International Journal of Computer Engineering in Research Trends
Multidisciplinary, Open Access, Peer-Reviewed and fully refereed
Research Paper Volume-9, Issue-9, 2022 Regular Edition E-ISSN: 2349-7084

Software Fault Prediction Using Machine Learning Algorithms

[1] D. Himabindu, [2] K. Pranitha Kumari

[1] M.Tech.-A.I. student, Department of CSE, CVR College of Engineering, Vastunagar, Mangalpally, Ibrahimpatnam, T.S., India – 501510.

[2] Associate Professor, Department of CSE, CVR College of Engineering, Vastunagar, Mangalpally, Ibrahimpatnam, T.S., India – 501510

.Email ID: [1]himabindu150697@gmail.com, [2]k.pranithakumari@cvr.ac.in

Corresponding author : D. Himabindu

Available online at: http://www.ijcert.org

Received: 22/08/2022, Revised: 09/09/2022, Accepted: 20/09/2022, Published: 29/09/2022

Abstract: Software quality, development time, and cost can all be improved by finding and fixing bugs as soon as possible. Machine learning (ML) has been widely used for software failure prediction (SFP), but there is a wide range in how well different ML algorithms predict SFP failures. The impressive results that deep learning can produce are useful in many different fields of study, including computer vision, natural language processing, speech recognition, and many others. This investigation into Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks seeks to address the factors that may affect the performance of both methods (CNNs). The earlier software errors are found and fixed, the less time, money, and energy are wasted and the higher the likelihood of success and customer satisfaction. While machine learning (ML) and deep learning (DL) have been widely applied to SFP, the results that different algorithms produce can be somewhat inconsistent. This research uses ANN-MLP-based boosting models like XGBoost and CatBoost to enhance accuracy on NASA datasets (Artificial Neural Network-Multi Layer Perceptron). We will use a voting ensemble consisting of ANN-MLP and booster models such as XGBoost and CatBoost to increase precision.

Keywords: — Convolutional Neural Networks, software fault prediction, Artificial Neural Network-Multi Layer Perceptron, *XGBoost*

---

1. Introduction

One of the challenging tasks for software developers is the making of high-quality software. In order to produce trustworthy and high-quality software, the development process must go through a series of steps with defined boundaries. The presence of bugs is a significant downside to having high-quality, reliable software, as bugs lead to less reliable final products that fail to satisfy customers. Adequate measures for planning and controlling the software development cycle are required to provide high-quality software [1]. As with any process involving humans, software development can have hiccups at any stage. Software fault prediction, which helps to prevent learning, is one quality model that contributes to lowering software failure rates and may provide useful advancements in software fault prediction. Creating bug-free software is an arduous task. Even if a team diligently applies development processes, hidden flaws and vulnerabilities may still be uncovered.

Identifying and mitigating software's possible flaws require careful planning, management, and testing. With the help of fault prediction, the development team will be able to run tests on modules or files that have a higher fault probability many times. This will increase scrutiny of the malfunctioning components. This will enhance the likelihood that the remaining bugs can be fixed, making the final software product better suited for its intended audience. The amount of time and energy needed to keep the project running well is reduced by using this method. Evidently, software bugs are to blame for poor software quality; fixing these bugs often takes a lot of time and money; and SFP has been used to mitigate the effects of these bugs. Additionally, the SFP reduces the amount of money, time, and energy invested in software products [2]. Finding and fixing bugs are recognized as the most time-consuming and resource-intensive parts of software development [3].

Support vector machine, genetic algorithm, Artificial Neural Network (ANN) [4], decision tree [5], etc. are only a few examples of the many machine learning approaches that have been studied for their ability to anticipate software faults. There has to be more research done. What are the most popular deep learning models and how are they

optimized? Do most deep learning practitioners work within open-source frameworks? What are the most pressing issues, and what are the researchers' recommendations for the future? These are all questions that [6] tackle head-on. We were able to save time and energy thanks to their comprehensive analysis of the literature, which uncovered the best available sources and findings.

