityEvaluation,
             opacity = 0.75,
             name = 'Complexity Evaluation',
             marker = dict(color = 'darkorange')
         )
         hist_data = [trace]
         hist_layout = go.Layout(barmode='overlay',
                             title = 'Complexity Evaluation',
                             xaxis = dict(title = 'Succesful - Redesign'),
                             yaxis = dict(title = 'Frequency')
         )
         fig = go.Figure(data = hist_data, layout = hist_layout)
         iplot(fig)
```

Figure 2: Complexity Evaluation of Bug Frequency for FPRNN-HTF (Proposed Prediction Model)


jupyter soft_def_prediction(SVM) (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size = validation_size, random_state

In [52]: from sklearn import svm
         model = svm.SVC(kernel='rbf', C=1.5)
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

         model.fit(X_train, y_train)

         y_pred = model.predict(X_test)

         print("SVM Algorithm")
         print(classification_report(y_test, y_pred))
         print(confusion_matrix(y_test, y_pred))
         from sklearn.metrics import accuracy_score
         #from sklearn.metrics import precision_score
         #from sklearn.metrics import recall_score
         #from sklearn.metrics import f1_score
         #print("Precision: ",precision_score(y_pred,y_test))
         #print("Recall: ",recall_score(y_pred,y_test))
         #print("F1-Score: ",f1_score(y_pred,y_test))
         print("ACC: ",accuracy_score(y_pred,y_test))

         SVM Algorithm
                       precision    recall  f1-score   support

             Redesign       1.00      0.81      0.89        21
            Succesful       0.95      1.00      0.98        79

             accuracy                           0.96       100
            macro avg       0.98      0.90      0.94       100
         weighted avg       0.96      0.96      0.96       100

         [[17  4]
          [ 0 79]]
         ACC:  0.96

In [ ]:
```

Figure 3: Calculation of confusion matrix, precision, recall, F1-Score and accuracy among different models and FPRNN-HTF (Proposed Prediction Model)

Table 1: Estimation of Precision, Recall, F1-Score and Accuracy among different models and FPRNN-HTF (Proposed Prediction Model)

Models	Precision	Recall	F1-Score	Accuracy
Random Forest	0.9	0.83	0.9	80.65 %
Naïve Bayes	0.94	0.82	0.88	80.10 %
Logistic Regression	0.93	0.8	0.88	79.5 %
Decision Tree	0.83	0.84	0.83	74.86 %
ANN	0.92	0.84	0.9	82.77 %
FPRNN-HTF (Proposed)	**0.95**	**0.96**	**0.98**	**96 %**

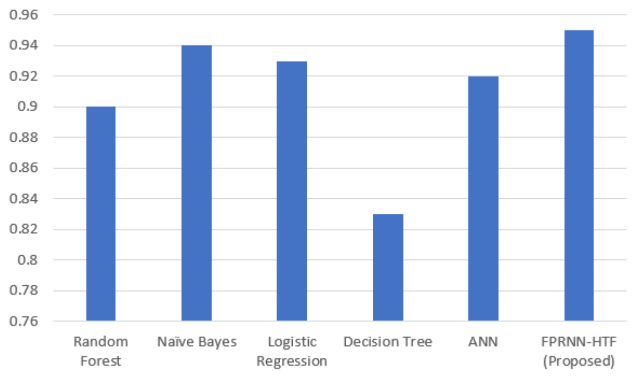


Figure 4: Graphical Analysis of Precision among different models and FPRNN-HTF (Proposed Prediction Model)

When compared to alternative models in the context of bug prediction, the following graphic shows that the recommended model offers higher accuracy. In terms of accuracy, FPRNN-HTF beats Naive Bayes by a margin of 0.01.

