ISSN: 2321-2152

IJMECE

International Journal of modern electronics and communication engineering

ISSN2321-2152 www.ijmece .com
Vol 9, Issuse.1Jan 2021

Reliability of Cloud Storage Systems in the Face of Natural Disasters using Geo-Distance Based 2-Replica Maintenance Algorithm

Sanapala Bhaskara Rao, Jami Janardhana Rao, R V L S N Sastry, Smita Rani Sahu
Assistant Professor, Associate Professor, Assistant Professor[3,4]
Department of CSE

Abstract:

Finding and correcting issues quickly may improve software quality, development time, and cost. Software failure prediction (SFP) using machine learning (ML) has become popular, although the accuracy of SFP predictions produced by various ML algorithms varies greatly. Computer vision, natural language processing, voice recognition, and many more disciplines may benefit from deep learning's remarkable outcomes. Examining what variables could influence the efficacy of Convolutional Neural Networks (CNNs) and Multi-Layer Perceptrons (MLPs) is the goal of this study. Saving time, cutting costs, and increasing the chance of success and customer happiness are all possible outcomes of finding and fixing software issues as soon as they are discovered. Although ML and DL have been extensively used for SFP, the outcomes from various algorithms may be inconsistent. The accuracy of XGBoost and CatBoost, two ANN-MLP boosting models used in this study, significantly improved when applied to NASA datasets. To enhance accuracy, we will use a voting ensemble that incorporates ANN-MLP along with booster models like XGBoost and CatBoost.

KEYWORDS: XGBoost, Artificial Neural Networks, Convolutional Neural Networks, Multi Layer Perceptrons, Software Fault Prediction

Introduction

Building reliable software is a tough nut to crack for programmers. Software development must adhere to a specified set of procedures in order to generate reliable and high-quality code. The occurrence of bugs is a major drawback of having dependable software, as they cause final products to be less trustworthy and fail to meet customer satisfaction. Software of a high quality can only be produced when the software development life cycle is well-planned and controlled [1]. Problems may arise at any point in software development, as they can in any human-driven process. There are a number of quality models that may assist reduce software failure rates and make progress in software fault prediction; one of these models is learning prevention. It is not an easy effort to create software that is devoid of bugs. Some vulnerabilities and defects may remain undetected, no matter how well a team follows development protocols.

It takes meticulous planning, management, and testing to find and fix software vulnerabilities. A development team may take use of fault prediction to repeatedly test files or modules with a greater fault probability. The components that aren't working will be examined more closely because of this. The remaining defects will be more likely to be addressed, and the final software product will be better suitable for its intended audience, if this happens. This approach reduces the time and effort required to maintain the project's smooth operation. Poor software quality is clearly caused by software defects; repairing these flaws may be a costly and time-consuming ordeal; and SFP has been used to lessen the impact of these problems. Spending less time, money, and effort creating software products is another benefit of the SFP [2]. The most labor-and resource-intensive aspects of software development are widely acknowledged to be bug finding and repairing [3].

The capacity of various machine learning techniques to predict software errors has been the subject of much research. Some of these methods include decision trees [5], support vector machines [4], genetic algorithms, and Artificial Neural Networks (ANNs). Additional investigation is necessary. How did the most well-known deep learning models get to be what they are today? optimized? Is it true that the majority of deep learning developers use open-source software? How can we address these critical challenges, and what do the experts think should be done next? The issues raised by [6] are all addressed directly. They saved us a lot of work and time by doing a thorough literature review that revealed the most reliable sources and results. This research used deep learning methods while keeping the limitations of previous versions in mind. Deep Learning is a subfield of ML that uses supervised and unsupervised techniques to train neural networks to learn very complex tasks. Impressive outcomes have been seen in several domains when this has been implemented [7]. Deep learning allows multi-layered computer models to gain data abstractions at several levels [8]. By automatically extracting essential qualities from raw data, the result becomes more resilient when the input is changed [9]. In addition to handling enormous amounts of data, deep learning provides a plethora of models for mining unlabeled data for useful patterns, and the representations learned by DNNs may be applied to many scenarios. One mathematical procedure that convolutional neural networks (CNNs) employ is [5]. Its operation is similar to that of feedforward networks [10]. Convolutional neural networks (CNNs) use stacks of convolutional layers of varied sizes to reduce the computational load. Several pairs-attached maximum-pooling (sub-sampling) layers (typically stacked one pooling layer below a convolutional layer). The last layer is then split in half according to its degree of connectivity. Connectivity between neurons in different layers of the convolutional layer is determined by their relative locations. In convolutional neural network (CNN) training, forward propagation is utilized to compute the input data's actual classification using the current parameters, and back propagation is employed to update the trainable parameters in order to minimize the differences between the actual and desired classification outputs. Convolutional neural networks (CNNs) are trained using the back propagation technique. After randomly assigning weights to all network nodes, it updates those weights using the most recent data. The fact that CNN makes use of weight sharing to reduce computing needs is one of its advantages. To minimize the impact of distortion, the output is sub-sampled after each nonlinear computing step, and max pooling is used to restrict the quantity of input data. With fewer parameters, there are fewer linkages, shared weights, and down samples [11]. By using a CNN, one may enhance performance, decrease memory use, and lessen the computational burden.

