Wireless Personal Communications (2023) 132:1975–1998
https://doi.org/10.1007/s11277-023-10694-9

Automatic Software Bug Prediction Using Adaptive Artificial Jelly Optimization With Long Short-Term Memory

R. Siva[1] · Kaliraj S[2] · B. Hariharan[1] · N. Premkumar[3]

Accepted: 22 July 2023 / Published online: 23 August 2023
© The Author(s) 2023

Abstract

In the software maintenance and development process, software bug detection is an essential problem because it is related to complete software success. It is recommended to begin anticipating defects at the early stages of creation rather than during the assessment process due to the high expense of fixing the found bugs. The early stage software bug detection is used to enhance software efficiency, reliability, and software quality. Nevertheless, creating a reliable bug-forecasting system is a difficult challenge. Therefore, in this paper, an efficient, software bug forecast is developed. The presented technique consists of three stages namely, pre-processing, feature selection, and bug prediction. At first, the input datasets are pre-processed to eliminate the identical data from the dataset. After the pre-processing, the important features are selected using an adaptive artificial jelly optimization algorithm (A^2JO) to eliminate the possibility of overfitting and reduce the complexity. Finally, the selected features are given to the long short-term memory (LSTM) classifier to predict whether the given data is defective or non-defective. In this paper, investigations are shown on visibly obtainable bug prediction datasets namely, promise and NASA which is a repository for most open-source software. The efficiency of the presented approach is discussed based on various metrics namely, accuracy, F- measure, G-measure, and Matthews Correlation Coefficient (MCC). The experimental result shows our proposed method achieved the extreme accuracy of 93.41% for the Promise dataset and 92.8% for the NASA dataset.

Keywords Artificial jelly optimization algorithm · Long Short-Term Memory · Software bug detection · Reliability · Software quality · And feature selection

✉ Kaliraj S
kaliraj.s@manipal.edu

[1] Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology, Kattankulathur, Tamilnadu, India

[2] Department of Information and Communication Technology, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India

[3] Department of Information Technology, Kongunadu College of Engineering and Technology, Thottiam, Tamilnadu, India

1 Introduction

The effect of software applications is expanding day by day. Reliability labor assessment is becoming increasingly important in both academia and business. A crucial test for every researcher and software professional is to boost software quality with limited testing resources while the length and overall cost of software testing keep rising. The primary goal of software bug prediction (SBP) techniques is to prioritize defective and non-faulty software modules. The engineer will then provide practical test resources and offer testing choices for various software modules to improve software quality.

The prevalence of software bugs has a significant impression on the dependability, performance, and operating costs of software. It takes a lot of effort to create bug-free software, even when software is used carefully since hidden defects are frequently present [1]. A significant difficulty in software engineering is creating a model for software bug prediction that can identify broken modules early on [2]. Predicting software bugs is a crucial step in the creation of software [3]. This is due to the reality that identifying problematic modules before the software is deployed enhances user happiness and overall software effectiveness [4]. Additionally, early software problem prediction enhances software adaption to various settings and boosts resource usage. In the initial phases of the software development life cycle, several software measures, such as class level, method level, file level, and process level, are utilized to identify software flaws without testing the software [5, 6]. Finding software bugs can be done using a variety of techniques, including statistical analysis, machine learning, expert systems, etc.

The software involves numerous flaws that are transmitted to the user, and this causes issues with system efficiency [2]. Therefore, a faster-computerized approach that can forecast approximations of system problems is required. Here, a neural network-based machine learning technique has been applied [7]. This gave an approximation of the outcome that was close to the real outcome already presented. By examining software measurements, it employs machine learning techniques to make predictions about when the software includes flaws, assisting software engineers in raising the caliber of their products [5, 8]. In general, classifier quality can be enhanced by using data preparation [9]. Software detection is typically a classification challenge and the effectiveness of the estimate is dependent on the information from the software metrics and the classifier's use [10]. There is currently research being done on different classifiers and data pre-processing techniques to increase the accuracy of identification models.

The technique of attempting to estimate bugs based on past data is known as bug prediction. Software flaws can have an impact on the product's dependability, quality, and maintenance costs [11, 12]. Countless undetected defects can lead to software failure in the future when designing software. Software maintenance costs between 40 and 60 percent of the total cost, hence it is very important to anticipate errors in the early phases of software development [13]. By foreseeing bugs, it is simple to lower the failure rate of software [50]. By examining software measurements, it employs machine learning techniques to make predictions about whether the software includes flaws, assisting software engineers in raising the quality of their products [15]. In recent years, many techniques have been analyzed including Support Vector Machine (SVM), Neural Network (NN), Naïve Bayes (NB), and k-nearest neighbor (K-NN).

Unfortunately, software defect prediction generally remains a confusing issue. Imperfect expectation selections and benchmarking results using AI classifiers have shown that no major presentation anomalies can be recognized [46] and that there are no specific

classifiers that perform best on every dataset. Large-scale software architecture requires an accurate defect prediction model. Two well-known areas of information excellence that can influence the organization process are class inequity and noise characteristics of information collections [47]. It has an imbalance with many faulty blocks that do not conflict with non-defective ones. Inconsistency can induce a non-practical model in software defect prediction, as most examples are expected to be defect-free [18]. Deriving from unbalanced datasets is problematic. Impaired data related to minority groups prevents a clear understanding of the inherent design of the dataset [19]. Because the dataset has noisy qualities [20] the implementation of software defect prediction is completely reduced [21].

When a machine learning task involves learning from high-dimensional and noisy attribute datasets, attribute selection is frequently used. Near-optimal configurations are challenging to obtain because the majority of feature selection computations perform a neighborhood search during the whole interaction. Metaheuristic optimization finds an answer in the entire search space and exploits the global search capability, fundamentally expanding the capability to find high-quality solutions within a reasonable time [22]. Some of the optimization algorithms used for feature selection are particle swarm optimization (PSO), genetic algorithm (GA), cuckoo search algorithm, ant colony optimization (ACO), etc.

In the current work, we suggest the A^2JO algorithm to increase the predictability of software defects. To create efficient defect prediction models, it is essential to choose the best features that may expose the intrinsic structures of the defect data. The suggested model's primary contribution is given below,

- To detect the software bugs, the adaptive artificial jelly optimization (A^2JO) algorithm and LSTM is used.
- The proposed A^2JO is a combination of the traditional AJO algorithm and chaotic opposition-based learning (COBL). This COBL strategy is used to increase the searching ability and convergence speed.
- To select the optimal features, the suggested method utilizes the artificial jelly optimization algorithm.
- LSTM is proposed here to detect the bug in the software.
- The experiment was performed on 15 different Promise repository datasets. To calculate the proposed performance, different metrics are evaluated.

The construction of the paper is prepared as tracks. In the next section, we will discuss the literature survey, architectural design, and algorithm of AJO and LSTM-based prediction is discussed in Sect. 3, the result and discussion are explained in Sect. 4, and the conclusion part is presented in Sect. 5.

2 Literature Review

Many of the researchers had software bug detection using artificial intelligence techniques. Among them few of the works are listed below; Lopes et al. [23] analyzed more than 4000 fault complaints gathered across three open-source database systems that were mechanically categorized using the Orthogonal Defect Classification (ODC) system. They were achieved under-sampling to evade unbalanced datasets. Experimental results reveal difficulties in automatically classifying some ODC attributes using only reports. Similarly,

Thung et al. [24] performed semi-supervised learning-based automated ODC defect-type classification. Here, they classified 500 bug reports collected from three software systems. The classification accuracy will affect if large datasets are used. Tan et al. [25] developed bug classification based on three components namely, impact, dimensions root cause, and affected component. For the classification process, they used machine learning techniques. Using machine learning techniques they automatically detect 109,014 bugs. Li et al. [26] introduced a machine-learning algorithm to analyze bug features in open-source software. Similar to Tan et al., they introduced to classification of a bug based on concurrency Memory, and Semantic bugs. In [27], Ray et al. analyzed the programming study and cipher excellence of open-source projects. To achieve this objective, they introduced machine learning classifiers. Ni et al. [28] predicted root cause categories from coding based on abstract syntax trees (ASTs) and tree-based (TBCNN). They illustrious six major origin reason classes and 21 subcategories.

