Computer Systems Science & Engineering
DOI: 10.32604/csse.2023.025991
Article

Tech Science Press

Adaptive Deep Learning Model for Software Bug Detection and Classification

S. Sivapurnima* **and D. Manjula**

Department of Computer Science and Engineering, College of Engineering, Guindy, Anna University, Chennai, 600025, Tamilnadu, India
*Corresponding Author: S. Sivapurnima. Email: sivampur@gmail.com
Received: 12 December 2021; Accepted: 16 February 2022

Abstract: Software is unavoidable in software development and maintenance. In literature, many methods are discussed which fails to achieve efficient software bug detection and classification. In this paper, efficient Adaptive Deep Learning Model (ADLM) is developed for automatic duplicate bug report detection and classification process. The proposed ADLM is a combination of Conditional Random Fields decoding with Long Short-Term Memory (CRF-LSTM) and Dingo Optimizer (DO). In the CRF, the DO can be consumed to choose the efficient weight value in network. The proposed automatic bug report detection is proceeding with three stages like pre-processing, feature extraction in addition bug detection with classification. Initially, the bug report input dataset is gathered from the online source system. In the pre-processing phase, the unwanted information from the input data are removed by using cleaning text, convert data types and null value replacement. The pre-processed data is sent into the feature extraction phase. In the feature extraction phase, the four types of feature extraction method are utilized such as contextual, categorical, temporal and textual. Finally, the features are sent to the proposed ADLM for automatic duplication bug report detection and classification. The proposed methodology is proceeding with two phases such as training and testing phases. Based on the working process, the bugs are detected and classified from the input data. The projected technique is assessed by analyzing performance metrics such as accuracy, precision, Recall, F_Measure and kappa.

Keywords: Software bug detection; classification; pre-processing; feature extraction; deep belief neural network; long short-term memory

1 Introduction

For now, programming support is a time consuming and time-consuming part of software engineering, where diagnosing and dealing with bugs and dealing with improvements are the most basic tasks. Many attempts using programming test techniques like standard and dynamic testing, white in addition dark boxes testing and other testing techniques are aimed at systematic error recognition [1]. Errors reported by end customers, despite programming tests, should be explored in light of the fact that error reports reveal errors not detected during the product testing phase. Besides, they further enhance the client

knowledge of the product in addition update it to new end-customer needs. Programming error emergency structures such as Bugzilla can be utilized to continue the product, particularly when getting error reports from end customers and when the product is slammed [2]. Error reports can be considered, branded, and in addition fingered in emergencies until assigned to designers. There are many issues in the error reporting area, e.g., it is important to focus on error reports, especially some error reports such as security error reports, which can directly affect the final client's weak rates, which should not actually be resolved when it is conceivable. There are several attempts towards a guaranteed guarantee of security error reports [3,4]. Besides, the severity of each error report should be expected, which can be tested at an early stage [5].

The complex relationship of error reporting observed by emergencies can be the location of identical error intelligences that represent up to 70% of intelligences in the box of error emergency structures, particularly aimed at open-source schemes with massive end client networks [6]. There can be two phases to recognizing duplicate error reports, and they address complex issues. The first challenge is the technique of extracting an element that detects highly productive components from error reports [7]. The next test is the copy detection technique, which requires an indicator or classifier model. There are several choices in the two attempts to work on providing the location of duplicate error reports [8]. The first challenge is very fundamental in that separating the most efficient features will help in the best copy discovery; For instance, it cannot likely to distinguish oranges and applies based on their length [9]. Similarly, the volume is louder when it comes to length, width and height. Therefore, the best free highlights can accommodate duplicate authorization. This review center focuses on finding features that are most helpful in further enhancing the location performance of duplicate error reports by incorporating the best of the class components [10].