Deep learning techniques were utilized in this study, taking the drawbacks of earlier versions into mind. Deep Learning is a branch of machine learning that employs supervised and/or unsupervised methods to learn at a deep level within neural networks. The results in a number of fields where this has been tried are spectacular [7]. Multi-layered computational models can use deep learning to acquire multi-level abstractions of data [8]. Essential traits are automatically extracted from raw data, and the output becomes robust in the face of changes to the input [9]. In addition to being able to process massive volumes of data, deep learning also offers numerous models that allow for the exploitation of unlabeled data to find meaningful patterns, and the representations learnt by deep neural networks can be reused in a variety of contexts. The mathematical operation convolution is used in convolutional neural networks (CNNs) [5]. This network operates similarly to feed forward networks [10].

To lessen the computational burden, CNNs combine stacks of convolutional layers of varying sizes. Layers of maximum pooling (sub-sampling) that adhere to one or more pairs (often each pooling layer is placed after a convolutional layer). Then, we divide the remaining layer into two categories based on whether or not it is totally connected. Depending on their positions, neurons in the convolutional layer are linked to those in the layer below. In CNN, the training process consists of forward propagation, which is used to compute the actual classification of the input data with the current parameters, and back propagation, which is used to update the trainable parameters to minimize the differences between the actual and desired classification output. The back propagation algorithm is used to train CNNs. First, it assigns initial weights to each node in the network at random, and then it adjusts those weights based on the latest information. One of the benefits of CNN is that weight sharing minimizes computational requirements. At each nonlinear computation stage, max pooling is employed to limit the amount of the input data, and sub-sampling the result reduces the influence of distortion to a minimum. The number of connections, shared weights, and down sampling all decrease as the number of parameters goes down [11]. Using a CNN can improve performance, use less memory, and reduce the amount of work that needs to be done on the computer.

The remainder of the paper is organized as follows: section 2: Discuss the literature review ; section 3: Presents the problem statement ; and section 4: Discuss the Methodology of the proposed work. Section 5 deliberates the study's results and analysis; and Section 6 concludes the paper with the future scope of the research.

2. Literature Review

In this section, we describe the findings of the state-of-the-art research that has used ML, NN, and Deep Learning and we review the most relevant works that have focused on SFP. Numerous studies have shown that better resource management, bug fixes, and higher standards for overall software quality all lead to happier customers. Therefore, these studies must be investigated to ensure comprehension of SFP's facets.

A. MACHINE LEARNING

Successfully applying ML to SFP and realizing that there is still room to increase prediction accuracy, [12] proposes a CRC-based CSDP technique, ready to categorize whether the query software modules are flawed or not. We conducted experiments using ten datasets from NASA's MAP Dataset Project and compared the suggested method to others, including weighted Naive Bayes (NB), cost-sensitive boosting neural network (CBNN), compressed C4.5 decision tree (CC4.5), and coding-based ensemble learning (CEL). It seems that the provided method achieved the best results. The neuron is depicted in Figure 2 [6]. The Majority Vote-based Feature Selection algorithm (MVFS) was created by. [13] and is used to determine the most important software metrics. To evaluate MVFS's efficacy, the researchers applied four different machine learning algorithms (PC1, CM1, KC1, and JM1) that each made use of a different filter combination (Information Gain, Symmetrical Uncertainty Relief feature, and Correlation-based method).

B. NEURAL NETWORK APPROACH

In most cases, there are three main parts to a neural network. Neurons come first. To put it plainly, these are computational cells. Each neuron can take in data, process it, and then send it on to other cells as an output. In order to obtain desirable results, neural networks rely heavily on their interaction with one another. The neuron model depicted in Figure 2 consists of a collection of connecting links, each of which is described by a weight, an adder for summing the input, and an activation function. The second part is the infrastructure of the network itself. Most neural networks follow the feed forward architecture, which consists of an input layer, a hidden layer, and an output layer. Data flows from the input nodes to the hidden layers and then to the output nodes in a feed-forward network. The third component is a learning algorithm, which is used during training to define the process of adjusting the network's weights in order to decrease the output errors. Through the use of a back propagation technique [6, 7], the weights are trained iteratively by feeding erroneous signals from the output layer back into the network. A neural network's processing capacity comes from its innate intelligence and it's dispersed massively parallel architecture. Thanks to these features, it can effectively solve difficult tasks.

