Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Cloud-based bug tracking software defects analysis using deep learning

Tao Hai[1,2,3], Jincheng Zhou[1,2*], Ning Li[4,5], Sanjiv Kumar Jain[6*], Shweta Agrawal[7] and Imed Ben Dhaou[8,9,10*]

Abstract

Cloud technology is not immune to bugs and issue tracking. A dedicated system is required that will extremely error prone and less cumbersome and must command a high degree of collaboration, flexibility of operations and smart decision making. One of the primary goals of software engineering is to provide high-quality software within a specified budget and period for cloud-based technology. However, defects found in Cloud-Based Bug Tracking software's can result in quality reduction as well as delay in the delivery process. Therefore, software testing plays a vital role in ensuring the quality of software in the cloud, but software testing requires higher time and cost with the increase of complexity of user requirements. This issue is even cumbersome in the embedded software design. Early detection of defect-prone components in general and embedded software helps to recognize which components require higher attention during testing and thereby allocate the available resources effectively and efficiently. This research was motivated by the demand of minimizing the time and cost required for Cloud-Based Bug Tracking Software testing for both embedded and general-purpose software while ensuring the delivery of high-quality software products without any delays emanating from the cloud. Not withstanding that several machine learning techniques have been widely applied for building software defect prediction models in general, achieving higher prediction accuracy is still a challenging task. Thus, the primary aim of this research is to investigate how deep learning methods can be used for Cloud-Based Bug Tracking Software defect detection with a higher accuracy. The research conducted an experiment with four different configurations of Multi-Layer Perceptron neural network using five publicly available software defect datasets. Results of the experiments show that the best possible network configuration for software defect detection model using Multi-Layer Perceptron can be the prediction model with two hidden layers having 25 neurons in the first hidden layer and 5 neurons in the second hidden layer.

Keywords: Software defects, Prediction, Detection, Deep learning, Multi-layer perceptron

Introduction

The ease and speed that comes with the use of software has made it an indispensable tool in the daily lives of humans. embedded software is used in a wide range of sectors including health, finance, manufacturing, transportation, and sales among others. The importance and merits of software cannot be overemphasized- it makes the decision-making process faster, increases productivity, makes manual processes automatic, and provides an overall better customer experience. As more developments come around, software becomes more complex in order to keep up with the ever-growing and changing user requirements.

The proliferation of Internet of Things technology (IoT) has put new requirements for the design of embedded software. Three categories of devices are used in the IoT ecosystem: Low-end, middle-end, and high-end devices

*Correspondence: guideaaa@126.com; sanjivkj@gmail.com; imed.bendhaou@utu.fi

[1] School of Computer and Information, Qiannan Normal University for Nationalities, 558000 Duyun, Guizhou, China
[6] Electrical Engineering Department, Medi-Caps University, Indore, Madhya Pradesh, India
[9] Department of Computing, University of Turku, 20500 Turku, Finland
Full list of author information is available at the end of the article

[1]. Low-end devices encapsulate a set of microcontrollers that typically have a limited RAM (capacity less than 50kB) and a flash memory that doesn't exceed 250kB in size. There are three types of low-end devices: 8-bit, 16-bit, and 32-bit architecture. Several real-time operating systems have been developed for low-end devices: RIOT, Contiki, tinyOS, freeRTOS, Zepher, etc., [2].

For IoT system, the principal goal of software engineering is to produce and deliver effective and efficient software within the specified timeframe and budget. There are various non-functional features of software which can be used to determine its quality. The most important feature to consider here is its reliability [3, 4]. To assess its reliability, the amount of defects in the software have to be detected and calculated. So the lower the number of defects, the more reliable the software is. Non-functional software features that can be used to assess software quality are reliability, security, maintainability, and scalability. Scalability measures the greatest workloads that the system can handle while still meeting performance criteria. Reliability, This quality feature describes the likelihood that the system or its component will operate without failure for a certain amount of time under preset conditions. It is represented as a percentage of likelihood. The security criterion ensures that any data included within the system or its components is safe from malware assaults or unauthorized access. The time necessary for a solution or its component to be corrected, updated to improve performance or other attributes, or adapted to a changing environment is defined as maintainability There are several factors which could cause defects in software but the most common are the misinterpretation, incompleteness or unimplementation of the requirements [5].

Embedded systems are prone to security attacks, opaque, and less controllable from the end-user. Finding bugs in embedded software received scant attention by the research communities. The reasons as narrated in [6] are, among other things, the proliferation of embedded devices and the lack of forensic tools for low-end devices. The widely used approaches for the analysis of the embedded software are by obtaining device firmware, and static and dynamic firmware analyses.

A defect is essentially an error or flaw which prevents the software from meeting its requirements [7]. It is impossible to create a software that is 100% defect-free, so the only way to curb this ~ is to detect and fix as many defects as can be detected before the final product is delivered to the users. A vital stage in the production of software to ensure its quality is the phase of software testing. The early detection of the defective software components assist the software quality assurance teams in identifying the units to prioritize during the testing phase [8]. However, there are challenges associated with this stage of production. The high cost, long duration and limited resources prevent the intensive testing of every software component [9]. It is crucial to create smart tools capable of detecting software components that are defective in the testing phase [10, 11] . The prediction and detection of software defects is a fast-developing field in the research community. Software defect detection is the development of models to be used in the early testing process to pinpoint which components in the software are defective [12, 13]. It is an automatic approach which greatly enhances the testing process [14]. A survey conducted by GitLab revealed that in the software development process, the testing phase consumes the most time and causes the most delays [15]. According to the World Quality Report 2019-2020, the software testing phase takes up almost 30% of the entire software project costs [16]. A software defect is a coding fault that results in inaccurate or unexpected output from a software programme that does not fulfil actual requirements. Human aspect, communications failure, unrealistic development timetable, poor design login, faulty debuggers, poor coding methods, poor tools, lack of version control, and bugged 3rd party tools are the primary causes of software errors. Early defect detection tests are a cost-effective and accurate method of developing secure, resilient software. As defects or vulnerabilities go undiscovered, they generate a never-ending costs in terms of cost-to-fix or remediation.

The primary contribution of this research is the design of a software defect detection model using the Multi-Layer Perceptron Neural Network, a deep learning technique to improve its accuracy in detecting defects. This research also provides an evaluation of the performance of the proposed model, and uses it as a base model to assess the approach of other deep learning techniques. In summary, the contributions of this research include:

1. To examine deep learning and machine learning approaches used in software defect detection.
2. To identify public software defect datasets which can be used to train the software defect detection models
3. To propose a detection model using the Multi-Layer Perceptron Neural Network
4. To examine the performance of the proposed model and present its experimental results.

The remainder of our paper is organized as follows. Section 2 provides a literature review of the software defect development. Section 3 illustrates the main concepts that will be used in the proposed system. The performance evaluation of the proposed system are introduced in Section 4. In Section 5, the results are analysed. Finally, we conclude the paper in Section 6.