The conventional testing procedure can be consisting of high time and its fails to detect the bug efficiently. The enhancement of the classification and detection time which decrease the human errors, the Artificial Intelligence (AI) can be consumed in the structure. Different techniques are developed by the experts to identify the software bug detection such as Bayesian Network and Artificial Neural Network (ANN) [11], Support Vector Machine (SVM) [12]. The machine learning techniques are providing efficient outcomes but its not suitable for high storage of datasets. So, recently, deep learning is consumed to identify the software bug detection such as Deep Belief Neural Network (DBNN), Deep Neural Network (DNN) and Convolutional Neural Network (CNN) [13]. In the software bug detection also, feature selection can be utilized to efficient software bug detection. This selection and weighting parameter selection is achieved with the assistance of the optimization algorithm such as Whale Optimization Algorithm (WOA), Grey Wolf Optimization (GWO), Firefly Algorithm (FA) and Particle Swarm Optimization (PSO) [14,15]. This selected the optimization algorithm is affected by the convergence analysis. So, the optimal detection and classification should be designed in the paper.

Main Contribution of the Work

- ADLM is developed for automatic duplicate bug report detection and classification process.
- The proposed ADLM is a combination of CRF-LSTM and DO. In the CRF, the DO can be used towards chose the optimal weight values in network. The proposed automatic bug report detection is proceeding with three stages like pre-processing, feature extraction in addition bug detection with classification. Initially, the bug report dataset is gathered from the online system.
- In the pre-processing phase, the unwanted information from the input data are removed by using cleaning text, convert data types and null value replacement. The pre-processed data is sent into the feature extraction phase.

- ❖ In the feature extraction phase, the four types of feature extraction method are utilized such as contextual, categorical, temporal and textual. Finally, the features are sent to the proposed ADLM for automatic duplication bug report detection and classification.
- ❖ The proposed methodology is proceeding with two stages like training and testing stages. Based on the working process, the bugs are detected and classified from the input data. The proposed methodology is assessed through analyzing performance metrices like accuracy, precision, Recall, F_Measure in addition kappa.
- ❖ The projected technique can be contrasted with the traditional techniques like SVM, ANN, DNN in addition DBNN respectively.

The remaining part of the article is pre-planned as follows; Section 2 provides the detail analysis process of the software bug detection. Section 3 provides the detail description of the proposed method. Section 4 delivers the part clarification of the projected classification process. The results and discussion of the projected technique can be explained in this Section 5. Finally, the summary of the article is given in the Section 5.

2 Related Works

Dissimilar kinds of techniques can be obtainable towards identify and identification of software bug from the software databases. Few of the works are analyzed in this portion.

Pandey et al. [16] have presented SBP using deep depiction and group learning (BPDET) methods. This is matched through ensemble learning (EL) and Deep Representation (DR). Product measurements used for SBP are generally routine. Stacked denominating auto-encoder (SDA) can be utilized aimed at in-depth depiction of programming measurements, that can be a vigorous component learning technique. The proposed model is mostly alienated into two phases: the deep learning level in addition the two layers of the EL level (TEL). Rhmann et al. [17] have introduced composite calculations for software defects based on change measurements. Programming imperfect anticipation emphasizes the utilization of error detection designs for the ID of errors previous to product arrival. The use of error expectation models helps to decrease the cost and effort required to develop programming. Incomplete expectation models utilize actual information achieved from programming programs to produce designs in addition test the design during future arrival of the product.

Malhotra et al. [18] have introduced the perfect framework for error prediction using AI methods with Android programming. The primary objectives of the review were (I) to explore AI strategies using databases derived from well-known open-source programming (ii) to utilize appropriate processing computations to quantify the display of flaw expectation designs (iii) to utilize fact tests for potential correlation. Approval of models on various approaches to AI methods and (iv) information packages. Use object-sorted measurements to predict adequate classes using 18 machine learning practices in this review. Results were approved between 10 overlay and discharge approval techniques. The reliability in addition substance of the results can be assessed utilizing scalable testing in addition post-trial testing. Khan et al. [19] have introduced unique AI methods to program imperfect expectation using seven extensively used databases. Machine leaning algorithm-based performance was presented. Faseeha et al. [20] have introduced a system for programming disability expectation by collecting highlight determination and learning methods. The structure contains of four phases: 1) dataset selection, 2) pre-processing, 3) classification and 4) outcomes. This system can be carried out on clean NASA MDP databases that are generally accessible and the implementation is reflected using a variety of measures: F-scale, accuracy, MCC in addition ROC. First of all, the implementation of all the trial techniques within the design within each database was different and a strategy was identified that was most significant to each other and to

each performance scale. Furthermore, the effects of the projected design with all hunting techniques can be analyzed with the consequences of 10 significant managed classification methods.