Goseva et al. [29] analyzed security and non-security-based errors using supervised and unsupervised learning algorithms. Wu et al. [30] predicted high-impact errors based on active learning with machine learning techniques. Xia et al. [31] presented a machine learning algorithm and Fecher selection technique for predicting Mandelbucks and Borbucks. Later, Du et al. [32] developed a system for cross-project domain adaptation serving the same function. Also, [33] clearly explains error detection and offers a good impression of papers on classifying and prioritizing errors.

In 2018, Hammouri et al., [34] presented a software bug prediction approach depending on machine learning (ML) algorithms. Three monitored ML techniques were used to predict possible software issues depending on historical information. The evaluation approach showed that ML algorithms can be used correctly and effectively. Empirical outcomes demonstrated that the ML technique outperforms other techniques, such as linear AR and POWM models, in terms of effectiveness for the estimation method. Wang, et al. [35] analyzed software bug prediction in terms of creating, modifying, and assessing bug forecasting models in real-world continuous software evolution settings. ConBuild rethinks the selection of training data for models by employing the differential properties of bug prediction data. ConEA redefines effort-aware assessment in continual software development by leveraging the growth of file-bug probability. Investigations of six large-scale open-source software systems' 120 regularly released versions demonstrate the usefulness of methods.

Khan et al. [36] analyzed Artificial Immune Networks (AIN) and machine learning classifiers based on software bug detection. To increase the reliability of the bug prediction process, the hyperparameters were optimally selected. Gupta and Saxena [37] analyzed a model for an object-oriented software bug prediction system (SBPS). Through the Promise Software Engineering Repository, a few open-source projects with problem datasets of a comparable nature were gathered for this investigation. Among all classifiers, the Logistic Regression Classifier has the best accuracy.

In [38], Moustafa et al. analyzed software bug fault identification techniques that use the collective sorting method. The methods were evaluated on datasets of various sizes and applied to utilize various groups of software measurements as features of the sorting algorithms. According to the findings, update measurements performed better than static code measurements and a technique that combines equal parts of data. Qu and Yin, [39] developed by using and expanding node2defect, a bug detection framework that concatenates integrated vectors using conventional software engineering measurements, and assesses network embedding techniques in bug detection. Seven connectivity embedding techniques, two effort-aware models, and 13 open-source Java systems were used in the experiments.

The use of deep learning techniques in software development studies, such as the forecasting of defects and vulnerabilities and the localization of faults, is well-explained in a recent survey [40]. More than 5,400 sentences from publishing articles were manually categorized by Huang et al. [41] into seven categories, such as "Information Delivery" and "Problem Discovery." To forecast these objectives, they subsequently developed a deep neural network. Mahajan and Chaudhary [42] developed software bug localization. To achieve this objective, a hybrid optimization-based CNN was developed. They introduced hybridized cuckoo search-based sea lion optimization algorithm for feature selection. The method attained good results compared to other methods. Rani et al. [43] introduced deep reinforcement learning technique-based bug detection in video games. Wang et al. [44] developed a graph CNN-based software version-to-version bug prediction system. Choetkiertikul et al. [45] had analysed deep learnig algorithm based bug prediction. Cynthia, et al. [46] developed software bug detection based on Feature transformation. Here, they mainly focused on feature selection-based prediction. Moreover, Giray, et al. [47] developed a deep learning algorithm-based bug prediction. Here, they analyzed different machine learning algorithms and deep learning algorithm performance.

When analyzing the literature survey, many of the researchers focused on machine learning algorithm-based prediction and deep learning techniques. In this, most of the researchers were not focused on optimal features; they directly process all the features. This will increase the computation complexity and time consumption. To avoid the issues, in this paper, feature selection-based software bug prediction is proposed.

3 Proposed System Model

The primary aim of the presented approach is to predict the bug in the software. To achieve this objective, an LSTM classifier and adaptive artificial jelly optimization algorithm are used. In this paper, firstly, the software coding is composed through the dataset, and collected datasets are pre-processed. After the pre-processing, the important features are selected using the AJO algorithm. Then, the selected variables are specified to the LSTM classifier to categorize whether the software has a bug or not. The overall structure of the presented methodology is shown in Fig. 1.

3.1 Preprocessing

The software elements in the real-world dataset have identical class labels and software measurements. Machine learning suffers as a result of these recurrent occurrences. Additionally, they hinder the effectiveness of the simulation and lengthen the learning algorithm. To overcome those problems, the duplicate data instances are removed from the software model in preprocessing steps. Once the duplicate data is removed, the subsequent output is served to the feature selection development.

3.2 Feature Selection Using Adaptive Artificial Jelly Optimization

After the pre-processing, the important features are selected from the dataset. For feature selection, in this paper A^2JO algorithm is utilized. The behavior of jellyfish in the ocean served as the inspiration for the probabilistic algorithms known as AJO [48]. The initial spark for examining jellyfish behavior is whether they are traveling as a swarm or into the

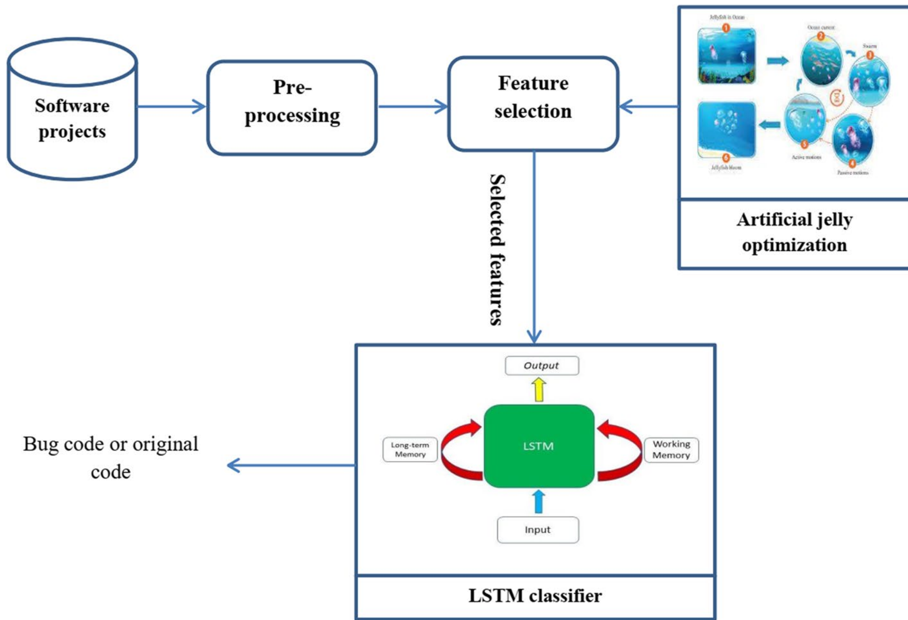

Fig. 1 Semantic diagram of the overall proposed model

ocean current (performing either active or passive movement). The behavior of AJO is given in Fig. 2. Three rules are at the basis of optimization:

- System for controlling the time that determines when jellyfish are in the swarm or within the swarm in the sea current.
- Increased jellyfish migration in the direction of the nutrition source.
- The amount of material used to select the location and its ultimate purpose.

In this paper, we add a quasi opposition-based learning strategy with artificial jelly optimization to increase the searching ability and convergence speed. The step-by-step process of feature selection is explained below;

Step 1: Initialization: The optimization algorithm works based on the initial solution. At first, the initial solutions are generated randomly. The solution consists of only features. The random population of AJO is formulated as follows;

$$W_i = \{w_1, w_2, \ldots\ldots, w_U\} \tag{1}$$

A representation of the solution (w_l) is given in Fig. 3.

In this initialization process, the 0 value can be formulated as the feature not selected and 1 can be represented as the corresponding feature is selected. The main aim of the variable collection process is to reduce the number of variables by improving the efficiency in the sorting algorithm like accuracy and reducing complexity.

Step 2: Create quasi-oppositional solution: To improve the searchability, a quasi-opposite solution is constructed after the solution initialization. This approach is utilized to speed up AJO convergence while also decreasing computing time.