Literature review

The development of a software defect detection model with high accuracy has proven to be a herculean task. Over the past decade, several approaches have been proposed, but most of them have not been able to meet the standards in their accuracy of predicting and detecting the defects [17, 18]. According to [19], the introduction of network computing technologies like cloud computing has provided users all around the world with an affordable and flexible network-based service provision scheme. This storage service allows users to outsource their large local data to cheaper remote storage servers, reducing cost and protecting the integrity of the data. The blockchain has proven to be effective in preventing the leakage of data in 5G environments. Unnecessary reliance on a Trusted Third Party and the burden of significant overhead are some of the challenges the blockchain has mitigated, allowing the secure generation and sharing of watermarked content [20]. As a result of the widespread adoption of network storage services, there are emerging performance and security issues that affect the scalability. The high cost, reliance on third parties and repeated auditing of data are also challenges faced by the existing data auditing mechanism. To resolve this, a blockchain-based deduplication scheme was proposed to help check the data integrity and credibility of audit results in [21]. Deep learning, designed from hierarchical structure comprising multiple neural layers, has the ability to extract and learn information for generation of the reconstruction features from the input data through neural processing layer-by-layer. It has a wide range of uses including language understanding, visual recognition and threat detection in a network [22]. In [23], an efficient attribute-based scheme was proposed to prevent the breach of privacy of the access subject in the process of decision-making through the introduction of a state-of-the-art hash-based binary search tree. Renewable energy sources (RES) are of vital importance in modern power systems, but they are easily affected by the environment. Most dispatching mechanisms depend on centralized organizations, but the authors in [24] tried to resolve this by proposing a blockchain-based scheme to dispatch energy for RES systems.

Deep learning is a hot topic in computing, but building an effective deep learning model is a very challenging task, as a result of its dynamic nature, and the differences in real-world data and problems. In [25], a comprehensive review of deep learning techniques was presented, considering different types of real-world tasks like unsupervised or supervised, and their real-world application areas. Kantardzic [26] presented the state-of-the-art techniques for the analysis and extraction of information from massive amounts of data in high-dimensional data spaces, as well as an extensive view on software tools. Han and Kamber [27] and Han et al. [28] revealed how deep learning techniques have been used to solve a wide range of real-world problems with great success. In [29], the theoretical bases for the Multi-Layer Perceptron Neural Network was presented for the backpropagation learning algorithm and the architecture. The MLP model comprises of the input layer, hidden layer, and the output layer. The nodes in the MLP model are activated through the Sigmoid function by the Weka 3.8.6 tool, which is used to test software defect prediction models with varying configurations for MLP networks [30]. In [31], various metric-based bug datasets were collected and assessed in order to acquire a common set of source code metrics. The primary aim was to show how effective the dataset is in bug prediction. Cetiner and Sahingoz [32] presented a comparative and comprehensive analysis about deep learning and machine learning-based software defect prediction models through the comparison of 10 different learning algorithms on public datasets. The experimental results revealed that the proposed model showed high accuracy in the prediction of software defect and therefore, increased the quality of the software. [33] presented an Intelligent Cloud enabled Internet of Everything infrastructure as a first step in combining these two broad sectors and offering important services to end users. The Wind Driven Optimization Technique is used to enhance energy usage by clustering the different IoT networks. Rajput et al. [34] proposed a reference model for assisting diabetics in remote areas The concept enhances communications and interactions between patients and physicians. The current study's analysis goal is to analyze the risk variables and the correlations that exist among these risk factors.For prediction, Naive Bayes, SVM, random forest, logistic regression, decision tree and KNN classifiers are employed. Rupa et al. [35] A blockchain-based cloud-integrated IoT solution is offered, which can aid in the detection of intruders via virtual surveillance. The key feature of this technique is that it may work in regions where monitoring and control is difficult, and data is saved in a tamper-proof blockchain environment

Proposed method

Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP) is one of the supervised learning models used in deep learning [25]. MLP neural networks have been used to solve a variety of complex and diverse real-world problems with great success [26]. This section describes the different aspects of an MLP. This model is made up of three types of layers which are input layer, hidden layer, and output layer [27]. The MLP neural network consists of one or more hidden layers between the input layer and the output layer. Each layer is made up of small units called neurons. The artificial neurons function similarly to biological neurons where the neurons receive inputs from other neurons, process them and produce an output.

Tools

Weka 3.8.6 tool was used to train and test the software defect prediction models with different configurations for Multi-layer Perceptron neural network. Weka uses Sigmoid function as the activation function of all the nodes in the MLP neural network [29]. Sigmoid function can be defined as follows,

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

The Sigmoid function performs better in classification problems with linearly non- separable classes [11]. The sigmoid function may be exploited in complex classification functions because it provides non-linear bounds when coupled with a non-linear framework. The kernel function is better suited for the classification tasks with linearly non-separable classes because the sigmoid function is homogeneous, continuous, and differentiable everywhere and its derivative can be defined in terms of itself. Furthermore, The Sigmoid function accepts any real number as input and returns a value in the range of 0 to 1 as output. Figure 1 illustrates the Sigmoid function.

During the learning phase, the weights are adjusted using the gradient descent approach. Gradient descent algorithm is an adoption in traditional back propagation in which the network weights are shifted along the negative gradient of the response surface together with learning rate and momentum. A response to the learning issues is a weighted combination that minimize the error function. Weights are updated during the learning process using the following formulas [19]. 'w' refers to the weight assigned to a connection. 'Δw' denotes change in weight 'w' and it is calculated using (2).Gradient is determined using the back propagation algorithm. The next value of weight 'w' is denoted by 'w_{next}' and it is calculated using (3).

$$\Delta w = -learning\ rate \times gradient + momentum \times \Delta w_{previous} \tag{2}$$

$$w_{next} = w + \Delta w \tag{3}$$

The learning rate and momentum are applied to update the weights during the learning process. The learning rate and momentum were assigned fixed values.

Learning rate is a hyper parameter that indicates how well the model should react differently to the predicted error each time the system model weights are adjusted. Momentum can help to expedite training, and learning rate plans can aid in the optimization process. According to the problem, the momentum and the learning rate both are allocated definite values in order to achieve stable convergence.