Pandey et al. [16] have presented SBP using deep depiction and group learning (BPDET) methods. However, this method is consuming high processing time for identify the bugs from the databases. Rhmann et al. [17] have introduced composite calculations for software defects based on change measurements. Moreover, this technique is fails to achieve efficient accuracy level during identification stage. Malhotra et al. [18] have introduced the perfect framework for error prediction using AI methods with Android programming. Hence, this method consumes huge memory consumption during implementation. Khan et al. [19] have introduced unique However; this method is trapped in local optima problem. Faseeha et al. [20] have introduced a system for programming disability expectation by collecting highlight determination and learning techniques. However, this method provides efficient results for this specified dataset.

3 Proposed Methodology

Generally, the software bug source, various kinds of bugs are presented. Moreover, some bug tracking systems given a software bug classification. Normally, different kinds of software bug detection techniques with different uses. Compared with the conventional study, the bug classification techniques may not be applied to the classification and detection of software bug. Hence, in this paper, ADLM is developed to detection and classification of software bugs. The projected technique can be proceeding with the three stages like pre-processing, feature extraction in addition classification with detection stage. The complete block diagram of the projected technique is presented in Fig. 1.

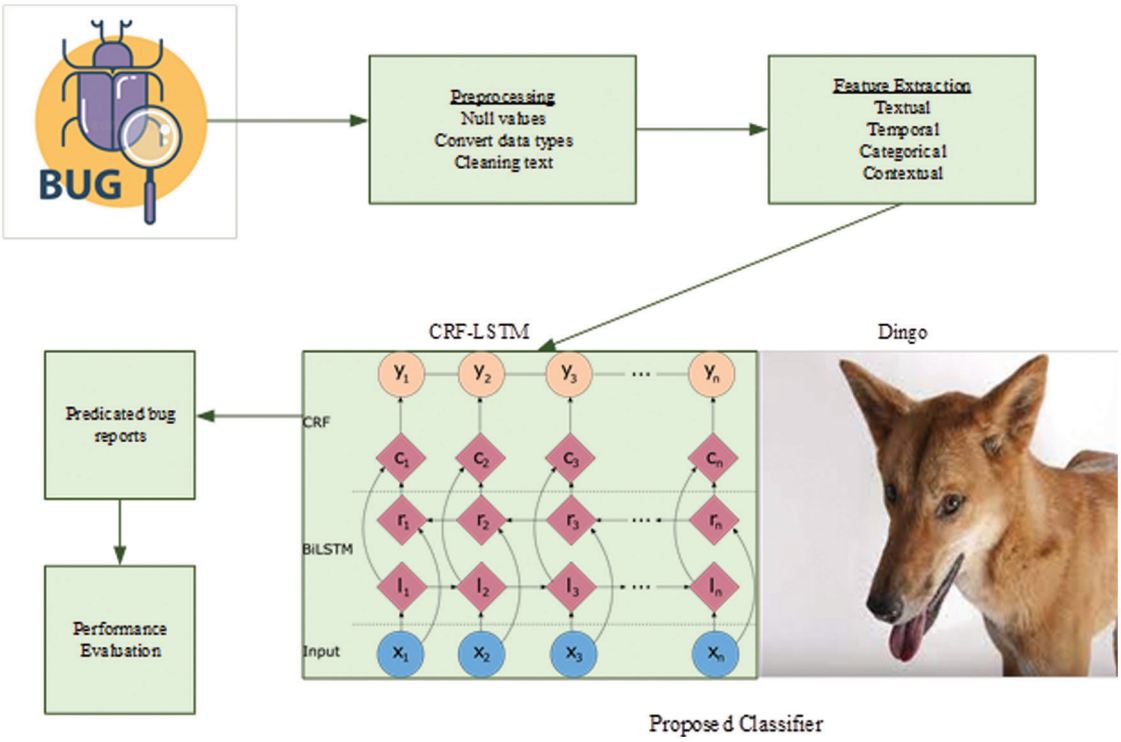

Figure 1: Block diagram of proposed methodology

At initial, the databases are gathered from the online system. After that, the data can be cleaned based on null value removal, convert data types and cleaning text. To identify the bugs from the software data, the feature extraction can be an vital step to extract the features. Different kinds of feature set are available to extracting features from the pre-processed data. To perfect identification of bugs from the software data, optimal feature extraction methods are utilized in the proposed methodology such as textual, temporal, categorical and contextual. Finally, the extracted features can be sent to the classifier aimed at identifying and detecting the bugs from the software data. The proposed classifier is designed by combination of CRF-LSTM and DO optimizer. The CRF-LSTM network may be affected by the training error which solved by utilizing DO optimizer. The detail description of the projected methodology can be explained in the below section.

3.1 Pre-processing

The pre-processing phase can be vital towards removing unwelcome information obtainable in the input databases. The null values are removed from the input report. Additionally, it is changed into numerical parameters. The collected databases are cleaned the text by eliminating useless in addition frequent words, removing redundant words, removing conjunctions, punctuation in addition stemming words.