Here is how the rest of the paper is structured: Part 2 delves into a discussion of the literature review, while Part 3 introduces the issue statement and Part 4 delves into the methodology of the proposed task. Section 5 discusses the study's findings and analysis, and Section 6 wraps up the report by outlining potential areas for further research.

Literature Review

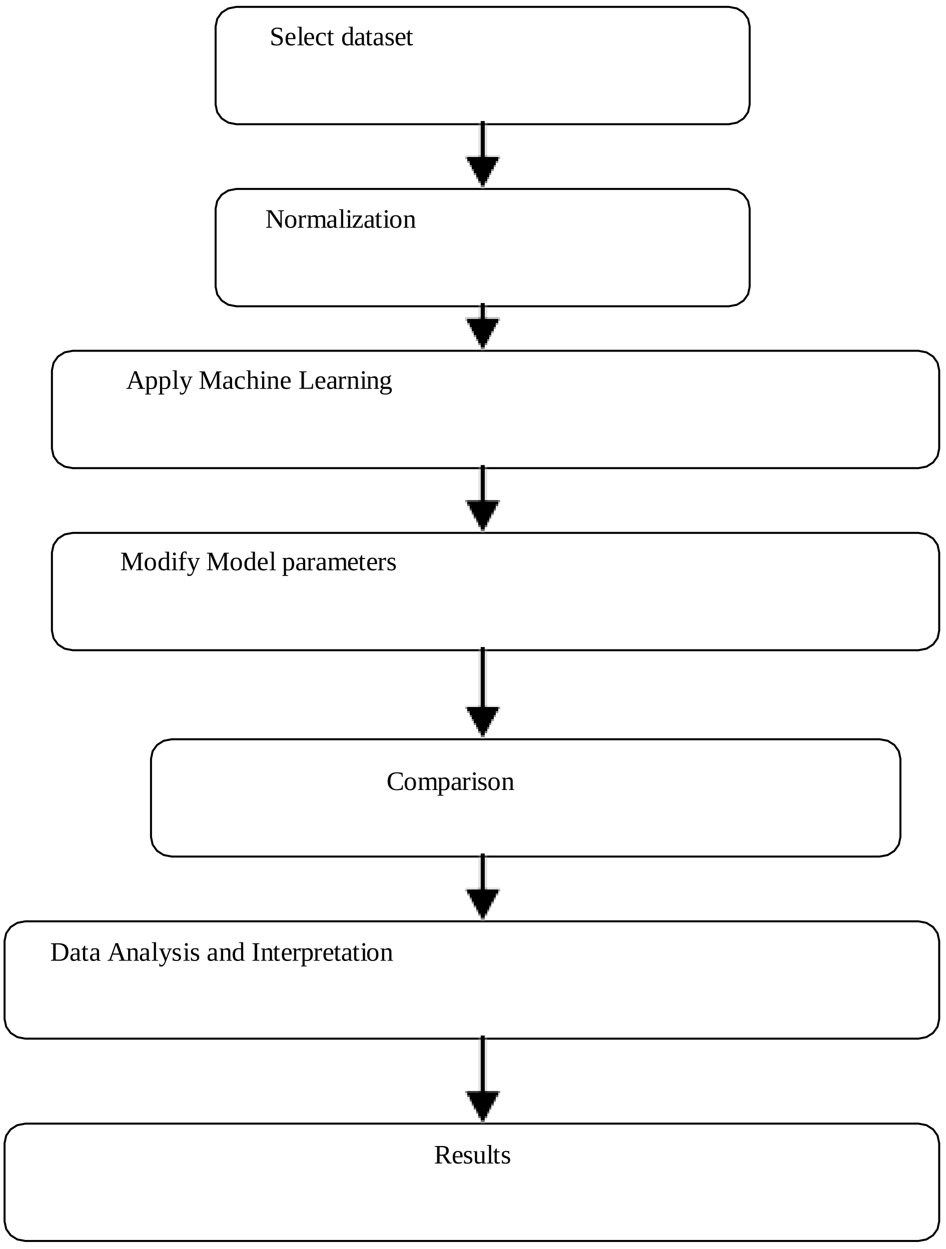
In this section, we describe the findings of the state-of- the-art research that has used ML, NN, and Deep Learning and we review the most relevant works that have focused on SFP. Numerous studies have shown that better resource management, bug fixes, and higher standards for overall software quality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