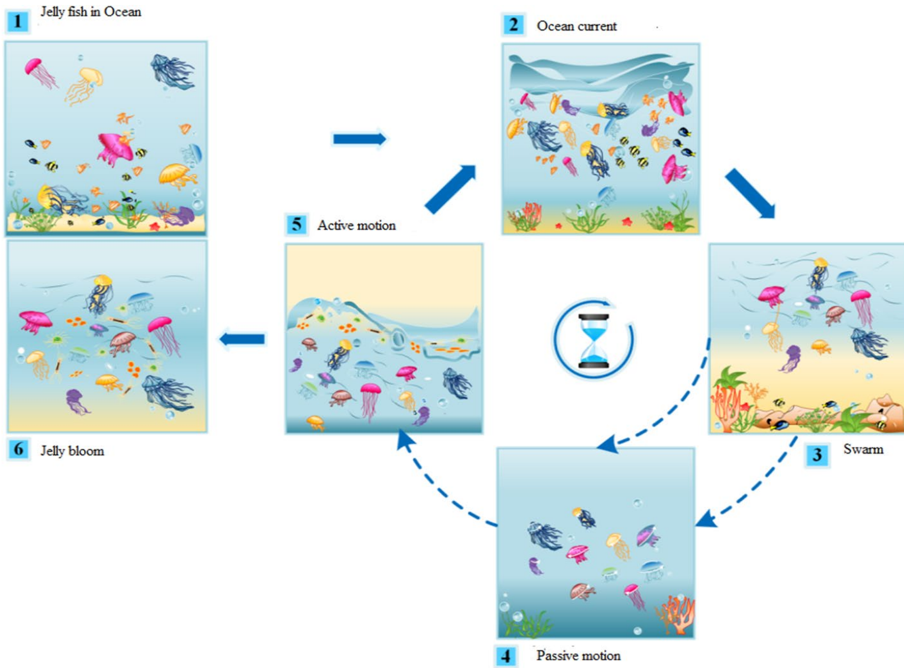

Fig. 2 Behaviors of artificial jelly in Ocean

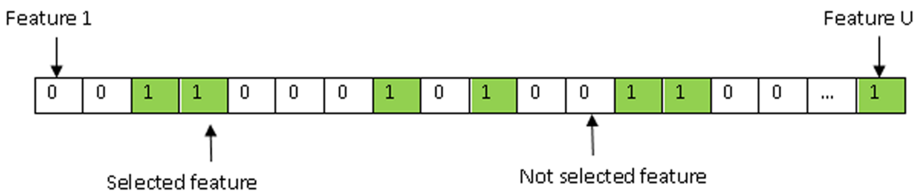

Fig. 3 Representation of solution initialization

For any arbitrary solution $W \in [u, v]$, its opposite solution W_0 can be written as;

$$W_0 = u + v - W \tag{2}$$

The following can be expressed as the multi-dimensional search space (d-dimensions);

$$W_0^i = u^i + v^i - W^i \; ; \; i = 1, 2, ..., d \tag{3}$$

For any arbitrary solution $W \in [u, v]$, its quasi-opposite solution Sq_0 can be written as;

$$W_{q0} = rand\left(\frac{u + v}{2}, W_0\right) \tag{4}$$

It is possible to write the following for the multi-dimensional search space (d-dimensions):

$$W_{q0}^{1} = rand\left(\frac{u^{i}+v^{i}}{2}, W_{0}^{i}\right) \quad (5)$$

Step 3: Fitness calculation: Fitness is determined for each initialized solution to discover the best result. The fitness role is characterized as the maximum value of accuracy, and it is given below,

$$F = max(accuracy) \quad (6)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (7)$$

where TP represents the true positive value, TN denotes the true negative value, FP represents the false positive value and FN denotes the false negative value.

Step 4: Sea current: Because the sea is so rich in resources, jellyfish are attracted to it. A regular of entirely the paths connecting each jellyfish in marine to the jellyfish that is now in the best location is used to determine the sea current's path (P), which is given in Eq. (8).

$$P = \frac{1}{n}\sum P_{i} = \frac{1}{n}\sum\left(W^{*} - ecW_{i}\right) = X^{*} - ec\frac{\sum W_{i}}{n} = W^{*} - ec\sigma \quad (8)$$

$$Setdf = e_{c}\sigma \quad (9)$$

$$P = W^{*} - df \quad (10)$$

where n represents the population, W^{*} denotes the finest site, ec represents the desirability, and σ denotes the mean of all jellyfish. df denotes alteration and the mean location of all jellyfish.

In a region of everything, the average position comprises a specified chance of every jellyfish based on the assumption since jellyfish have a regular geographical extent in all perspectives, where β is the distribution's standard deviation.

Consequently,

$$df = \mu \times \beta \times rand\ (0, 1) \quad (11)$$

$$Set\beta = rand\ \alpha(0, 1) \times \sigma \quad (12)$$

Hence,

$$df = \beta \times rand\,f(0, 1)\mu \quad (13)$$

Here, $e_{c} = \beta \times rand\ (0, 1)$

$$P = X^{*} - \beta \times rand\ (0, 1) \times \mu \quad (14)$$

The new location is as follows,

$$W_{i}(t + 1) = W_{i}(t) + rand\ (0, 1) \times P \quad (15)$$

Is given by,

$$W_i(t+1) = W_i(t) + rand(0,1) \times (W^* - \beta \times rand(0,1)) \times \mu \tag{16}$$

$\beta > 0$ is relative to the distance connecting two locations' distribution coefficient (P). according to the outcomes of quantitative experiments utilizing a sensitivity examination: $\beta = 3$ is attained.

Step 5: Jellyfish swarm: Either passive (type A) or aggressive (type B) motions are used by jellyfish to move in swarms. The swarm takes shape at first, and almost all jellyfish transfer in a type A signal. They eventually start to exhibit type B motions. The positions are refunded by Type A motion, as well as the subsequently updated locations, are provided by Type B motion:

$$W_i(t+1) = W_i(t) + \gamma \times rand(0,1) \times (U_b - L_b) \tag{17}$$

where $U_b and L_b$ are the upper duty-bound and lower duty-bound. $\gamma > 0$ is a motion parameter, which is used to quantify movement around jellyfish location. The results of quantitative analyses research are $\gamma = 0.1$

A jellyfish (j) is chosen at arbitrary, and a vector of jellyfish of interest (i) is chosen, simulating type B movement. When there are more foods accessible in the chosen jellyfish's position (j) than there are in the jellyfish of interest's location (i) the latter moves closer to the former; if there are fewer foods obtainable in the chosen jellyfish's position (j) than there are in the interest's location I the latter swims away immediately (i).

Since both (20) and (21) imitate a route of circulating, each jellyfish in a cluster follows the best route to find food and updates its location. This modification is perceived as financial misuse of the local search region.:

$$S = W_i(t+1) - W_i(t) \tag{18}$$

where,

$$S = rand(0,1) \times D \tag{19}$$

$$D = \begin{cases} W_j(t) - W_i(t) iff(W_i) \geq f(W_i) \\ W_i(t) - W_j(t) iff(W_i) \leq f(W_i) \end{cases} \tag{20}$$

where, f is an objective function of site W.

Hence,

$$W_i(t+1) = W_i(t) + S \tag{21}$$

Period administration is utilized to coordinate elaborate movements throughout time. It controls the distribution in the sea present in addition to the motions of type A and type B in the swarm. The next sections go into further information about the time management strategy.

Step 6: Time control mechanism: The type of motion across time is examined using the time control technique. If jellyfish are pointed in the direction of an ocean current, it is useful to control active and passive motion and look at how they move. The definition of the period regulator function is an arbitrary value that oscillates between 0 and 1. Constant C_0, the mean value between 0 and 1, is present and has a value of 0.5. The given equation is used to calculate the random value of the time control function, which ranges from 0 to 1,

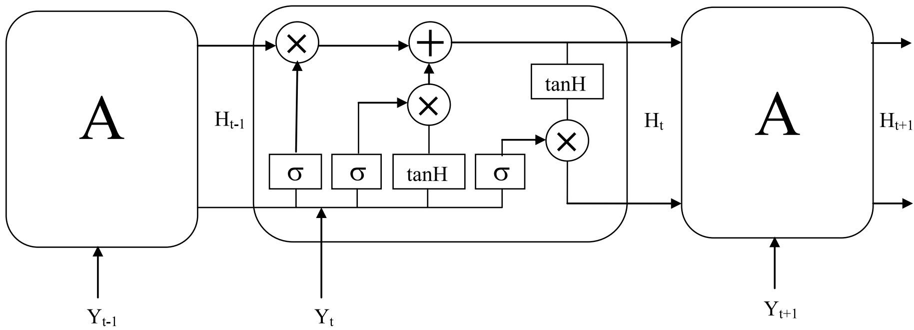

Fig. 4 Structure of LSTM

$$c(t) = \left| \left(1 - \frac{t}{Max_{iter}} \right) W(2xrand(0,1) - 1) \right| \tag{22}$$

where t represents time, Max_{iter} represents the maximum iterations.