3.1.1 Stop Word Removal

The stop word removal step is utilized to remove the stop word from the text. At last, the stemming process is returned to the root by computing the suffix and prefix of the word. If the word is related to a set of letters that have an interest in separation.

3.1.2 Stemming

Deleting the prefixes and additions of each word converts the correct inflection types of some words to a similar source. For example, there is a so-called normal root or root segment that meets sets aside and includes all of the parts. Based on the pre-processing stage, the required texts are collected and unwanted features are removed from the input text tweets. Stop word removal is a procedure to remove the stop word from the input database. The synonyms of words are separated with the consideration of pre-processing stage [21].

3.2 Feature Extraction

Features are divided as data fields of bug reports and it describes and every feature is extracted by utilizing unique method. Many different feature extraction methods are utilized to extract essential features such as (1) textual, (2) temporal, (3) Categorical and (4) Contextual.

3.2.1 Textual Feature

Texture can be a set of features that isolate the text fields of software error reports. From these models is the BM25F model, that can be the complete weighted average of the term frequency (TF) in addition the reverse document frequency (IDF) for altogether normal terms in a couple of dubious duplicate statements. The dominant query of this investigate lies in the TF-IDF features that determine whether the BM25F model and other overall operations of TF in addition IDF are more effective in addition expressive aimed at DBRD than the weighted average of TF in addition IDF. The mathematical formulation of the textual feature can be presented in the following equation,

$$TF_D(T,\ D) = \sum_{F=1}^{K} \frac{w_f \times occurrences(d[f],\ t)}{1 - b_f + \dfrac{b_f \times length}{Average\ Length_f}} \tag{1}$$

$$IDF(t,\ d) = log\frac{N}{|\{D\varepsilon d: t\varepsilon d[f]|} \tag{2}$$

$$BM25F_{ext}(d,\ Q) = \sum_{I\varepsilon d[f]\cap Q[f]} IDF(t,\ total\ text\ fields\ of\ bug\ reports) \times \frac{TF_D(T,\ d[f])}{k_1 + TF_D(t,\ d[f])} \tag{3}$$

$$SizeDiff\,(d,\ Q) = SizeDiff\,(d,\ Q) = abs(|d[f]| - |Q[f]|) \tag{4}$$

where, f can be described as index of the textual arenas of a bug report, $Average\ Length_f$ can be described as average length of the all word in this field, d is defined as document, *length* is defined as the length of the characters in the term (t), k is defined as number of textual fields, w_f is defined as weight factor, k_1 and Q can be described as constant value for avoiding division by zero, || is defined as normalization and *abs* is defined as absolute value [22].