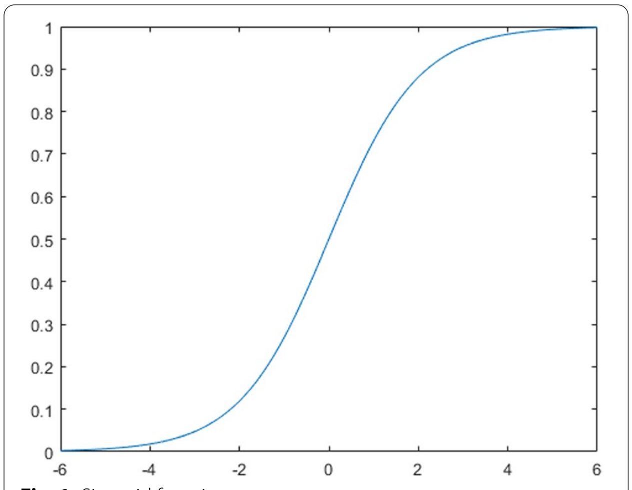


Fig. 1 Sigmoid function

Datasets

This study considered using public datasets to train and verify the proposed software defect detection model. Furthermore, the literature suggests exploring and using new datasets for developing software defect detection models. Therefore, this research also considered investigating and using new software defect datasets to train and test the proposed model. Five different public datasets formed by [31] were selected as one of the main concerns of this study is to use new software defect datasets. Three datasets were selected from Tera-Promise and two datasets were selected from GitHub Bug Repository. The datasets selected from Tera-Promise are Xalan 2.6, Velocity 1.5 and Poi 3.0. Netty 3.6.3 and mcMMO 1.4.06 are the two datasets chosen from GitHub Bug Repository. The properties of the selected datasets that include the number code metrics found in the original dataset, the number of code metrics calculated using OSA, total number of instances, number of defective instances and number of non-defective instances are illustrated in Table 1.

Proposed WORKFLOW

The workflow used for proposing a software defect prediction model using MLP is illustrated in Fig. 2. The proposed workflow mainly consists of four steps. First step is dataset selection. The second step is generation of the prediction model using a specific MLP network configuration. Training and testing the generated prediction model using training and test datasets is performed in third step. The final stage involves performance evaluation of the software defect prediction model.

Table 1 Properties of the selected datasets

Data Source	Dataset	Number of metrics in the original dataset	Number of metrics calculated with OSA	Number of instances	Number of defective instances	Number of non-defective instances
Tera-promise	Xalan 2.6	22	60	885	411 (46.44%)	474 (53.56%)
	Velocity 1.5	22	60	213	141 (66.29%)	72 (33.71%)
	Poi 3.0	22	60	442	281 (63.57%)	161 (36.43%)
GitHub Bug Repository	Netty 3.6.3	60	60	1143	271 (23.71%)	1072 (76.29%)
	mcMMO 1.4.06	60	60	301	57 (18.94%)	244 (81.06%)

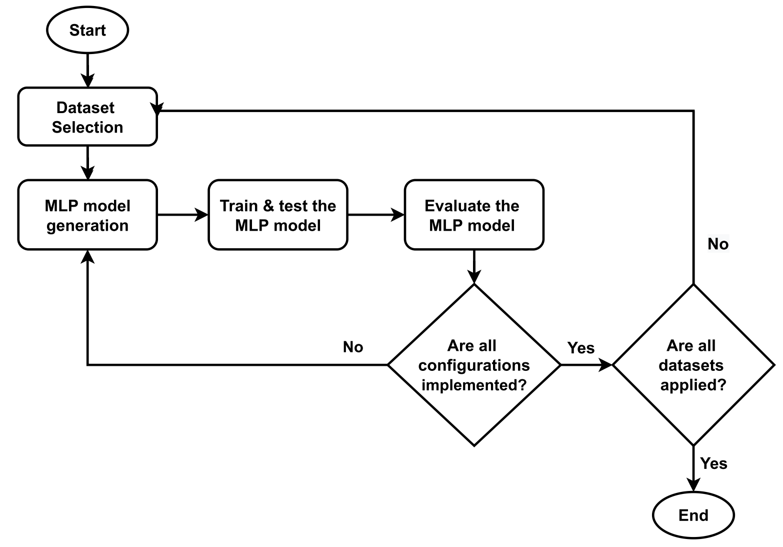

Start
Dataset Selection
MLP model generation
Train & test the MLP model
Evaluate the MLP model
Are all configurations implemented?
No
Yes
Are all datasets applied?
No
Yes
End

Fig. 2 Proposed workflow

According to [28] there are no hard and fast guidelines for deciding the ideal number of hidden layers and the ideal number of neurons in each layer. Therefore, this research conducted an experiment with 4 different network configurations on the five selected datasets to determine the best possible number of hidden layers and the number of neurons in each hidden layer. The following guidelines were followed when deciding the combination of the number of hidden layers and the number of neurons in each hidden layer.

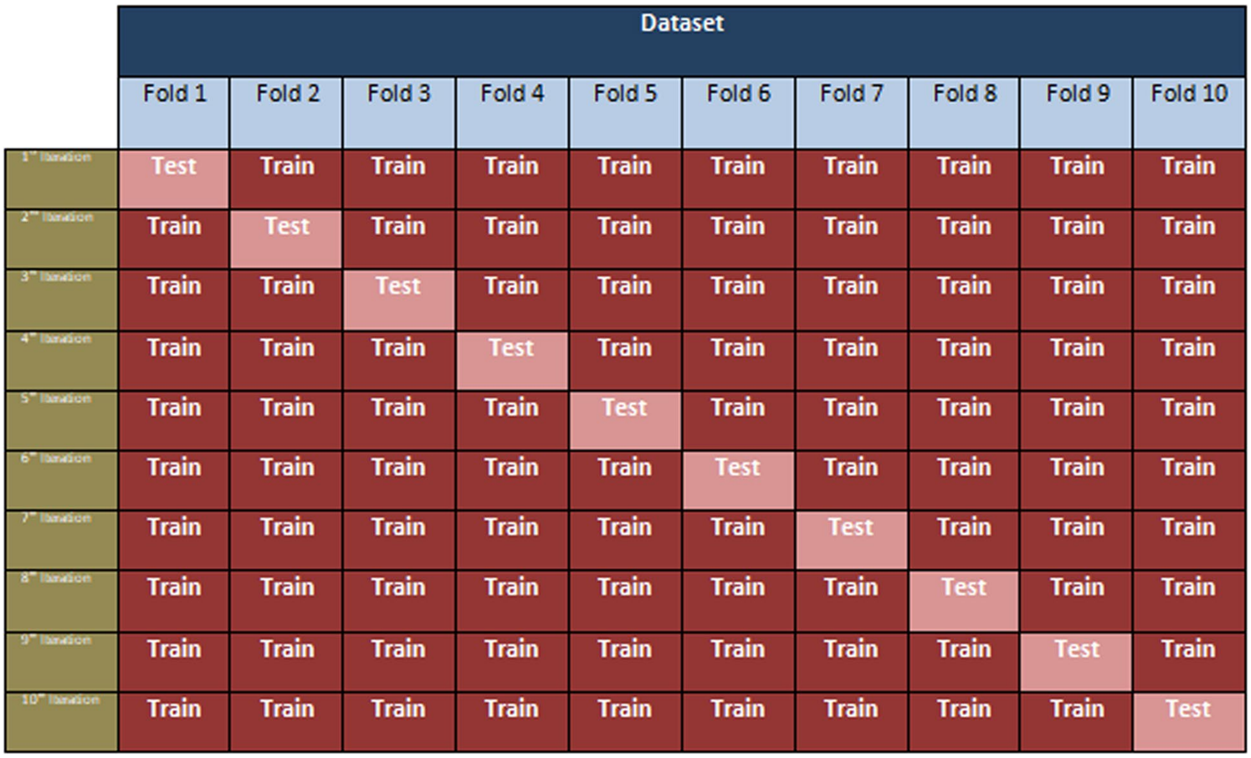
- The experiment was conducted setting the number of hidden layers to 2 and 3
- The number of neurons in each hidden layer was selected between the number of neurons in the input layer (60) and the number of neurons in the output layer (2)
- The number of neurons was gradually decreased from the first hidden layer to the last hidden layer.