Step 7: Termination Criteria: The solution will be updated using the two operators until the optimal solution or weight parameters are found. If the desired result is obtained, the algorithm will be terminated.

3.3 Bug Detection Using LSTM

Here, the proposed approach uses the RNN based on LSTM [49] for error prediction. RNNs are a class of artificial neural networks that may interact with the organization of contributions to in-depth learning and maintain their status when dealing with the following information sources. The LSTM network is a type of intermediate neural system. LSTM contains four neural systems that interface in an optimal method. LSTM can enhance or erase data to the recollection cell state, using an exceptionally planned architecture labeled "Gateway". This is the area where the gateway function information is selected, ie elements of information. It consists of the layering and multiplication work of the sigmoid neural structure. The sigmoid layer reverses the information of the features by the sigmoid capability and evaluates the results somewhere between 0 and 1, depicting information elements that can be experienced in section A of the system. "0" designates that no information is allowed to be sent. "1" indicates that all information is allowed to be sent. At each successor list level, the gate structure in the LSTM is associated with an information gateway, a forgetting gate, and an output gate. The structure of LSTM is given in Fig. 4.

3.4 Forget Gate

The forget gate would decide which details about recent recollection to retain or reject:

$$F_G = \sigma\left[w^F\left(F_t, Y_{t-1}\right) + c^F\right] \tag{23}$$

where, F_G can be described as the forget gate. C and w indicate the control and weight boundaries. F_t addresses input at the existing timestamp; Y_t-1 indicates the result got at the timestamp t-1 since the past square of LSTM. σ indicates the calculated sigmoid capacity

and they give the resulting esteem between 0 and 1. On the off chance that the result is '0' then it implies obstructing entryways. In case the result is '1' entryways let everything pass.

3.5 Input Gate

The input gate I_G chooses the information that should be stored:

$$I_G = \sigma\left[w^I\left(F_t, Y_{t-1}\right) + C^I\right] \tag{24}$$

3.6 Output Gate

Last but not least, the output gate chooses which portion of the storage will eventually provide results:

$$O_G = \sigma\left[w^O\left(F_t, Y_{t-1}\right) + C^O\right] \tag{25}$$

Another candidate memory call M_t is made by a tanH layer and is denoted as,

$$M_t = \tan H\left[w^M\left(F_t, Y_{t-1}\right) + c^M\right] \tag{26}$$

where, tanH allows LSTM to add or eliminate data from the last input. The information gateway selects the age of the incoming memory cell, and the forget gate chooses whether to hold or delete data to generate the last memory.

$$M_t = F_G * M_{t-1} + I_G * M_t \tag{27}$$

where Mt represents the memory cell state at the time (t) and *denotes the element-wise multiplication. Lastly, the output is assessed by,

$$Y_t = O_G * \tan H\left(M_t\right) \tag{28}$$

where,* denotes the element-wise multiplication, Y_t points to the output from the current block. M_t represents the memory cell state. Finally, by using the MSE as the mistake computation, the loss function of the system is assessed,

$$Loss = \sum_{t=1}^{N} \left(Y_t - T_t\right)^2 \tag{29}$$

where *Tt* denotes the desired output. N is the sample of n data points. If the rated score is 0, at that point, the component is considered a bug and if the rated score is 1, at that point, the included information is considered a bug. In its light, the proposed technique distinguishes the bug in the software.

4 Experimentation and Analysis

An experimental result obtained from the proposed software bug predictions is analyzed in this section. For analysis two types of datasets are used namely, NASA and promise datasets.

4.1 Experimental Setup

The execution is done in the python. The execution used system has a 2 GHz dual-core computer with 4 GB RAM running a 64-bit version of Windows 2007.

4.2 Experimental Evaluation Metrics

For experimental analysis, we used the six evaluation metrics namely, Accuracy, F-measure, G-measure, and Matthews Correlation Coefficient (MCC).

Measures	Formula
Accuracy	$\frac{(TN+TP)}{(TN+TP+FN+FP)}$
F-Measure	$2 * \frac{Recall*precision}{Recall+precision}$
Recall	$\frac{TP}{(FN+TP)}$
Precision	$\frac{TP}{(FP+TP)}$
MCC	$\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$
G-Measure	$2 * \frac{Recall*pf}{Recall+(1-pf)}$

4.3 Dataset Description

For experimental analysis, two sets of datasets are used namely, the PROMISE and NASA datasets. In this paper, we analyze 10 real software responsibility schemes since the PROMISE public software engineering repository, which are extremely suggested by numerous investigators in software engineering. Here, 2775 instances are used for experimental analysis. The attribute present in the PROMISE dataset is given in Table 1. Moreover, in this paper, we examine five projects from the NASA dataset. For the NASA dataset, 11,262 instances are utilized for experimental analysis. The attributes present in the NASA dataset are given in Table 2.

4.4 Performance Analysis of Proposed Bug Detection Model

In this section, the suggested technique performance is analyzed. For that, the method considered the ten datasets through the Promise dataset and five datasets from the NASA dataset. The performance result of the proposed model is tabulated in Tables 3 and 4.

Table 3 shows the performance analysis of the suggested model using the promise dataset. Here the proposed method is considered the ten projects for evaluation. From the

Table 1 PROMISE dataset features and selected features

Attributes/features	Description of features	Selected
Wmc	Weighted methods for class	Not selected
Dit	Depth of inheritance tree	Not selected
Noc	Number of children	Selected
Cbo	Coupling between objects	Selected
Rfc	Response for classes	Selected
Lcom	Lack of cohesion of methods	Not selected
Ca	After coupling	Not selected
Ce	Efferent coupling	Not selected
Npm	Number of public methods	Selected
lcom3	Lack of cohesion in methods	Not selected
Loc	Lines of code	Selected
Dam	Data access metric	Selected
Moa	Measure of aggregation	Not selected
Mfa	Measure of functional Anstraction	Not selected
Cam	Cohesion among methods of class	Selected
Ic	Inheritance coupling	Not selected
Cbm	Coupling between methods	Not selected
Amc	Average method complexity	Selected
Max_cc	Cyclomatic complexity (Max)	Selected
Avg_cc	Cyclomatic complexity (Avg)	Not selected

Table 2 NASA dataset features and selected features

Attributes/features	Selected
Line count of code	Selected
Count of blank lines	Not selected
Count of code and comments	Not selected
Count of comments	Selected
Line count of executable code	Not selected
Number of operators	Not selected
. Number of operands	Selected
Number of unique operators	Not selected
Number of unique operands	Not selected
Halstead_Length	Selected
Halstead_Volume	Not selected
Halstead_Level	Not selected
Halstead_Difficulty	Selected
Halstead_Content	Selected
Halstead_Effort	Not selected
Halstead_Error_Estimate	Selected
Halstead_Programming_Time	Not selected
Cyclomatic_Complexity	Not selected
Design_Complexity	Not selected
Essential_Complexity	Selected

Table 3 The proposed performance of the Promise dataset

Projects	Accuracy	F-measure	G-measure	MCC
Ant 1.6	84.68	0.864	0.795	0.63
Ant 1.7	88.82	0.836	0.89	0.56
Camel 1.4	66.78	0.842	**0.878**	0.517
Camel 1.6	87.66	0.71	0.62	0.482
Jedit 4.3	77.92	0.823	0.863	0.52
Log4j 1.0	81.88	0.835	0.792	0.586
Prop 4	**93.41**	**0.883**	0.852	**0.68**
Xalan 2.4	67.84	0.79	0.712	0.412
Xalan 2.5	76.87	0.674	0.72	0.455
Xerces 1.2	75.86	0.559	0.652	0.382

The [Bold] values indicate the highest value for each metric among the corresponding projects. Specifically, for each metric (Accuracy, F-measure, G-measure, MCC), the[Bold] value represents the project with the best performance in that metric. This emphasis aims to facilitate the identification of the most outstanding performance in each metric across the listed projects

Table 4 The proposed performance of the NASA dataset

Projects	Accuracy	F-measure	G-measure	MCC
CM1	79.84	0.886	0.92	**0.58**
JM1	80.68	0.823	0.873	0.479
KC1	85.42	0.889	0.895	0.452
KC2	88.95	0.862	0.889	0.447
PC1	**92.8**	**0.962**	**0.957**	0.42

The [Bold] values indicate the highest value for each metric among the corresponding projects. Specifically, for each metric (Accuracy, F-measure, G-measure, MCC), the[Bold] value represents the project with the best performance in that metric. This emphasis aims to facilitate the identification of the most outstanding performance in each metric across the listed projects

proposed performance analysis result, project prop 4 achieves the maximum accuracy, f-measure, and MCC value is 93.41%, 0.883, and 0.68. Project camel 1.4 achieves the maximum G-measure value is 0.878.