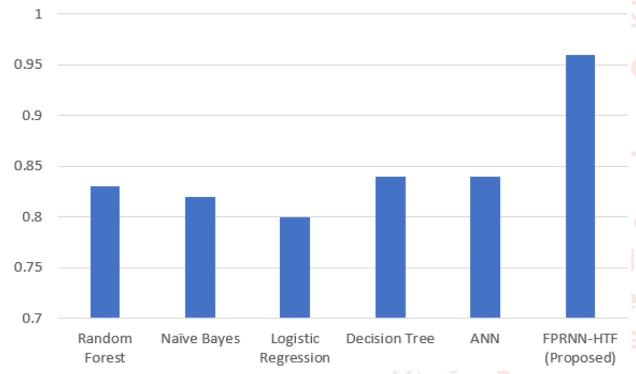


Figure 5: Graphical Analysis of Recall among different models and FPRNN-HTF (Proposed Prediction Model)

The graph above illustrates how the proposed model outperforms earlier models in terms of recall for bug prediction. FPRNN-HTF has a 0.12 improvement in recall over the Decision Tree and ANN prediction models.

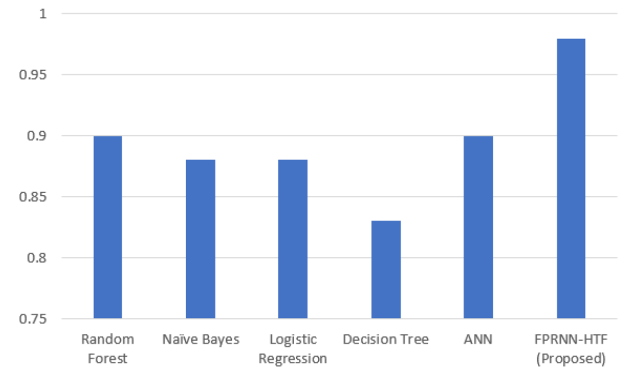


Figure 6: Graphical Analysis of F1-Score among different models and FPRNN-HTF (Proposed Prediction Model)

The graph above illustrates how the proposed model's F1-score is greater than that of earlier models. In terms of F1-score, FPRNN-HTF is superior than Random Forest and ANN by 0.08 points.

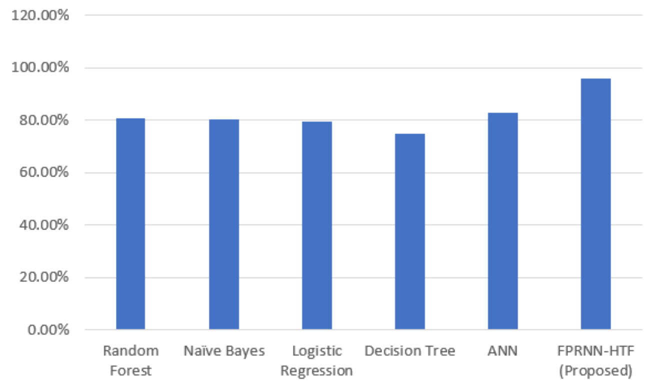


Figure 7: Graphical Analysis of Accuracy among different models and FPRNN-HTF (Proposed Prediction Model)

When compared to current models, the following data shows that the recommended model offers a greater accuracy for predicting bugs. The accuracy of FPRNN-HTF prediction model is 13.23% greater than that of ANN prediction model.

7. CONCLUSION

We have used the FPRNN-HTF model in this investigation to get the intended findings. Our research demonstrates that prior attempts did not pay enough attention to feature selection and cross validation. The recommended technique outperforms other ones in terms of accuracy (98.16%) on large datasets. The combination of the five developed approaches yields the best results since it is computationally demanding (it avoids overfitting and gives fast prediction speeds) and versatile in application (it can be used for both regression and classification problems). Investigating this method further for bug prediction in deep learning models has been a continuous endeavor.

There should be conclusions to these:

1. The accuracy of the proposed model is higher than that of FPRNN-HTF. The accuracy has increased by 0.01 compared to Naïve Bayes.
2. The proposed model achieves higher recall than FPRNN-HTF Regression. FPRNN-HTF has a 0.12 improvement in recall over the Decision Tree and ANN prediction models.
3. In terms of F1-Score, the suggested model performs better than the FPRNN-HTF. The difference between Random Forest and ANN is 0.08.
4. The proposed model has a greater accuracy in comparison to ANN. There is an accuracy increase of 13.23%.