3.2.2 Temporal

Temporal is a one of the feature sets which compute the interval time among two bug documents. The low parameter of this feature presents the huge probability of resemblance of two bug documents. This difference computed based below equations.

$$f_{id}(d,\ Q) = abs(d.BugId - Q.BugId) \tag{5}$$

$$f_{date}(d,\ Q) = abs(d.open\ date - Q.open\ date) \tag{6}$$

3.2.3 Categorical

Categorical can be a one kind of feature which demonstrations how much two bug reports can be related with the consideration of subtraction or equality comparisons of the specific categorical fields. These features are computed based on below equations,

$$f_{product}(q,\ Q) = \begin{cases} 1 & if\ d.product = Q.product \\ 0 & otherwise \end{cases} \tag{7}$$

$$f_{company}(q,\ Q) = \begin{cases} 1 & if\ d.company = Q.company \\ 0 & otherwise \end{cases} \tag{8}$$

$$f_{type}(q,\ Q) = \begin{cases} 1 & if\ d.type = Q.type \\ 0 & otherwise \end{cases} \tag{9}$$

$$f_{priority}(q,\ Q) = \frac{1}{1 + |d.priority - Q.priority} \tag{10}$$

3.2.4 Contextual

It can be a type of feature which can be utilized to contrast textual fields of a bug document with a word list consisting the select gratified such as Latent Semantic Indexing (LSI), Latent Dirichlet Analysis (LDA), Performance of software and security. The results are achieved from the semi textural features denote how much document involves exact settings.

4 Proposed CRF-LSTM

In the proposed system, the proposed classifier is used to differentiate and identify software defects. Independent LSTM and independent CRF models are first produced autonomously. The CRF structure additionally gives the best results in the order phase, which suffers due to the very large data set and the error rate of the arrangement. To implement the CRF model [23], LSTM is integrated with that CRF structure. The proposed classifier design is outlined in Fig. 2.

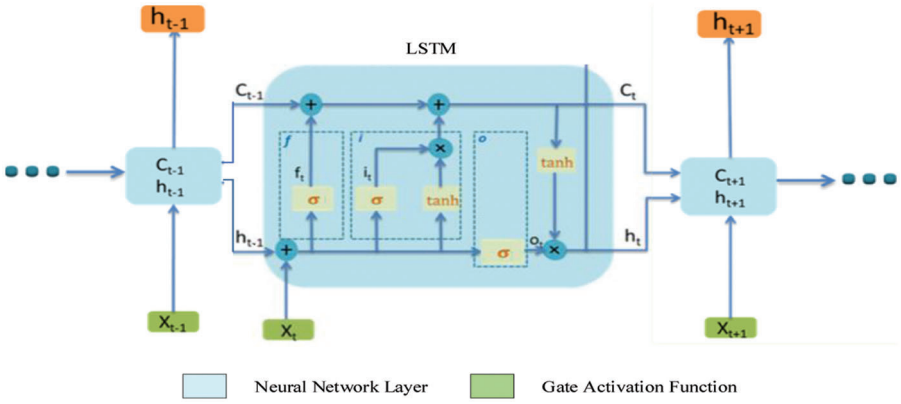

Figure 2: Projected classifier design of CRF-LSTM

4.1 CRF

In the CRF model, the features are utilized to take decision independently that can be tremendously optimal for every output. Moreover, the classification self-sufficiently can be inadequate due to the output has robust dependances. The CRF developed by Lafferty is an optimal solution for software bug detection and classification. CRF is one of the efficient methods which provide efficient classification and detection methods. Let, $x = \langle e_1,\ e_2,..,\ e_n \rangle$ can be described as a genetic input sequence, where e_1 can be described as vector of the i^{th}word. Let, $y = \langle y_1, y_2,.., y_n \rangle$ can be described as a set of LSTM conditions every of that can be related with the respected label. The likely tag arrangements for a verdict x which is computed based on below equations,

$$P(y|x;\ w,\ B) = \frac{\prod_{i=1}^{n} \Psi_i(y_{i-1},\ y_i,\ x)}{\sum \grave{y}\ \varepsilon y \prod_{i=1}^{n} \Psi_i(\grave{y}_{i-1}\ ,\ \grave{y}_i,\ x)} \tag{11}$$