Table 4 shows the performance examination of the suggested technique using the NASA dataset. Here the five projects are considered for proposed evaluation, such as CM1, JM1, KC1, KC2, and PC1. From the proposed performance analysis result, project PC1 achieves the maximum accuracy, f-measure, and G-measure value is 92.8%, 0.962, and 0.957. The project CM1 achieves the maximum MCC value is 0.58. The following section discusses the comparative study of the suggested mechanism and the results are contrasted with other research papers.

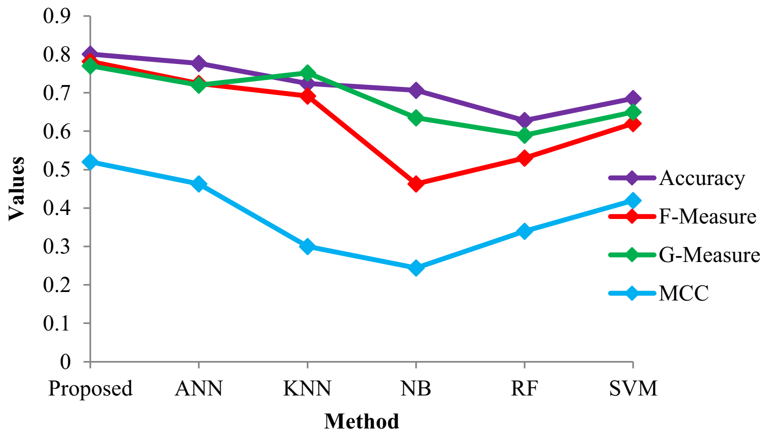

Fig. 5 Average results of bug detection for promise dataset

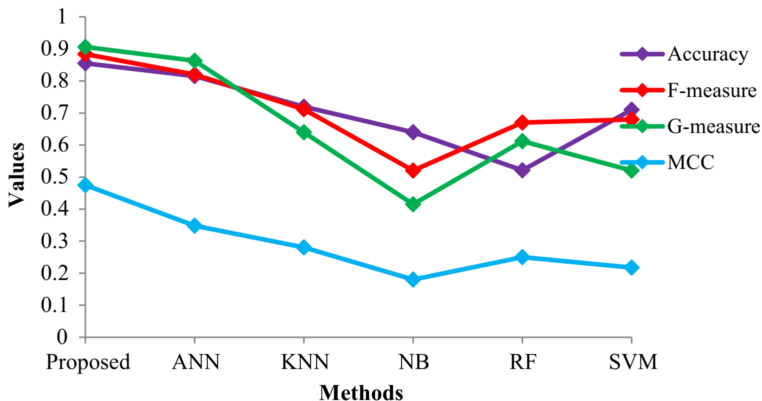

Fig. 6 Average results of bug detection for the NASA dataset

4.5 Comparative Study of Different algorIthms

Here, the experimental results are compared with other software bug prediction models. For comparison, the method considered in the existing bug detection algorithm is ANN, KNN, Naive bias (NB), Random forest (RF), and Support vector machine (SVM). The average results are plotted below,

In Fig. 5 comparison result of bug detection for the promise dataset is analyzed. Here the suggested technique attains an accuracy value is 80.01% but the current technique attains the minimum accuracy rate. The F-measure and G-measure assessment of the suggested approach is 0.7816 and 0.77, which is an extreme assessment when associated with the ANN, KNN, NB, RF, and SVM. The proposed MCC value of the proposed promise bug detection dataset is 0.52, but the existing ANN, KNN, NB, RF, and SVM achieve the MCC value is 0.463, 0.3, 0.244, 0.34, and 0.42. After the results, the suggested technique accomplishes the maximum accuracy, F-measure, G-measure, and MCC value compared to the existing method.

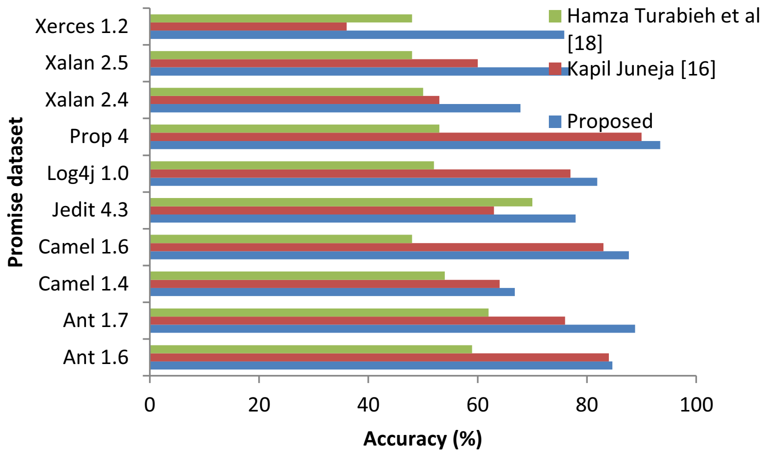

Fig. 7 Accuracy comparison of promise dataset

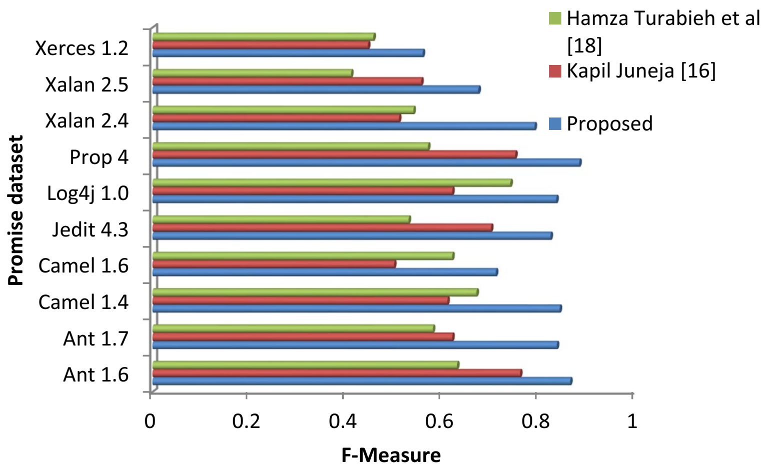

Fig. 8 F-Measure comparison of promise dataset

The comparison result of bug detection for the NASA dataset is shown in Fig. 6. Here the suggested technique achieves an accuracy value is 85.5% nevertheless the existing ANN, KNN, NB, RF, and SVM achieve accuracy value is 81.5%, 72%, 64%, 52.1%, and 71% which is the lowest assessment when compared to the suggested value. The F-measure and G-measure value of the suggested technique is 0.884 and 0.906, which is an extreme assessment when associated with the ANN, KNN, NB, RF, and SVM. The proposed MCC value of the proposed NASA bug detection dataset is 0.47. After the results, the technique achieves the maximum accuracy, F-measure, G-measure, and MCC value associated with the existing method.