Performance evaluation

This study utilized k-fold cross-validation approach. Furthermore, the value of k was set to 10 and therefore, 10-fold cross validation was used to get an evaluation result and estimate of the error.

In 10-fold cross validation, the original dataset is divided into equal sizes of 10 sub datasets randomly [28]. The training and testing are repeated 10 times for a selected dataset. Figure 3 illustrates the procedure of 10-fold cross validation.

Furthermore, the process of 10-fold cross-validation can be described as follows. In the first iteration, the first subset of data is used as the testing dataset while the rest of the 9 subsets of data are used as training datasets. In the second iteration, the second subset of data is used as the testing dataset while the remaining 9 sub datasets are used for training the model. This process is repeated 10 times. It produces 10 results, and an average result is computed as the final result. In k-fold



	Dataset									
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
1st Iteration	Test	Train	Train	Train	Train	Train	Train	Train	Train	Train
2nd Iteration	Train	Test	Train	Train	Train	Train	Train	Train	Train	Train
3rd Iteration	Train	Train	Test	Train	Train	Train	Train	Train	Train	Train
4th Iteration	Train	Train	Train	Test	Train	Train	Train	Train	Train	Train
5th Iteration	Train	Train	Train	Train	Test	Train	Train	Train	Train	Train
6th Iteration	Train	Train	Train	Train	Train	Test	Train	Train	Train	Train
7th Iteration	Train	Train	Train	Train	Train	Train	Test	Train	Train	Train
8th Iteration	Train	Train	Train	Train	Train	Train	Train	Test	Train	Train
9th Iteration	Train	Train	Train	Train	Train	Train	Train	Train	Test	Train
10th Iteration	Train	Train	Train	Train	Train	Train	Train	Train	Train	Test

Fig. 3 10-fold cross-validation technique

cross validation, each subset of data is utilized one time for testing and an equal number of times for training.

As illustrated in Fig. 4, a confusion matrix for the binary classification problem produces 4 outcomes which are True Positive (TP), False Negative (FN), False Positive (FP) and True Negative (TN) [32].

- True Positive (TP) presents the number of positive instances which were correctly predicted as positive [1];
- False Negative (FN) presents the number of positive instances which were incorrectly predicted as negative [1];
- False Positive (FP) presents the number of negative instances which were incorrectly predicted as positive [1];
- True Negative (FN) presents the number of instances which were correctly predicted as negative [1].

Therefore, True Positive (TP) and True Negative (FN) reflect that the prediction has been made correctly whereas False Positive (FP) and False Negative (FN) indicate that the prediction has been made incorrectly.

The proposed configurations of MLP neural network for software defect detection were evaluated using several performance measures which are accuracy, precision, recall, F-measure and ROC Area. The Weka tool provides all these performance indicators. Some of the established performance metrics applied for measurements are as follows;

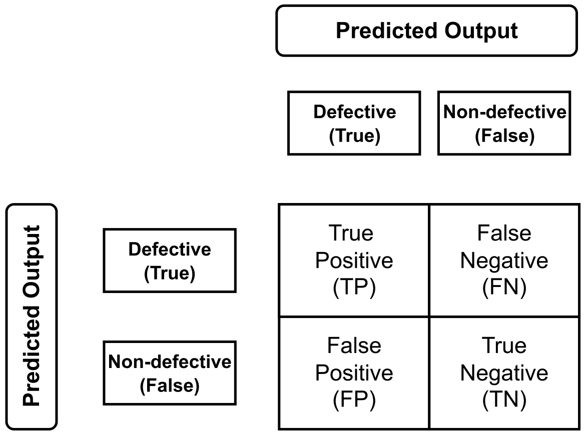


Fig. 4 Confusion Matrix

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

The score of 1.0 reflects a perfect recall whereas the score of 0.0 reflects the worst recall. A higher value of recall means the prediction model has higher ability of detecting true positives as positive.

$$Precision = \frac{TP}{TP + FP} \quad (6)$$

The perfect precision is defined by the score of 1.0 whereas the worst precision is defined by the score of 0.0. A higher value of precision reflects that the prediction model shows a low rate of incorrectly classifying negative instances as positive.

Since it is clear that neither recall nor precision can give a complete measure on their own. F-Measure combines both recall and precision into a single measure by calculating the harmonic mean of them. Therefore, F-Measure gives equal weights for both recall and precision which is more important when imbalanced data is used for training.

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (7)$$

Results analysis

This section analyses the experimental results obtained for all MLP network configurations on the five selected datasets. The performance of each MLP architecture was evaluated using accuracy, precision, recall, F-Measure and ROC Area. Table 2 illustrates the percentage of prediction accuracy of each MLP network configuration on each dataset.

The MLP model with two hidden layers and 25 and 5 neurons in the first and second hidden layers, respectively, shows the highest prediction accuracy for Xalan 2.6, Velocity 1.5, Poi 3.0 datasets. However, the MLP model with the same network configuration shows the lowest prediction accuracy for Netty 3.6.3 dataset. The MLP which consists of three hidden layers and 15 and 10 nodes in the first and second hidden layers respectively and 5 nodes in the final hidden layer has performed better with higher prediction accuracy in Netty 3.6.3 dataset. The average accuracy of this MLP model is 78.2120%.

Table 3 shows the evaluation results including precision, recall and F-Measure for all considered MLP network configurations on Xalan 2.6 dataset. The results present the precision, recall and F-Measure for each target class as well as the weighted average of per-class values. According to the findings, in precision, recall and F-Measure, the MLP model with two hidden layers and 25 and 5 neurons in each hidden layer has performed well in both output classes when compared to the other configurations of MLP neural network.

Table 4 presents the evaluation results on Velocity 1.5 dataset. It shows that in precision, the MLP with two hidden layers and 15 and 5 neurons in first and second hidden layers respectively performed well for the true class whereas the MLP with two hidden layers and 25 and 5 neurons in each hidden layer performed well in the false class. In recall, vice versa has happened for true and false classes. However, when the weighted average in recall and precision are considered, the MLP with two hidden layers and 25 and 5 neurons in each hidden layer performed well for Velocity 1.5 dataset.

The evaluation results of Poi 3.0 dataset are provided in Table 5. According to the results, the MLP neural network with two hidden layers and 25 and 5 neurons in the first and second hidden layers respectively has shown the best performance in recall, precision and F-Measure in both true and false classes when compared to other network configurations.