 https://doi.org/10.22362/ijcert/2022/v9/i9/v9i901

C. DEEP LEARNING

To predict defects, [14] propose using a convolutional neural network (DP-CNN). A convolutional neural network (CNN) is used to automatically learn the semantic and structural properties of software. Starting with Abstract Syntax Trees (ASTs)[15], tokens are extracted and encoded as numerical vectors; the next three steps include refining the extracted tokens. Then, it takes advantage of CNN and blends it with conventional defect prediction features. It employs Logic Regression to determine if there are any lingering bugs in the source code. In studies conducted across seven open-source projects, the DP-CNN [16] was found to be an average 12% improvement over the state-of-the-art approach. Automatic feature learning is utilized by [17] to create a prediction model for modeling source code in order to predict defects. They employ a long-term memory network (LSTM)[18] with a tree structure that is a perfect fit for the abstract syntax tree (AST) representation of source code (ASTs)[19]. To more accurately capture the underlying syntactic and multi-level semantics of source code, the model is constructed as a tree-structured network of LSTM units. Automatically determining whether or not new files, whether they're part of the current project or not, are flawed is now possible thanks to the model's training outcomes. Not only that, but it can pinpoint the exact lines of code in a source file that are likely to be the root of a problem. This helps to understand and know exactly what the model is taking into account and how well it works overall [20].

Their methods consist of four stages:

a. Parse the source code into an Abstract Syntax Tree (AST);
b. Embedding AST nodes to map each AST node's label name into a fixed-length continuous-valued vector;
c. Feeding the AST embedding into a tree-based network of LSTMs to get a vector representation of the whole AST; and
d. Using a classifier like Logistic Regression.

3. Problem Statement

The goal is to find the most effective machine learning algorithms for SFP and use those to get the best potential results. Based on the findings of this study, it appears that CNN and MLP networks are adequate for software fault prediction (SFP) higher accuracy, but that newer algorithms such as XGBoost and CatBoost in combination with ANN-MLP networks can further improve this accuracy. The experiments were performed on the JM1, KC1, PC1, and PC2 NASA datasets. The layers, epochs, batches, dropout rate, and optimizer are all components of this system

3. Proposed Methodology

Methodology steps to be followed as shown in figure .1

- Select Dataset
- Normalization
- Applying machine learning algorithms
- Modifying model parameters
- Comparison
- Data analysis and interpretation

Result

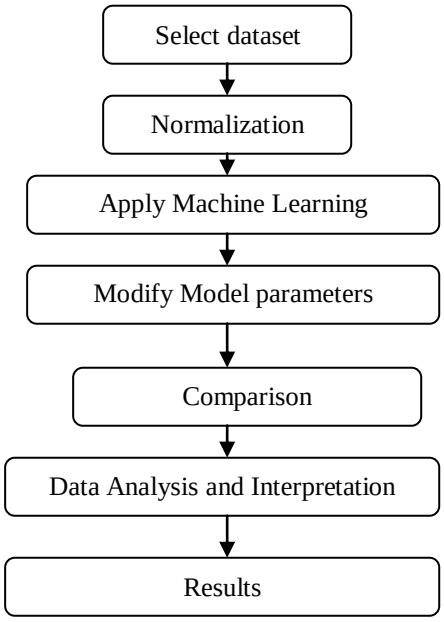

Figure 1. Proposed work Flow model

4. Methodology

4.1 Dataset

In our study, we have used 22 attributes for building our automated fault predict model. Table 1 shows 22 different attributes from software defect datasets including 21 independent metrics and one is outcome information. i.e. which is faulty and no-fault.

Table 1. Attributes of the dataset

ATTRIBUTE ID	ATTRIBUTE	DEFINITION
1	LOC	McCabe's line count of code
2	V(G)	McCabe's cyclomatic complexity
3	EV(G)	McCabe's essentional complexity
4	IV(G)	McCabe's design complexity
5	N	halstead total operators+operands
6	V	halstead volume
7	I	halstead program length
8	D	halstead difficulty
9	i	halstead intelligence
10	e	halstead effort
11	b	halstead error estimate
12	t	halstead time estimator
13	IOCode	halstead line count
14	IOComment	halstead count of lines of comment
15	IOBlank	halstead count blank lines
16	locCodeAndComment	numeric
17	Uniq_op	unique operators
18	uniq_opnd	unique operands
19	total_op	total operators
20	total_opnd	total operands
21	branchcount	flow graph
22	defects	{true,false}

 https://doi.org/10.22362/ijcert/2022/v9/i9/v9i901

The experiment result shows that MLP (XGBoost & Catboost) algorithm has achieved 82.6% accuracy than other algorithms.