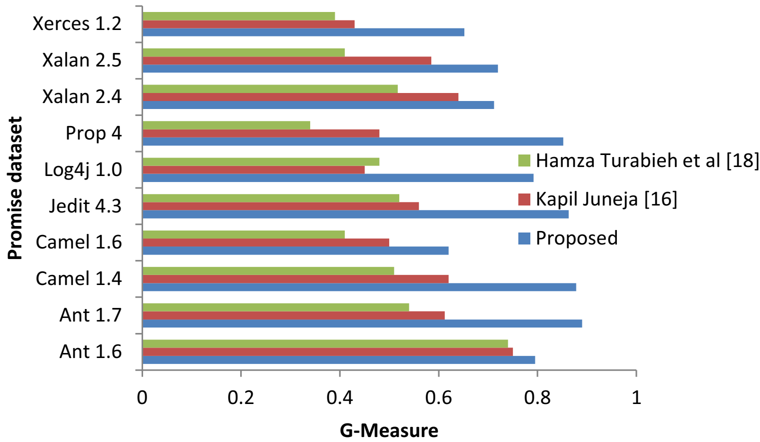

Fig. 9 G-Measure comparison of promise dataset

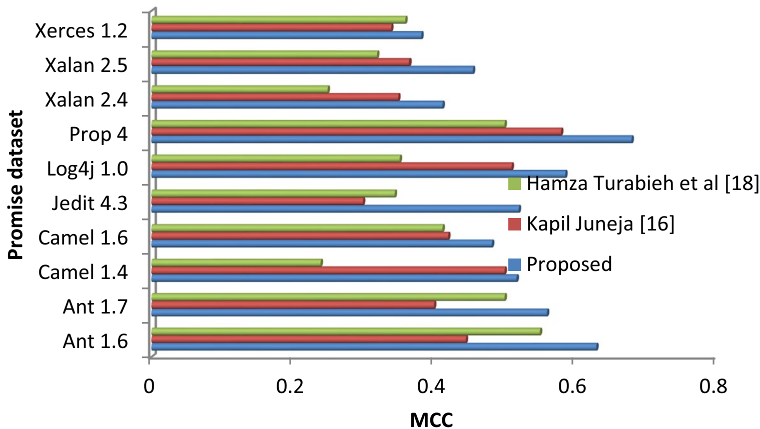

Fig. 10 MCC comparison of promise dataset

4.6 Comparison with Existing Works

To achieve the assessment, the suggested technique considers various existing research papers by Kapil Juneja [14], Hamza Turabieh [16], Zhou Xu [17], and Sushant Kumar Pandey [18]. The existing research paper [14] uses the software bug prediction technique of is a fuzzy-filtered neuro-fuzzy framework. In [16], software bug detection is done by iterated feature collection algorithms with layered RNN, and in [17] kernel PCA and weighted extreme learning machines are used for software bug detection. In [18], bug detection is done by Deep Representation and Ensemble Learning Techniques. The comparison results are box plotted below,

From the above Figs. 7, 8, 9, 10, it shows the comparison result of the promise dataset. The experimental result shows that the technique attains better consequences for altogether ten projects of the promise dataset. The proposed accuracy value of Ant 1.6

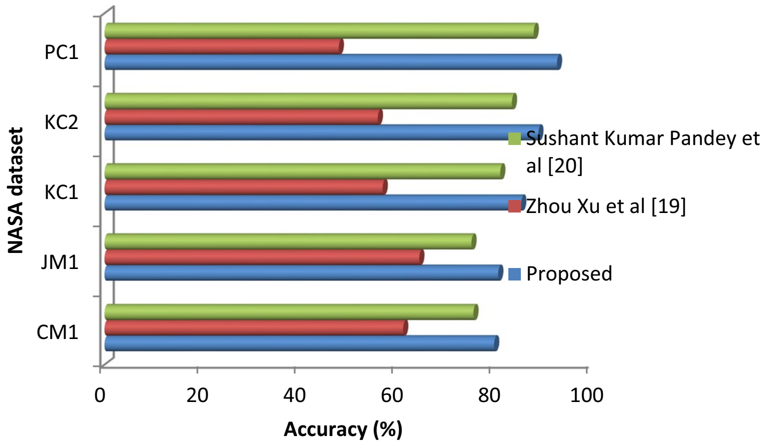

Fig. 11 Accuracy comparison of NASA dataset

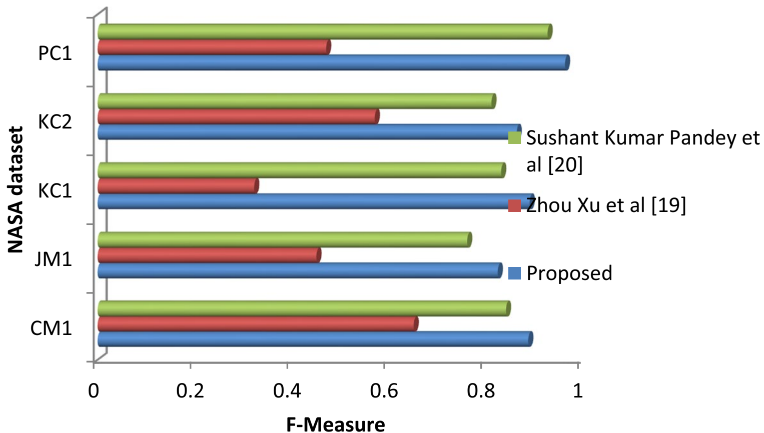

Fig. 12 F-Measure comparison of NASA dataset

is 84.68%, Ant 1.7 is 88.82%, Camel 1.4 is 66.78%, Camel 1.6 is 87.66%, Jedit 4.3 is 77.92%, Log4j 1.0 is 81.88%, Prop 4 is 93.41%, Xalan 2.4 is 67.84%, Xalan 2.5 is 76.87% and Xerces 1.2 is 75.86%. Among these, the Prop 4 project achieves the supreme correctness value compared to all the other projects. From the results, the accuracy of the technique is the supreme value when associated with the existing method [14] and the existing method [16]. The f-measure and g-measure value of the proposed method is the maximum value for all ten projects compared to the existing methods. The proposed MCC value of Ant 1.6 is 0.795, Ant 1.7 is 0.89, Camel 1.4 is 0.878, Camel 1.6 is 0.62, Jedit 4.3 is 0.863, Log4j 1.0 is 0.792, Prop 4 is 0.852, Xalan 2.4 is 0.712, Xalan 2.5 is 0.72 and Xerces 1.2 is 0.652. Among these, the Ant 1.7 project achieves the extreme MCC value associated with all the other projects. When compared, the suggested technique attains improved consequences associated with the methods.

The assessment result of the NASA dataset is exposed in the above Figs. 11, 12, 13, 14. The experimental result shows that the technique attains better consequences for all five

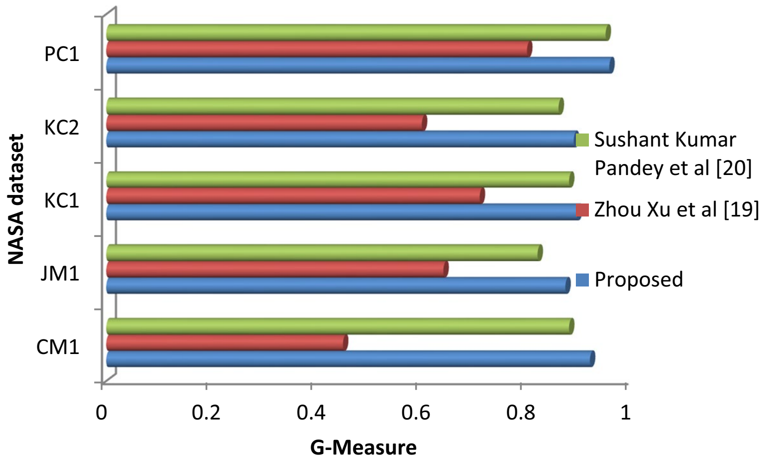

Fig. 13 G-Measure comparison of NASA dataset

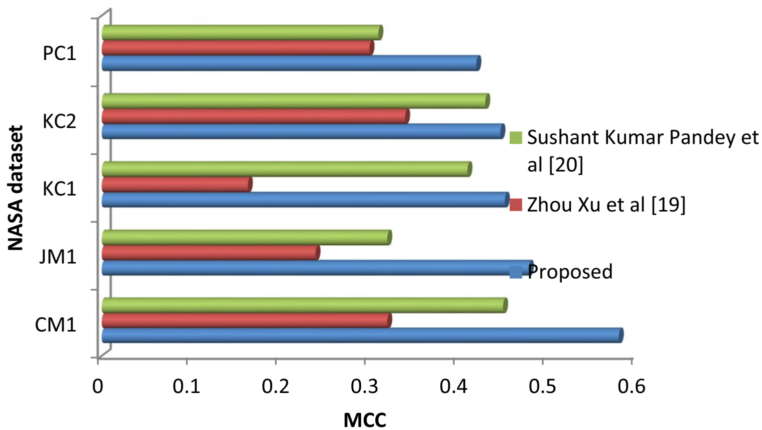

Fig. 14 MCC comparison of the NASA dataset

projects of the NASA dataset associated with the algorithm. The proposed accuracy value of CM1 is 79.84%, JM1 is 80.68%, KC1 is 85.42%, KC2 is 88.95%, and PC1 is 92.8%. Among these, the PC1 project reaches the extreme accuracy worth associated with all the other projects. From the results, the accuracy of the method is of extreme value when associated with the current technique [17] and the existing method [18]. The f-measure and g-measure value of the technique is the maximum value for all five projects. The proposed MCC value of CM1 is 0.58, JM1 is 0.479, KC1 is 0.452, KC2 is 0.447, and PC1 is 0.42. Among these, the CM1 project attains the maximum MCC value compared to all the other projects. When compared, the suggested approach attains healthier consequences associated with the methods.