The results obtained for Netty 3.6.3 dataset with each MLP model is presented in the Table 6. It is clear that in precision, recall and F-Measure, the MLP with three hidden layers having 15, 10 and 5 nodes from the first hidden layer to the third hidden layer respectively has well detected both true and false classes.

Table 2 Prediction accuracy results

Dataset	H1=15, H2=5	H1=25, H2=5	H1=15, H2=10, H3=5	H1=30, H2=15, H3=5
Xalan 2.6	73.2203	75.2203	74.3503	72.8814
Velocity 1.5	75.1174	77.4648	75.1174	76.0563
Poi 3.0	72.8507	76.0181	73.9819	73.0769
Netty 3.6.3	80.4899	79.9650	81.7148	80.7524
mcMMo 1.4.06	84.7176	82.3920	83.0565	81.0631
Average Accuracy	77.2792	78.2120	77.6342	76.7660

Table 3 Results obtained for Xalan 2.6 dataset

MLP Model	Precision			Recall			F-Measure		
	True	False	Weighted Average	True	False	Weighted Average	True	False	Weighted Average
H1=15 H2=5	0.779	0.707	0.740	0.591	0.854	0.732	0.672	0.774	0.727
H1=25 H2=5	0.796	0.731	0.761	0.635	0.859	0.755	0.706	0.790	0.751
H1=15 H2=10 H3=5	0.774	0.725	0.748	0.633	0.840	0.744	0.696	0.778	0.740
H1=30 H2=15 H3=5	0.773	0.705	0.736	0.589	0.850	0.729	0.669	0.771	0.723

Table 4 Results obtained for Velocity 1.5 dataset

MLP Model	Precision			Recall			F-Measure		
	True	False	Weighted Average	True	False	Weighted Average	True	False	Weighted Average
H1=15 H2=5	0.879	0.598	0.784	0.723	0.806	0.751	0.794	0.686	0.757
H1=25 H2=5	0.855	0.646	0.784	0.794	0.736	0.755	0.824	0.688	0.778
H1=15 H2=10 H3=5	0.838	0.614	0.763	0.773	0.708	0.751	0.804	0.658	0.755
H1=30 H2=15 H3=5	0.852	0.624	0.774	0.773	0.736	0.761	0.810	0.675	0.765

Table 5 Results obtained for Poi 3.0 dataset

MLP Model	Precision			Recall			F-Measure		
	True	False	Weighted Average	True	False	Weighted Average	True	False	Weighted Average
H1=15 H2=5	0.813	0.611	0.739	0.744	0.702	0.729	0.777	0.653	0.732
H1=25 H2=5	0.840	0.649	0.771	0.769	0.745	0.760	0.803	0.694	0.763
H1=15 H2=10 H3=5	0.832	0.620	0.755	0.740	0.739	0.740	0.783	0.674	0.744
H1=30 H3=5	0.805	0.619	0.737	0.762	0.677	0.731	0.782	0.647	0.733

Table 7 illustrates the results of mcMMO 1.4.06 dataset with all experimented MLP model configurations. As per the findings, the MLP neural network with two hidden layers, with 15 and 5 nodes in each hidden layer has relatively high precision, recall and F-Measure in both true and false classes when compared to other MLP network configurations.

According to precision, recall and F-Measure, the MLP neural network with three hidden layers, with 30 neurons in the first hidden layer, 15 neurons in the second hidden layer and 5 neurons in the final hidden layer has not shown higher performance in any of the datasets.

According to ROC Area results presented in Table 8, the MLP neural network with two hidden layers and 25 neurons in the first hidden layer and 5 neurons in the

Table 6 Results obtained for Netty 3.6.3 dataset

MLP Model	Precision			Recall			F-Measure		
	True	False	Weighted Average	True	False	Weighted Average	True	False	Weighted Average
H1=15 H2=5	0.620	0.844	0.791	0.458	0.913	0.805	0.527	0.877	0.794
H1=25 H2=5	0.597	0.847	0.788	0.476	0.900	0.800	0.530	0.873	0.791
H1=15 H2=10 H3=5	0.646	0.856	0.806	0.506	0.914	0.817	0.567	0.884	0.809
H1=30 H2=15 H3=5	0.620	0.846	0.794	0.456	0.914	0.808	0.534	0.879	0.797

Table 7 Results obtained for mcMMO 1.4.06 dataset

MLP Model	Precision			Recall			F-Measure		
	True	False	Weighted Average	True	False	Weighted Average	True	False	Weighted Average
H1=15 H2=5	0.622	0.887	0.837	0.491	0.930	0.847	0.549	0.908	0.840
H1=25 H2=5	0.542	0.877	0.814	0.476	0.910	0.824	0.495	0.893	0.818
H1=15 H2=10 H3=5	0.568	0.875	0.817	0.439	0.922	0.831	0.495	0.898	0.822
H1=30 H2=15 H3=5	0.500	0.867	0.797	0.404	0.906	0.811	0.447	0.886	0.803

Table 8 ROC Area results

MLP Model	Xalan 2.6	Velocity 1.5	Poi 3.0	Netty 3.6.3	mcMMO 1.4.06
H1=15, H2=5	0.808	0.801	0.794	0.754	0.774
H1=25, H2=5	0.822	0.794	0.816	0.777	0.756
H1=15, H2=10, H3=5	0.803	0.772	0.785	0.783	0.764
H1=30, H2=15, H3=5	0.791	0.766	0.766	0.789	0.749

second hidden layer has performed better in Xalan 2.6 and Poi 3.0 datasets in discovering between the defective and non-defective classes. For Velocity 1.5 dataset, the MLP with two hidden layers having 15 nodes and 5 nodes respectively in the first and second hidden layers has produced the highest ROC Area. Furthermore, the MLP with three hidden layers with 15, 10 and 5 nodes from the first hidden layer to the last hidden layer respectively has shown a higher ROC Area on mcMMO 1.4.06dataset. The MLP with three hidden layers and 30 neurons in the first hidden layer, 15 neurons in the second hidden layer and 5 neurons in the last hidden layer has performed well with ROC Area value of 0.789 on Netty 3.6.3 dataset.

ROC curves generated by the MLP with two hidden layers and 25 neurons in the first hidden layer and 5 neurons in the second hidden layer for true class of Xalan 2.6, Velocity 1.5, Poi 3.0, Netty 3.6.3 and mnMMO 1.4.06 datasets are presented in Figs. 5, 6, 7, 8 and 9 respectively. X-axis in each plot represents False Positive rate while Y-axis represents True Positive rate.