Thus, for software bug prediction, FPRNN-HTF (Forward Pass RNN with Hyperbolic Tangent Function) is a more accurate method.

We propose a technique that enhances diagnostic precision-a critical component of successful treatment. New datasets should be used to evaluate the accuracy in the future, and further AI approaches should be used to confirm the correctness of the estimate. Owing to the enormous amount of data required for train data performance estimate, the suggested model has a processing time limit. The effectiveness of the system will be estimated in the future using real-time data and the same algorithms.

REFERENCES

[1] Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, Bedir Tekinerdogan, "On the use of deep learning in software defect prediction", The Journal of Systems & Software, 2023.

[2] Iqra Batool, Tamim Ahmed Khan, "Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review", Computers and Electrical Engineering, May 2022.

[3] Haowen Chen, Xiao-Yuan Jing, Yuming Zhou, Bing Li, Baowen Xu, "Aligned metric representation based balanced multiset ensemble learning for heterogeneous defect prediction", Information and Software Technology, July 2022.

[4] Cagatay Catal, Görkem Giray, Bedir Tekinerdogan, Sandeep Kumar & Suyash Shukla, "Applications of deep learning for phishing detection: a systematic literature review", Knowledge and Information Systems, 2022.

[5] Xieling Chen, Haoran Xie, Zongxi Li, Gary Cheng, "Topic analysis and development in knowledge graph research: A bibliometric review on three decades", Neurocomputing, October, 2021.

[6] Cagatay Catal, Görkem Giray, Bedir Tekinerdogan, "Applications of deep learning for mobile malware detection: A systematic literature review", 2021.

[7] Konstantin S. Kobylkin, Anton V. Konygin Ilya P. Mezentsev and Vladimir E. Misilov, "A Survey on Software Defect Prediction Using Deep Learning", IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2021.

[8] Farah Atif, Manuel Rodriguez, Luiz J. P. Araújo, Utih Amartiwi, Barakat J. Akinsanya & Manuel Mazzara "A Survey on Data Science Techniques for Predicting Software Defects", IEEE Conf. of Software Engineering, 2021.

[9] Saleema Amershi; Andrew Begel; Christian Bird; Robert DeLine; Harald Gall; Ece Kamar; Nachiappan Nagappan, "Software Engineering for Machine Learning: A Case Study", IEEE Conf. on Machine Learning, 2019.

[10] George G. Cabral; Leandro L. Minku; Emad Shihab; Suhaib Mujahid, "Class Imbalance Evolution and Verification Latency in Just-in-Time Software Defect Prediction", IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019.

[11] Kwabena Ebo Bennin; Jacky Keung; Passakorn Phannachitta; Akito Monden; Solomon Mensah, "MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction", IEEE Transactions on Software Engineering, 2018.

[12] Yuxiang Gao, Yi Zhu, Yu Zhao, "Dealing with imbalanced data for interpretable defect prediction", IEEE Conf on Data Analysis, 2017.

[13] Kwabena Ebo Bennin; Jacky Keung; Akito Monden; Yasutaka Kamei; Naoyasu Ubayashi, "Investigating the Effects of Balanced Training and Testing Datasets on Effort-Aware Fault Prediction Models", IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2016.

[14] Faruk Arar, Kürşat Ayan, "Software defect prediction using cost-sensitive neural network", Applied Soft Computing, Volume 33, August 2015.

[15] Deepika Badampudi, Claes Wohlin, Kai Petersen, "Experiences from using snowballing and database searches in systematic literature studies", 19th International Conference on Evaluation and Assessment in Software Engineering, 2015.

[16] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", Computation and Language, 2014.

[17] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", Artificial Intelligence, 2011

[18] Gul Calikli; Ayse Tosun; Ayse Bener; Melih Celik, "The effect of granularity level on software defect prediction", 24th International Symposium on Computer and Information Sciences, 2009.

[19] Cagatay Catal, Banu Diri, "A systematic review of software fault prediction studies", Expert Systems with Applications, Volume 36, Issue 4, May 2009.

[20] Alessandro Birolini, "Reliability and Availability of Repairable Systems", Reliability Engineering, 2004.