4.1 SYSTEM ARCHITECTURE

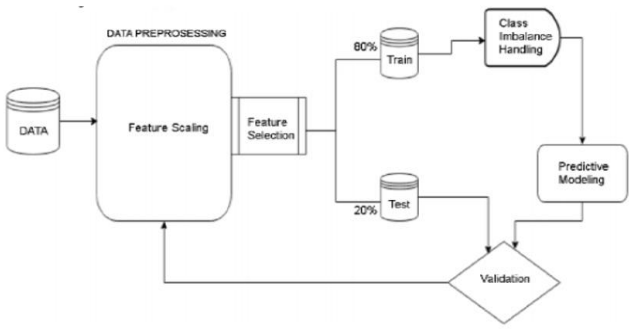

Figure 2. System architecture

The overall proposed system architecture is depicted as shown in Figure 2. The system loads the dataset as an input process and then forwards it to the data preprocessing section. Data preprocessing handles the noise of the data. Data deduplication and data filtering processes are performed by the data preprocessing. After completion of data preprocessing, it selects the features and then applies the training and testing the dataset for training 80% and testing at 20 %. After training the model, it applies the machine learning algorithms to predict the software fault and measure the performance of the proposed models with its evolution metrics such as accuracy.

5. Results and Analysis

We extended the work of the project by applying a novel algorithm known as ensemble-xgboost-catboost-MLP, which achieved an accuracy of 82.6% on the jm1 dataset. This was accomplished because XGBoost is a powerful machine learning algorithm. It has only recently been presented. It falls under the category of learning through direct instruction. The concept of gradient boosting serves as the primary foundation for it. In order to make an accurate prediction of the target, XG Boost relies on a technique known as parallel tree boosting, which combines the findings from several less accurate models. It is incredibly quick as well as accurate. Catboost can work well with its default settings, which cuts down a lot on the time needed to tune its parameters.

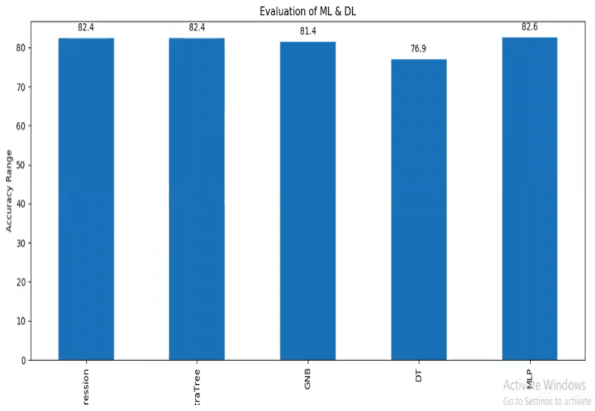

Figure 3. Performance evaluation of the Proosed Model

6. Conclusion

In this work, we demonstrate how deep learning produces concrete performance in terms of prediction across domains as diverse as computer vision, natural language processing, bioinformatics, software engineering, and more. The writers of this work set out to find the answers to two primary research questions: (1) Is there any way to improve the performance of the algorithms analysed by adjusting their parameters; and (2) which deep learning methods achieve the best SFP results? The primary goal of this research is to discover which aspects of deep learning algorithms' application to the SFP area actually affect their performance. It can be concluded from the studies that the influence of improving parameters had extraordinary impacts, leading to satisfactory outcomes, in particular with regard to the detection rate. We want to do more experiments and work with other data sets in the future to find out if the data set itself has a big effect (domain) or if it really just depends on the parameters of the algorithms.

References

[1] S. Parnerkar, A. V. Jain, and C. Birchha, ''An approach to efficient software bug prediction using regression analysis and neural networks,'' Int. J. Innov. Res. Computer. Commun. Eng., vol. 3, no. 10, Oct. 2015.