After analysing the results obtained for all performance metrics including accuracy, precision, recall, F-Measure and ROC Area as well as considering the

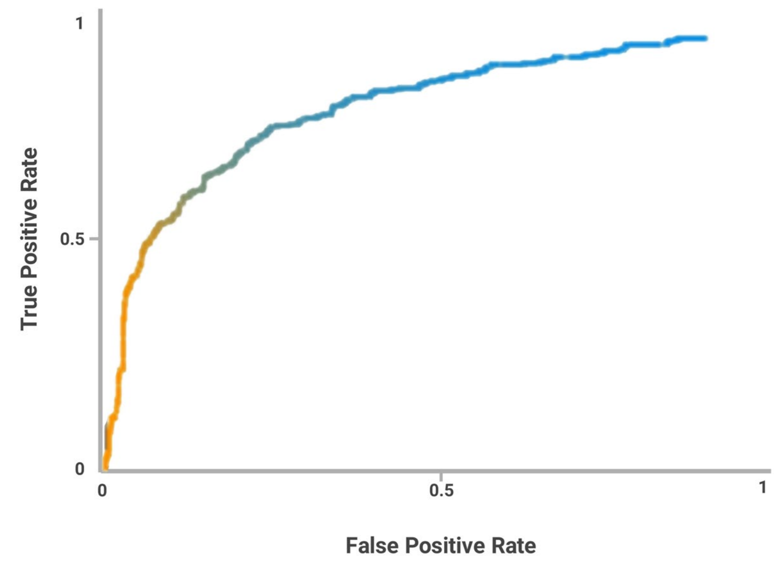


Fig. 5 ROC curve of Xalan 2.6

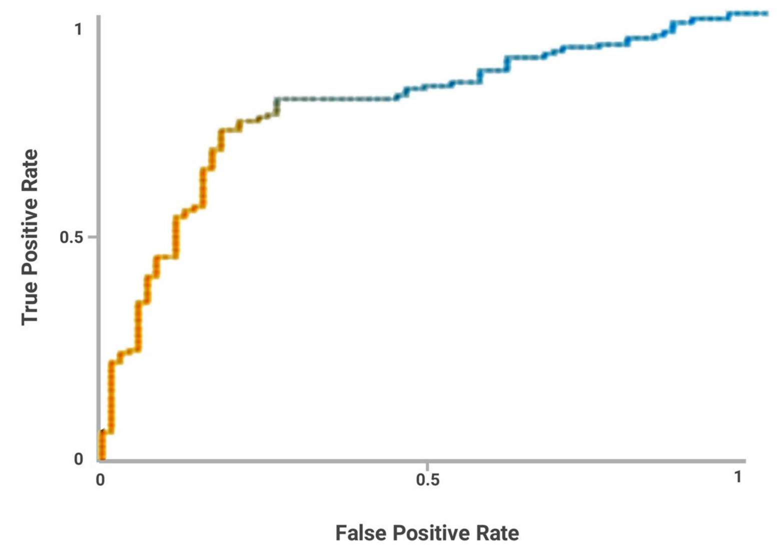


Fig. 6 ROC curve of Velocity 1.5

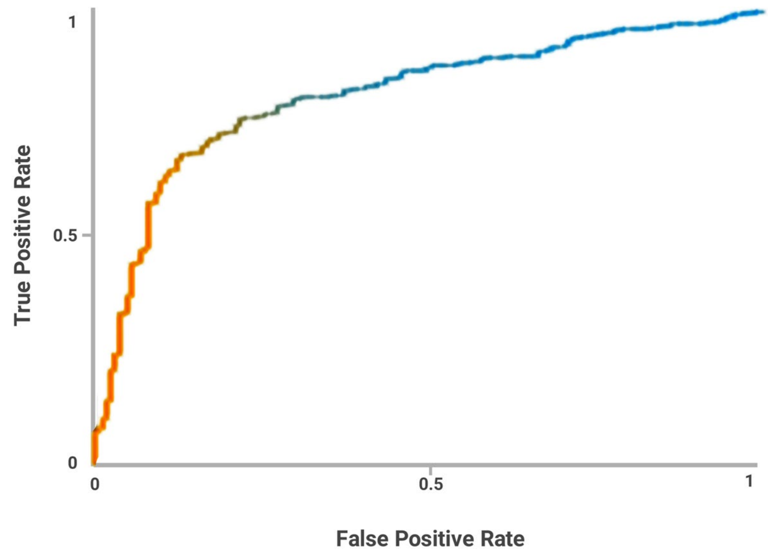


Fig. 7 ROC curve of Poi 3.0

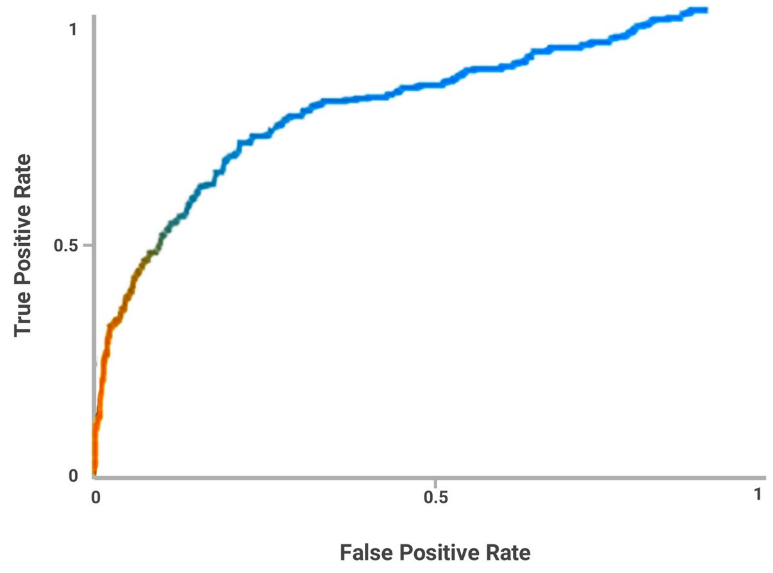


Fig. 8 ROC curve of Netty 3.6.3

class imbalance issue existing in the datasets, this study has discovered that the best possible network configuration for software defect detection model using MLP can be the prediction model with two hidden layers having 25 neurons in the first hidden layer and 5 neurons in the second hidden layer. Figure 10 illustrates proposed MLP architecture.

Conclusion

The field of software engineering mainly focuses on delivering high-quality software within a specified budget and period. However, the defects found in the software can cause delayed the delivery process while reducing the quality of software. Therefore, early detection of defects in the software being developed