Machine Learning

Following the successful application of ML to SFP, the authors of [12] acknowledge the need to further improve prediction accuracy and provide a CRC-based CSDP approach that may classify the query software modules according to their level of defect. We compared the proposed technique to others, such as weighted Naive Bayes (NB), cost-sensitive boosting neural network (CBNN), compressed C4.5 decision tree (CC4.5), and coding-based ensemble learning (CEL), and we ran trials using 10 datasets from NASA's MAP Dataset Project. The findings appear to have been best reached by using the given technique. Fig. 2 [6] shows the neuron. In order to identify the most critical software metrics, the author developed the Majority Vote-based Feature Selection method, or MVFS [13]. Four distinct machine learning algorithms—PC1, CM1, KC1, and JM1—each using a unique mix of filters—Information Gain, Symmetrical Uncertainty Relief feature, and Correlation-based method—were used to assess MVFS's effectiveness.

The Neural Network Method

A neural network typically consists of three primary components. We start with neurons. These are computational cells, to put it simply. In a neuron, information may be input, processed, and then passed on to other cells. Neural networks depend substantially on their interconnections to achieve desired outcomes. Figure 2 shows a model of a neuron that includes a set of connecting connections, where each link is defined by a weight, an adder to sum the input, and an activation function. Part two consists of the actual network's framework. The input, hidden, and output layers that make up the feed forward architecture are the most common in neural networks. In a feed-forward network, information travels from the nodes that receive it to those that process it in the hidden layers. The third part is a learning algorithm that specifies how to change the weights of the network during training to reduce output errors. By repeatedly feeding the network incorrect signals from the output layer, the weights are learned using a back propagation approach [6, 7]. The inherent intelligence and distributed massively parallel design of a neural network provide its processing capability. Its ability to tackle challenging jobs is a direct result of these traits.



Section A: Self-Study
The use of a deep convolutional neural network (DP-CNN) for defect prediction is suggested in [14]. Software may have its semantic and structural features automatically learned by using a convolutional neural network (CNN). Tokens are first retrieved and encoded as numerical vectors using Abstract Syntax Trees (ASTs) [15]. Then, in the following three processes, the extracted tokens are refined. Afterwards, it includes CNN in its mix of traditional defect prediction characteristics. It checks the source code for remaining defects using Logic Regression. Research on seven different open-source projects indicated that the DP-CNN [16] outperformed the state-of-the-art method by an average of 12%. For the purpose of defect prediction in source code, [17] use automatic feature learning to construct a prediction model. When representing source code as an abstract syntax tree (AST)[19], they use a long-term memory network (LSTM)[18] whose tree topology is ideal. A tree-structured network of LSTM units is used to build the model, which allows it to better capture the underlying syntactic and multi-level semantics of source code. Results from training the model make it feasible to automatically detect corrupted files, regardless of whether they are part of the active project or not. Furthermore, it is capable of isolating problematic lines of code inside a source file. Insight into the model's reasoning and its overall efficacy may be gained from this [20].
There are four steps to their procedures: a. Extract the source code into an Abstract Syntax Tree (AST); b. Create a continuous-valued vector representing each AST node by embedding its label name into it; c. Feed this vector into a tree-based network of Long Short-Term Memory (LSTM) neurons to obtain a representation of the entire AST; and d. Apply a classifier, such as Logistic Regression.

Statement of the Problem

Discovering and implementing the optimal machine learning algorithms for SFP is the main objective. The results of this research suggest that although CNN and MLP networks may achieve a sufficient level of accuracy for software failure prediction (SFP), the use of more recent algorithms like XGBoost and CatBoost in conjunction with ANN-MLP networks can significantly enhance this accuracy. Datasets from NASA's JM1, KC1, PC1, and PC2 were used for the experiments. A number of parts make up this system, including layers, epochs, batches, dropout rate, while optimizer

Suggested Approach

Following the procedures shown in the diagram is the recommended methodology.1. Choose a dataset. 2. Apply machine learning techniques.
Modifying model parameters
Comparison
Data analysis and interpretation Result
Figure 1. Proposed work Flow

4 Approach Collection of Data

In order to construct our automated fault forecast model, we have used 22 features in our research. Included in the 22 features shown in Table 1 are 21 independent metrics and 1 outcome information derived from software defect databases. that is, which is defective and which is not.
Table 1: Dataset Attributes

The results of the experiment demonstrate that the MLP (XGBoost & Catboost) method outperformed the other algorithms with an accuracy of 82.6%.