[2] A. V. Phan, M. L. Nguyen, and L. T. Bui, ''Convolutional neural networks over control flow graphs for software defect prediction,'' in Proc. IEEE 29th Int. Conf. Tools Artif. Intell. (ICTAI), Nov. 2017, pp. 45–52.

[3] E. Erturk and E. A. Sezer, ''Iterative software fault prediction with a hybrid approach,'' Appl. Soft Comput., vol. 49, pp. 1020–1033, Dec. 2016.

[4] R. Kumar and D. Gupta, ''Software Bug Prediction System Using Neural Network,'' Eur. J. Adv. Eng. Technol., vol. 3, no. 7, pp. 78–84, 2016.

[5] I. B. Y. Goodfellow and A. Courville, Deep Learning, 1st ed. Cambridge, U.K.: MIT Press, 2016.

[6] S. Haykin, Networks and Learning Machines. London, U.K.: Pearson, 2009.

[7] Y.-S. Su and C.-Y. Huang, ''Neural-network-based approaches for software reliability estimation using

dynamic weighted combinational models,'' J. Syst. Softw., vol. 80, no. 4, pp. 606–615, Apr. 2007.
[8] A. Pahal and R. S. Chillar, ''A hybrid approach for software fault prediction using artificial neural network and simplified swarm optimization,'' IJARCCE, vol. 6, no. 3, pp. 601–605, Mar. 2017.
[9] Y. LeCun and Y. H. Bengio And Hinton, ''Deep learning,'' Nature, vol. 521, no. 7553, pp. 436-444, 2015.
[10] S. Yang, L. Chen, T. Yan, Y. Zhao, and Y. Fan, ''An ensemble classification algorithm for convolutional neural network based on AdaBoost,'' in Proc. IEEE/ACIS 16th Int. Conf. Comput. Inf. Sci., May 2017, pp. 401–406.
[11] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, ''Hardware accelerated convolutional neural networks for synthetic vision systems,'' in Proc. IEEE Int. Symp. Circuits Syst., May 2010, pp. pp. 257–260.
[12] C. W. S. Jin Jin and M. J. Ye, ''Artificial neural network-based metric selection for software fault-prone prediction model,'' IET Software, vol. 6, no. 6, pp. 479–487, Dec. 2012.
[13] C. Zhang, P. Patras, and H. Haddadi, ''Deep learning in mobile and wireless networking: A survey,'' IEEE Commun. Surveys Tuts., vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019.
[14] D. Kaur, A. Kaur, S. Gulati, and M. Aggarwal, ''A clustering algorithm for software fault prediction,'' in Proc. Int. Conf. Comput. Commun. Technol. (ICCCT), Sep. 2010, pp. 603–607.
[15] M. Park and H. Hong, ''Software fault prediction model using clustering algorithms determining the number of clusters automatically,'' Int. J. Softw. Eng. Appl., vol. 8, no. 7, pp. 199–204, 2014.
[16] R. S. Wahono and N. S. Herman, ''Genetic feature selection for software defect prediction,'' Adv. Sci. Lett., vol. 20, no. 1, pp. 239–244, Jan. 2014.
[17] H. Wang, T. M. Khoshgoftaar, J. Van Hulse, and K. Gao, ''Metric selection for software defect prediction,'' Int. J. Softw. Eng. Knowl. Eng., vol. 21, no. 02, pp. 237–257, Mar. 2011.
[18] J. Li, P. He, J. Zhu, and M. R. Lyu, ''Software defect prediction via convolutional neural network,'' in Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS), Jul. 2017, pp. 318–328.
[19] H. Khanh Dam, T. Pham, S. Wee Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, ''A deep tree-based model for software defect prediction,'' 2018, arXiv:1802.00921. [Online]. Available: http://arxiv.org/abs/1802.00921
[20] S. D. Chandra, ''Software defect prediction based on classification rule mining,'' Dept. Comput. Sci. Eng., Nat. Inst. Technol. Rourkela, Rourkela, India, Tech. Rep., 2013.

 https://doi.org/10.22362/ijcert/2022/v9/i9/v9i901