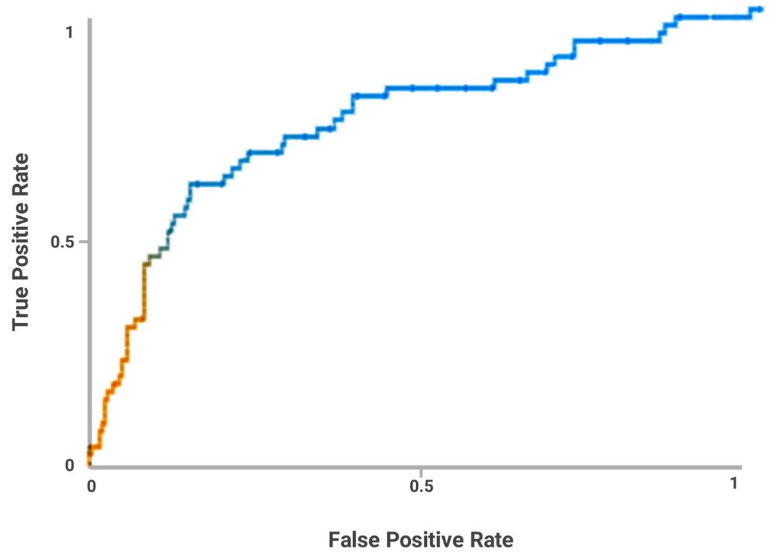


Fig. 9 ROC curve of mcMMO 1.4.06

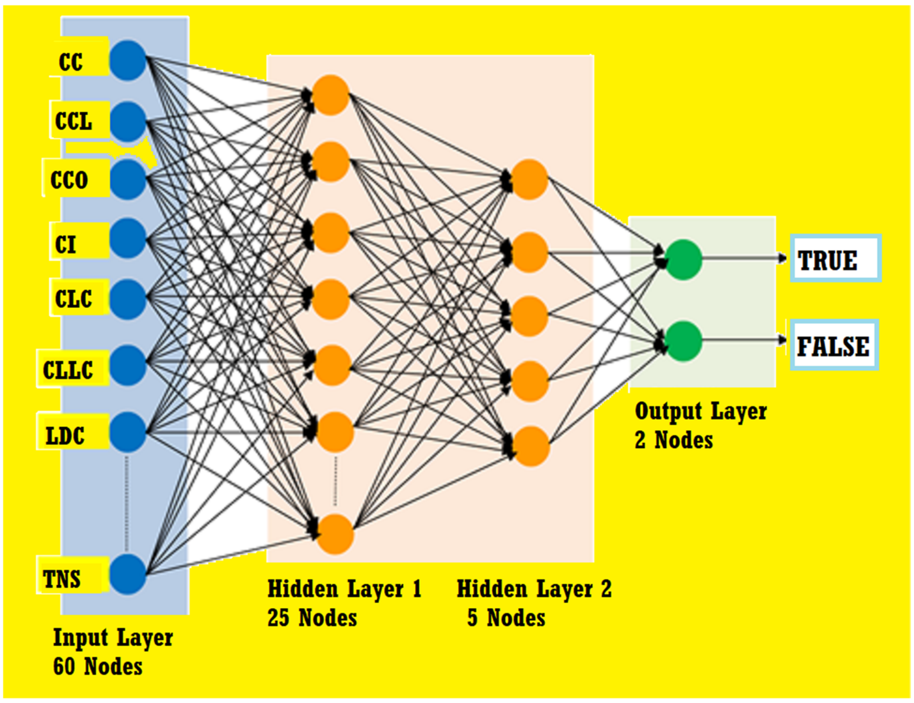


Fig. 10 Proposed architecture of MLP model for software defect prediction

and fixing before it is delivered to the end users is a crucial task. Due to the complexity of user requirements as well as the infeasibility of producing defect-free software, the software testing process requires high cost and time to deliver quality software. Early detection of defect prone modules in the software being developed assists software quality assurance teams to identify which modules require more attention during the testing process and thereby make use of available resources for software testing efficiently and effectively. Automation of early detection of defective modules in software has been an active research area for many years. With the evolution of different technologies, software defect detection has also become an emerging research area. Various approaches including statistical methods, machine learning techniques and deep learning techniques have been applied for software defect detection. This study dealt with the investigation of one of the emerging areas in AI which is deep learning for software defect detection. This research analysed the performance of Multi-layer Perceptron neural network, which is a supervised deep learning technique for software defect detection. An experiment with four different network configurations of MLP neural network was conducted to propose the best possible MLP architecture among them for software defect detection. The performance of the four MLP neural network configurations was evaluated using several metrics computed using a confusion matrix. The used evaluation metrics are accuracy, precision, recall, F-Measure and ROC area. Among the MLP network configurations used for the experiment, this study has discovered that the best possible network configuration for the software defect detection model using MLP can be the prediction model with two hidden layers having 25 neurons in the first hidden layer and 5 neurons in the second hidden layer. The study concludes that the proposed MLP neural network performs better on 3 out of 5 used datasets when compared to the other network configurations. However, this study concludes that more empirical studies should be conducted to assess the performance of the proposed software defect detection model and thereby help to refine it.

Acknowledgements
The authors express their appreciation to the Natural Science Basic Research Program of Shaanxi Province, China.

Authors' contributions
Conceptualization by Tao Hai; Methodology by Jincheng Zhou; Software by Ning Li and Imed Ben Dhaou; Formal analysis by Imed Ben Dhaou and Sanjiv Kumar Jain; Investigation by Tao Hai and Shweta Agrawal; Resources and data collection by Imed Ben Dhaou and Sanjiv Kumar Jain; Writing by: Tao Hai, Imed Ben Dhaou and Jincheng Zhou; Validation by: Ning Li and Imed Ben Dhaou; Funding Acquisition by Ning Li; All authors read and approved the final manuscript.

Funding
The project was supported by the Natural Science Basic Research Program of Shaanxi Province No.2022JQ-689.

Availability of data and materials
The supporting data can be provided on request.

Declarations

Ethical approval and consent to participate
The research has consent for Ethical Approval and Consent to participate.

Consent for publication
The research has research consent by all authors and there is no conflict.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]School of Computer and Information, Qiannan Normal University for Nationalities, 558000 Duyun, Guizhou, China. [2]Key Laboratory of Complex Systems and Intelligent Optimization of Guizhou, 558000 Duyun, Guizhou, China. [3]Institute for Big Data Analytics and Artificial Intelligence (IBDAAI), Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia. [4]School of Computer Science, Baoji University of Arts and Sciences, 721016 Baoji, Shaanxi, China. [5]Faculty of Computer Science and Engineering, Xi'an University of Technology, 710048 Xi'an, Shaanxi, China. [6]Electrical Engineering Department, Medi-Caps University, Indore, Madhya Pradesh, India. [7]Institute of Advance Computing, SAGE University, Indore, India. [8]Department of Computer Science, Hekma School of Engineering, Computing, and Informatics, Dar Al-Hekma University, 22246-4872 Jeddah, Saudi Arabia. [9]Department of Computing, University of Turku, 20500 Turku, Finland. [10]Higher Institute of Computer Sciences and Mathematics, Department of Technology, University of Monastir, 5000 Monastir, Tunisia.