4.1 SYSTEM ARCHITECTURE

ISSN2321-2152 www.ijmece .com
Vol 9, Issuse.1Jan 2021

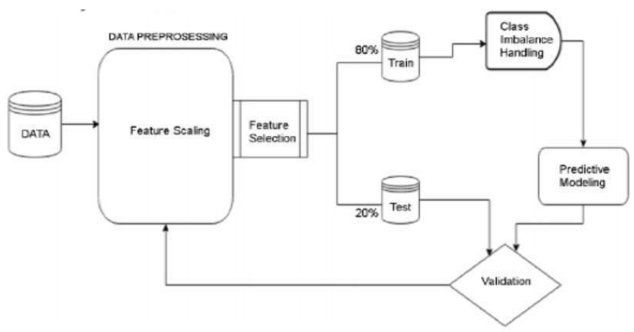
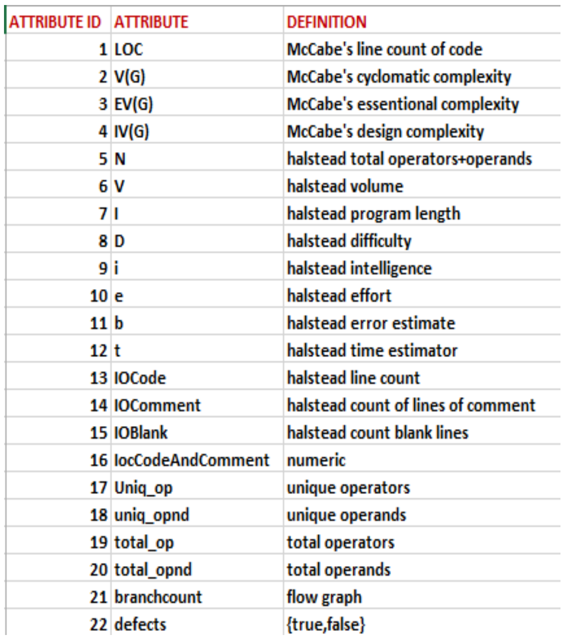


Figure 2. System architecture

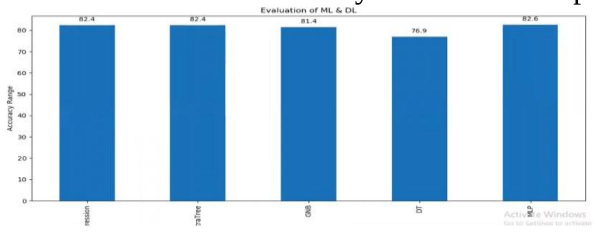


ATTRIBUTE ID	ATTRIBUTE	DEFINITION
1	LOC	McCabe's line count of code
2	V(G)	McCabe's cyclomatic complexity
3	EV(G)	McCabe's essentional complexity
4	IV(G)	McCabe's design complexity
5	N	halstead total operators+operands
6	V	halstead volume
7	l	halstead program length
8	D	halstead difficulty
9	i	halstead 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}

Figure 2 shows the proposed system architecture in its entirety. After the dataset is loaded, the system moves on to the data preparation stage as an input procedure. To deal with data noise, data preparation is necessary. As part of its preprocessing duties, data deduplication and filtering are carried out. Following the conclusion of data preprocessing, features are chosen, and the dataset is used for training with an 80% success rate and testing with a 20% success rate. Once the model has been trained, it uses machine learning techniques to forecast software faults and evaluates the suggested models based on measures like accuracy that track their progress.

Results and Analysis

After the dataset is loaded, the system moves on to the data preparation stage as an input procedure. To deal with data noise, data preparation is necessary. As part of its preprocessing duties, data deduplication and filtering are carried out. Following the conclusion of data preprocessing, features are chosen, and the dataset is used for training with an 80% success rate and testing with a 20% success rate. Once the model has been trained, it uses machine learning techniques to forecast software faults and evaluates the suggested models based on measures like accuracy that track their progress.