In Fig. 15, we analyze the performance of the proposed approach based on accuracy measures. For comparison, we used recently published works namely, [18, 26, 27, 44], and [17]. The detailed description of each research is explained in Sect. 2. When analyzing

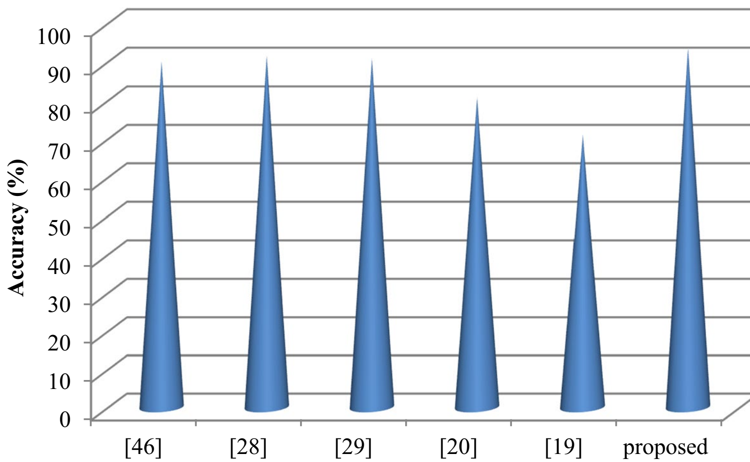

Fig. 15 Comparison with recently published articles

Fig. 15, we obtained the maximum accuracy of 93.41% which is high compared to other research works. This is due to the optimal feature section using the A^2JO algorithm.

5 Conclusion

A novel technique that integrates the metaheuristic optimization method (A^2JO) and deep learning algorithm for software bug prediction has been presented in this paper. For selecting the optimal features A^2JO algorithm has been used and for prediction LSTM-depend, an RNN has been used. The optimal features process leads to enhancing the performance of bug detection. The proposed A^2JO algorithm effectively increases the searching ability and convergence speed. We have chosen five NASA public datasets and ten promise datasets for our experiment. We analyze the suggested method utilizing accuracy, F-measure, G-Measure, and MCC and associated it with state-of-the-art approaches and different classifiers. Subsequently investigation, we create that the evaluation metrics of the approach are higher than the existing state-of-the-art techniques. The proposed method attains the maximum bug detection accuracy for the promise dataset is 93.41% and the detection accuracy for the NASA dataset is 92.8%. The experiment shows that our model is effective for prediction. In the future, we strategy to use a hybrid deep learning technique, which will lead to better results and also solve the problem of the class imbalance problem. We can instrument optimization methods, vectorization, and broadcast approaches for improved and faster consequences. Other deep learning frameworks can also be tried for error prediction. The suggested technique can be practical for many faults, which as software reliability.

Funding Open access funding provided by Manipal Academy of Higher Education, Manipal. The authors declare that we don't have competing interests and funding.

Declarations

Conflict of interest The corresponding author states that there is no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications, 144*, 113085.
2. Abozeed, S.M., ElNainay, M.Y., Fouad, S.A. & Abougabal, M.S. (2020). Software bug prediction employing feature selection and deep learning. In *2019 International Conference on Advances in the Emerging Computing Technologies (AECT)* (pp. 1–6). IEEE.
3. Panda, M. & Azar, A.T. (2021). Hybrid multi-objective grey wolf search optimizer and machine learning approach for software bug prediction. In *Handbook of research on modeling, analysis, and control of complex systems* (pp. 314–337). IGI Global.
4. Kumar, R., & Gupta, D. L. (2016). Software bug prediction system using neural network. *European Journal of Advances in Engineering and Technology, 3*(7), 78–84.
5. Chaubey, P.K., & Arora, T.K. (2020). Software bug prediction and classification by global pooling of different activation of convolution layers. *Materials Today: Proceedings*.
6. Ferenc, R., Gyimesi, P., Gyimesi, G., Tóth, Z., & Gyimóthy, T. (2020). An automatically created novel bug dataset and its validation in bug prediction. *Journal of Systems and Software, 169*, 110691.
7. Aggarwal, A., Dhindsa, K.S., & Suri, P.K. (2021). Enhancing software quality assurance by using knowledge discovery and bug prediction techniques. In *Soft computing for intelligent systems* (pp. 97–118). Springer, Singapore.
8. Kaen, E. & Algarni, A. (2019). Feature selection approach for improving the accuracy of software bug prediction. In *Journal of King Abdulaziz University: Computing and Information Technology Sciences, 8*(1), (pp. 35–44). https://www.kau.edu.sa/Files/320/Researches/72531_45679.pdf
9. Thaher, T. & Khamayseh, F. (2020). A classification model for software bug prediction based on ensemble deep learning approach boosted with SMOTE technique. In *Congress on intelligent systems* (pp. 99–113). Springer, Singapore.
10. Ateya, H. A. B., & Baneamoon, S. M. (2020). Software bug prediction using static analysis with abstract syntax trees. *International Journal of Engineering and Artificial Intelligence, 1*(4), 57–64.
11. Tamanna, O. P. S. (2022). Random permutation-based hybrid feature selection for software bug prediction using bayesian statistical validation. *International Journal of Engineering Trends and Technology, 70*(4), 188–202. https://doi.org/10.14445/22315381/IJETT-V70I4P216.
12. Sangeetha, Y., & Jaya Lakshmi, G. (2021). Prediction of software bugs using machine learning algorithm. In *Advances in Automation, Signal Processing, Instrumentation, and Control* (pp. 2683–2692). Springer, Singapore.
13. Kaur, A., Kaur, K., & Chopra, D. (2017). An empirical study of software entropy based bug prediction using machine learning. *International Journal of System Assurance Engineering and Management, 8*(2), 599–616.
14. Juneja, K. (2019). A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation. *Applied Soft Computing, 77*, 696–713.
15. Sharma, D., & Chandra, P. (2018). Software fault prediction using machine-learning techniques. In *Smart computing and informatics* (pp. 541–549). Springer, Singapore.
16. Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. *International Journal of Advanced Computer Science and Applications, 9*(2), 78–83.
17. Zhou, Xu., Liu, J., Luo, X., Yang, Z., Zhang, Y., Yuan, P., Tang, Y., & Zhang, T. (2019). Software defect prediction based on kernel PCA and weighted extreme learning machine. *Information and Software Technology, 106*, 182–200.