Received: 29 June 2022 Accepted: 25 July 2022
Published online: 30 August 2022

References
1. Ojo MO, Giordano S, Procissi G, Seitanidis IN (2018) A review of low-end, middle-end, and high-end iot devices. IEEE Access 6:70528–70554. https://doi.org/10.1109/ACCESS.2018.2879615
2. Silva M, Cerdeira D, Pinto S, Gomes T (2019) Operating systems for internet of things low-end devices: Analysis and benchmarking. IEEE Internet Things J 6(6):10375–10383. https://doi.org/10.1109/JIOT.2019.2939008
3. Reddivari S, Raman J (2019) Software quality prediction: an investigation based on machine learning. In: 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI). IEEE, pp 115-122
4. Liggesmeyer P, Trapp M (2009) Trends in embedded software engineering. IEEE Softw 26(3):19–25. https://doi.org/10.1109/MS.2009.80
5. Anajemba JH, Iwendi C, Razzak I, Ansere JA, Okpalaoguchi IM (2022) A counter-eavesdropping technique for optimized privacy of wireless industrial iot communications. IEEE Trans Ind Inform 18(9):6445–6454. https://doi.org/10.1109/TII.2021.3140109
6. Francillon A, Thomas SL, Costin A (2021) Finding Software Bugs in Embedded Devices. Springer International Publishing, Cham, pp 183–197. https://doi.org/10.1007/978-3-030-10591-4_11
7. Istqb glossary (2019). https://glossary.istqb.org/en/search/. Accessed 20 Mar 2022
8. Iqbal A, Aftab S, Ali U, Nawaz Z, Sana L, Ahmad M, Husen A (2019) Performance analysis of machine learning techniques on software defect prediction using nasa datasets. Int J Adv Comput Sci Appl 10(5)
9. Jiang P (2021) Research on software defect prediction technology based on deep learning. In: 2021 2nd International Conference on Computing and Data Science (CDS). IEEE, pp 104-107
10. Iwendi C, Khan S, Anajemba JH, Bashir AK, Noor F (2020) Realizing an efficient iomt-assisted patient diet recommendation system through machine learning model. IEEE Access 8:28462–28474. https://doi.org/10.1109/ACCESS.2020.2968537
11. Samir M, El-Ramly M, Kamel A (2019) Investigating the use of deep neural networks for software defect prediction. In: 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA). IEEE, pp 1-6
12. Prabha CL, Shivakumar N (2020) Software defect prediction using machine learning techniques. In: 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184). IEEE, pp 728-733
13. Iwendi C, Khan S, Anajemba JH, Mittal M, Alenezi M, Alazab M (2020b) The use of ensemble models for multiple class and binary class classification for improving intrusion detection systems. Sensors 20(9). https://www.mdpi.com/1424-8220/20/9/2559

14. Tian Z, Xiang J, Zhenxiao S, Yi Z, Yunqiang Y (2019) Software defect prediction based on machine learning algorithms. In: 2019 IEEE 5th International Conference on Computer and Communications (ICCC). IEEE, pp 520-525. https://doi.org/10.1109/ICCC47050.2019.9064412
15. Why software testing remains a bottleneck – the new stack. https://thenewstack.io/why-software-testing-remains-a-bottleneck/. Accessed 25 Mar 2022
16. Taking a new approach to reducing software testing costs | itproportal. https://www.itproportal.com/features/taking-a-new-approach-to-reducing-software-testing-costs/. Accessed 15 Feb 2022
17. Akimova EN, Bersenev AY, Deikov AA, Kobylkin KS, Konygin AV, Mezentsev IP, Misilov VE (2021) A survey on software defect prediction using deep learning. Mathematics 9(11):1180
18. Iwendi C, Anajemba JH, Biamba C, Ngabo D (2021) Security of things intrusion detection system for smart healthcare. Electronics 10(12). https://www.mdpi.com/2079-9292/10/12/1375
19. Xu Y, Ren J, Zhang Y, Zhang C, Shen B, Zhang Y (2020) Blockchain empowered arbitrable data auditing scheme for network storage as a service. IEEE Trans Serv Comput 13(2):289–300. https://doi.org/10.1109/TSC.2019.2953033
20. Xu Y, Zhang C, Zeng Q, Wang G, Ren J, Zhang Y (2021) Blockchain-enabled accountability mechanism against information leakage in vertical industry services. IEEE Trans Netw Sci Eng 8(2):1202–1213. https://doi.org/10.1109/TNSE.2020.2976697
21. Xu Y, Zhang C, Wang G, Qin Z, Zeng Q (2021) A blockchain-enabled deduplicatable data auditing mechanism for network storage services. IEEE Trans Emerg Top Comput 9(3):1421–1432. https://doi.org/10.1109/TETC.2020.3005610
22. Xu Y, Yan X, Wu Y, Hu Y, Liang W, Zhang J (2021) Hierarchical bidirectional rnn for safety-enhanced b5g heterogeneous networks. IEEE Trans Netw Sci Eng 8(4):2946–2957. https://doi.org/10.1109/TNSE.2021.3055762
23. Xu Y, Zeng Q, Wang G, Zhang C, Ren J (2020b) An efficient privacy-enhanced attribute-based access control mechanism. Concurr Comput Pract Experience 32(5):e5556. https://doi.org/10.1002/cpe.5556
24. Xu Y, Liu Z, Zhang C, Ren J, Zhang Y, Shen X (2022) Blockchain-based trustworthy energy dispatching approach for high renewable energy penetrated power systems. IEEE Internet Things J 9(12):10036–10047. https://doi.org/10.1109/JIOT.2021.3117924
25. Sarker IH (2021) Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. SN Comput Sci 2(6):1–20
26. Kantardzic M (2011) Data mining: concepts, models, methods and algorithms. Wiley, Hoboken
27. Han J, Kamber M (2012) Data mining: Concepts and techniques. Elsevier
28. Han J, Pei J, Kamber M (2011) Data mining: concepts and techniques. Elsevier
29. Morariu D, Creţulescu R, Breazu M (2017) The weka multilayer perceptron classifier. International Journal of Advanced Statistics and IT &C for Economics and Life Sciences 7(1)
30. More data mining with weka (5.2: Multilayer perceptrons) - youtube. https://www.youtube.com/watch?v=mo2dqHbLpQo. Accessed 28 Mar 2022
31. Ferenc R, Tóth Z, Ladányi G, Siket I, Gyimóthy T (2020) A public unified bug dataset for java and its assessment regarding metrics and bug prediction. Softw Qual J 28(4):1447–1506
32. Cetiner M, Sahingoz OK (2020) A comparative analysis for machine learning based software defect prediction systems. In: 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, pp 1-7
33. Sp RM, Bhattacharya S, Maddikunta PKR, Somayaji SRK, Lakshmanna K, Kaluri R, Hussien A, Gadekallu TR (2020) Load balancing of energy cloud using wind driven and firefly algorithms in internet of everything. J Parallel Distrib Comput 142:16–26
34. Rajput DS, Basha SM, Xin Q, Gadekallu TR, Kaluri R, Lakshmanna K, Maddikunta PKR (2022) Providing diagnosis on diabetes using cloud computing environment to the people living in rural areas of india. Journal of Ambient Intelligence and Humanized Computing 13(5):2829–2840
35. Rupa C, Srivastava G, Gadekallu TR, Maddikunta PKR, Bhattacharya S (2020) A blockchain based cloud integrated iot architecture using a hybrid design. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing. Springer, pp 550-559

Publisher's Note