ISSN2321-2152 www.ijmece .com
Vol 9, Issuse.1Jan 2021

Figure 3. Performance evaluation of the Proosed Model

Conclusion

Through our work, we show that deep learning can make predictions in many different fields, including software engineering, bioinformatics, computer vision, natural language processing, and more. Two main research objectives were put forth by the authors of this work: (1) how can the algorithms that were analyzed have their parameters adjusted to perform better, and (2) which deep learning approaches provide the best SFP outcomes. Finding out what parts of applying deep learning algorithms to the SFP domain truly impact their performance is the main aim of this study. The research shows that adjusting the settings had a significant effect, which paid off with good results overall and especially in terms of the detection rate. In the future, we want to conduct further tests and collaborate with other data sets to see if the data set has a significant impact (domain) or whether it is really dependent on the algorithm settings.

References

[1] "An approach to efficient software bug prediction using regression analysis and neural networks," International Journal of Innovation Research in Computer Communication Engineering, volume 3, issue 10, October 2015, by S. Parnerkar, A. V. Jain, and C. Birchha. Proc. IEEE 29th International Conference on Tools and Artificial Intelligence (ICTAI), November 2017, pp. 45-52, "Convolutional neural networks over control flow graphs for software defect prediction" (A. V. Phan, M. L. Nguyen, and L. T. Bui, 2017).
[3] In their 2016 paper "Iterative software fault prediction with a hybrid approach," Erturk and Sezer published their findings in the Journal of Applied Soft Computing, volume 49, pages 1020–1033.
[4] "Software Bug Prediction System Using Neural Network," published in 2016 by R. Kumar and D. Gupta in the European Journal of Advanced Engineering Technology, volume 3, issue 7, pages 78–84.
[5] In Deep Learning, edited by I. B. Y. Goodfellow and A. Courville, the 2016 edition was published in Cambridge, UK, by MIT Press. Networks and Learning Machines, by S. Haykin [6]. Published by Pearson in 2009 in London, United Kingdom.
Neuronal network methods for software reliability assessment using
"dynamic weighted combinational models," published in April 2007 in the Journal of Systems Software, volume 80, issue 4, pages 606-615.
*"A hybrid approach for software fault prediction using artificial neural network and simplified swarm optimization," [1] by A. Pahal and R. S. Chillar pp. 601-605, March 2017, IJARCCE, volume 6, issue 3.
In their 2015 article "Deep learning," Y. LeCun, Y. H. Bengio, and Hinton discuss deep learning in Nature, volume 521, issue 7553, pages 436–444.
In the proceedings of the 16th International Conference on Computer Science and Information (IEEE/ACIS), May 2017, S. Yang, L. Chen, T. Yan, Y. Zhao, and Y. Fan presented "An ensemble classification algorithm for convolutional neural network based on AdaBoost," which can be found on pages 401-406.
[4] "Hardware accelerated convolutional neural networks for synthetic vision systems," in Proc. IEEE Int. Symp. Circuits Syst., May 2010, pp. 257-260, by C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello.
"Artificial neural network-based metric selection for software fault-prone prediction model," published in December 2012 in the International Journal of Engineering Software, was written by C. W. S. Jin Jin and M. J. Ye.
“Deep learning in mobile and wireless networking: A survey” by C. Zhang, P. Patras, and H. Haddadi was published in the 2019 third quarter of the IEEE Communications Surveys (vol. 21, no. 3, pp. 2224-2287).
"A clustering algorithm for software fault prediction" was published in the proceedings of

the 2010 International Conference on Computer Communication and Technology (ICCCT) in September and includes the works of D. Kaur, A. Kaur, S. Gulati, and M. Aggarwal. The article "Software fault prediction model using clustering algorithms determining the number of clusters automatically" was published in 2014 by M. Park and H. Hong in the International Journal of Software Engineering Applications. [9] "Genetic feature selection for software defect prediction," by R. S. Wahono and N. S. Herman, published in January 2014 in Adv. Sci. Lett., volume 20, issue 1, pages 239–244. In the International Journal of Software Engineering and Knowledge Engineering, the authors Wang, Khoshgoftaar, Van Hulse, and Gao (2011) published an article titled "Metric selection for software defect prediction" with pages 237–257.
“Software defect prediction via convolutional neural network” (J. Li, P. He, J. Zhu, and M. R. Lyu, 2017), Proceedings of the IEEE International Conference on Software Quality and Reliability (QRS), July 2017, pages 318–328. In their article titled "A deep tree-based model for software defect prediction," the authors C.-J. Kim, A. Ghose, T. Kim, J. Grundy, S. Wee Ng, T. Tran, and H. Khanh Dam January 2018, with the arXiv preprint number: 1802.00921.[On the web]. You may access it at: http://arxiv.org/abs/1802.00921. in [13]A technical report published in 2013 by S. D. Chandra at the Department of Computer Science and Engineering at the National Institute of Technology in Rourkela in India describes software fault prediction using classification rule mining.