18. Sushant, K. P., Ravi, B. M., & Anil, K. T. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications, 144*, 113085.
19. Wang, T., Li, W., Shi, H., & Liu, Z. (2011). Software defect prediction based on classifiers ensemble. *Journal of Information & Computational Science., 16*(8), 4241–4254.
20. Kim, S., Zhang, H., Wu, R., & Gong, L. (2011). Dealing with noise in defect prediction. *Proceeding of the 33rd International Conference on Software Engineering*, pp 481–490.
21. Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2012). Reflections on the NASA MDP data sets. *IET Software, 6*(6), 549–558.
22. Kabir, M. M., Shahjahan, M., & Murase, K. (2012). A new hybrid ant colony optimization algorithm for feature selection. *Expert Systems with Applications, 39*(3), 3747–3763.
23. Lopes, F., Agnelo, J., Teixeira, C. A., Laranjeiro, N., & Bernardino, J. (2020). Automating orthogonal defect classification using machine learning algorithms. *Future Generation Computer Systems, 102*, 932–947.
24. Thung, F., Le, X.B.D., Lo, D. (2015). Active semi-supervised defect categorization. In: *23rd Int. conference on program comprehension*, pp 60–70.
25. Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., & Zhai, C. (2014). Bug characteristics in open source software. *Empirical Software Engineering, 19*(6), 1665–1705.
26. Zhang, N., Ying, S., Ding, W., Zhu, K., & Zhu, D. (2021). WGNCS: A robust hybrid cross-version defect model via multi-objective optimization and deep enhanced feature representation. *Information Sciences, 570*, 545–576.
27. Ray, B., Posnett, D. Filkov, V., Devanbu, P. (2014). A large scale study of programming languages and code quality in GitHub. In: *ACM SIGSOFT symposium on the foundations of software engineering*, pp 155–65
28. Ni, Z., Li, B., Sun, X., Chen, T., Tang, B., & Shi, X. (2020). Analyzing bug fix for automatic bug cause classification. *Journal of Systems and Software, 163*, 110538.
29. Goseva-Popstojanova, K., Tyo, J. (2018). Identification of security related bug reports via text mining using supervised and unsupervised classification. In: *Int. conf. on software quality, reliability and security*, pp. 344–355.
30. Wu, X., Zheng, W., Chen, X., Zhao, Y., Yu, T., & Mu, D. (2021). Improving high-impact bug report prediction with combination of interactive machine learning and active learning. *Information and Software Technology, 133*, 106530.
31. Xia, X., Lo, D., Wang, X., Zhou, B. (2014). Automatic defect categorization based on fault triggering conditions. In: *Int. conference on engineering of complex computer systems*, pp. 39–48.
32. Du, X., Zhou, Z., Yin, B., & Xiao, G. (2020). Cross-project bug type prediction based on transfer learning. *Software Quality Journal, 28*(1), 39–57.
33. Ahmed, H. A., Bawany, N. Z., & Shamsi, J. A. (2021). Capbug-A framework for automatic bug categorization and prioritization using NLP and machine learning algorithms. *IEEE Access, 9*, 50496–50512.
34. Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. *International Journal of Advanced Computer Science and Applications, 9*(2), 78–83.
35. Wang, S., Wang, J., Nam, J. & Nagappan, N. (2021). Continuous software bug prediction. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 1–12).
36. Khan, F., Kanwal, S., Alamri, S., & Mumtaz, B. (2020). Hyper-parameter optimization of classifiers, using an artificial immune network and its application to software bug prediction. *IEEE Access, 8*, 20954–20964.
37. Gupta, D. L., & Saxena, K. (2017). Software bug prediction using object-oriented metrics. *Sādhanā, 42*(5), 655–669.
38. Moustafa, S., ElNainay, M. Y., El Makky, N., & Abougabal, M. S. (2018). Software bug prediction using weighted majority voting techniques. *Alexandria engineering journal, 57*(4), 2763–2774.
39. Qu, Y., & Yin, H. (2021). Evaluating network embedding techniques' performances in software bug prediction. *Empirical Software Engineering, 26*(4), 1–44.
40. Yang, Y., Xia, X., Lo, D., Grundy, J. (2022). A survey on deep learning for software engineering. *ACM Computing Surveys (CSUR), 54*(10), 1–73.
41. Huang, Q., Xia, X., Lo, D., & Murphy, G. C. (2020). Automating intention mining. *IEEE Transactions on Software Engineering, 46*(10), 1098–1119.
42. Mahajan, G., & Chaudhary, N. (2022). Design and development of novel hybrid optimization-based convolutional neural network for software bug localization. *Soft Computing, 26*(24), 13651–13672.

43. Rani, G., Pandey, U., Wagde, A. A., & Dhaka, V. S. (2022). A deep reinforcement learning technique for bug detection in video games. *International Journal of Information Technology, 15*(1), 355–367.
44. Wang, Z., Tong, W., Li, P., Ye, G., Chen, H., Gong, X., & Tang, Z. (2023). BugPre: an intelligent software version-to-version bug prediction system using graph convolutional neural networks. *Complex & Intelligent Systems, 9*(4), 3835–3855.
45. Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T., Ragkhitwetsagul, C., & Ghose, A. (2021). Automatically recommending components for issue reports using deep learning. *Empirical Software Engineering, 26*(2), 1–39.
46. Cynthia, S.T., Banani, R., & Debajyoti, M. (2022). Feature transformation for improved software bug detection models. In *15th Innovations in Software Engineering Conference*, pp. 1–10
47. Giray, G., Kwabena, E. B., Ömer, K., Önder, B., & Bedir, T. (2023). On the use of deep learning in software defect prediction. *Journal of Systems and Software, 195*, 111537.
48. Xuewu, Z. H. A. O., Hongmei, W. A. N. G., Chaohui, L. I. U., Lingling, L. I., Shukui, B. O., & Junzhong, J. I. (2022). Artificial jellyfish search optimization algorithm for human brain functional parcellation. *Journal of Frontiers of Computer Science & Technology, 16*(8), 1829–1841.
49. Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena, 404*, 132306.
50. Immaculate, S. D., Begam, M. F., & Floramary, M. (2019). Software bug prediction using supervised machine learning algorithms. In *Proc. Int. Conf. Data Sci. Commun. (IconDSC)*, pp. 1–7.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

DR. R. Siva obtained his Bachelor's degree in Computer science and Engineering from Bharathidasan University ,Trichy, Jayaram College of Engineering & Technology, Trichy, in 2000. Then he obtained his Master's degree in Computer Science and Engineering from Manonmaniam Sundaranar University, Tirunelveli, in 2004 .He obtained Ph.D degree in Computer Science and Engineering from Anna University, Chennai, in 2019. Currently, He is an Assistant Professor in the Department of School of Computing at SRM Institute of Science and Technology. He has around 18 years of teaching experience in various technical institutions. He published more journals in national and International Level. His research interests are in Machine Learning, Cloud Computing, Network Security, Image Processing and Data Mining.. He is a member of the ISTE and a member of the CSI.

DR. Kaliraj S is a Senior Assistant Professor in the Department of Information and Communication Technology, MIT Manipal, Manipal Academy of Higher Education (Institution of Eminence), India. He received his B.E, M.E (Distinction) and PhD from Anna University, Chennai, Tamil Nadu, India. He has completed two industry certifications, MCTS (Microsoft Certified Technology Specialist) and the EMC Academic Associate, Data Science and Big Data Analytics. His area of research is Verification of Machine Learning Systems, Fault

Prediction and Localization, Data Science, Machine Learning Applications in Society, NLP and Software Testing. He has published 5 Patents and more than 25 research papers covering all major areas of Software Engineering, Machine Learning, and Data Science in top journals and conferences. He has guided more than 35 students in their master's and undergraduate research. He has served as a Session Chair and member of the Advisory Committee and Technical Committee of Various International Conferences. He has acted as a resource person for the faculty development programs, workshops, Guest Lectures, and conferences organized by various institutions and universities. He is the reviewer of Scopus and WOS-indexed international Journals in his area of research.

DR. B. Hariharan obtained his Bachelor's degree in Information Technology from Anna University Chennai, C.S.I Institute of technology, Thovalai, in 2008. Then he obtained his Master's degree in Computer Science and Engineering from Anna University Chennai, University College of Engineering, Nagercoil, in 2012. He obtained Ph.D degree in Computer Science and Engineering from Anna University, Chennai, in 2020. Currently, He is an Assistant Professor in the Department Computational Intelligence, School of Computing at SRM Institute of Science and Technology. He has around 14 years of teaching experience in various technical institutions. He published more journals in national and International Level. His research interests are in Machine Learning, Cloud Computing, Network Security, Image Processing, IoT and Data Mining. He is a member of the ACM, ISTE and a member of the CSI.

N. Premkumar is an Associate Professor in the Department of Information Technology at Kongunadu College of Engineering and Technology in Tiruchirappalli, Tamilnadu, India. His research interest includes Fog Computing, Cloud Computing, Software Engineering and Wireless Communication.