$$\Psi_i(\grave{y}_{i-1}\ ,\ \grave{y}_i\ ,\ x) = exp(w_{\grave{y}}^t x^i + b_{\grave{y}.y}) \tag{12}$$

where, $b_{\grave{y}.y}$ can be described as bias for the label pair $(\grave{y}, y)$ and $w_{\grave{y}}^t$ can be described as the weight vector. Here, the utilization of all-out provisional probability computation aimed at CRF training. The log of likelihood can be mathematically formulated as follows,

$$l(w;\ B) = \sum_i \log p\ (y|x;\ w,\ b) \tag{13}$$

This maximum conditional likelihood algorithm can be utilized to train parameters which exploit the log probability $l(w;B)$. In the decipherment process, the LSTM is utilized and which is utilized to detect the output arrangement which achieve the all-out score for output tag based on the below formulation,

$$y^* = argmax_{y\varepsilon y} p(y|x;\ w,\ b) \tag{14}$$

4.2 LSTM

In CRF engineering, the weighting variable involved in the characterization must be precisely tuned. To enable CRF, the secret layer of CRF is updated in the course of LSTM. Overall, LSTM [24] enjoys the benefits of time series information in the light of planning ability between information and organizes results with contextual information. The end door, input gateway and bypass door of the LSDM network are closed as follows. The weight components of the memory cell, the output gateway, the input door and the bypass door represent the as w^c, w^{out}, w^{in} and w^f separately. In addition, the predictive vectors of the doors mean as b^c, b^{out}, b^{in} and b^f separately.

4.2.1 Forget Gate

Forget gate is the most recent input from the last end memory module is X^i, denoted by the bypass door F^i. The enforcement capacity of the bypass door is denoted as φ^a , and it is selected in a general practice with the prototype of the strategic sigmoid, which activates how much information is placed in the upper cell. The forget gate is numerically created as follows,

$$h^t = \varphi^a(W^F \times [h^{t-1},\ x^t] + b^f) \tag{15}$$

4.2.2 Input Gate

The memory cell can be controller with the assistance of input gate that expressed as shadows,

$$i^t = \varphi^a(W^{in} \times [h^{t-1},\ x^t] + b^{in}) \tag{16}$$

4.2.3 Output Gate

The memory cell output can be skillful with the thought of output gate,

$$o^t = \varphi^a(W^{out} \times [h^{t-1},\ x^t] + b^{out}) \tag{17}$$

4.2.4 Memory Cell

This layer is projected with tanh, and it creates a vector of new up-and-comer values that include the found state,

$$C^T(t) = \tanh(w^c * [h^{t-1},\ x^t] + b^c) \tag{18}$$

$$h^t = o^t * \tanh(C^T) \tag{19}$$

In the memory cell, the state of the old memory cell is updated with the thought of the new memory cell C^T,

$$C^T = f^1 * C^{T-1} + i^T * C^T(t) \tag{20}$$

From the proposed classifier, CRF has been upgraded with the help of LSTM network architecture. CRF's product communication has been improved with the help of the LSTM system. The proposed classifier is used to organize and identify software bugs from information sites.

4.3 Dingo Optimizer

Nature was the most impressive educator from the beginning. Creatures that have always lived on Earth have a system of extraordinary elements for tolerance. Social interactions are one of them, it will change. In light of the overall investigation of the social behavior of the organism, it can be well categorized as part of the classification. The first class relies on natural elements, i.e., close to property access and difficulties caused by various species. Another class relies on personal conduct or excellence. With this in attention, Dingo inspires our work towards strictly adhere to social relationships. Dingo is a type of reed. -Dingo's logical name can be Canis lupus (wolf) Dingo, which was later altered after Canis Familiar (Corine) [25]. Dingos are chaotic, virgin and deeply friendly creatures. Dingos are talented trackers that live up to the normal size 1215 pack. The social progressive system is deeply organized, at the very top of the alpha pecking order, in addition it can be male or female. It can be differentiated related on tasks such as simply determining, enchanting and hunting. The primary and most basic part is usually called the alpha, which is considered the head of the set of dingos. This reflects the fact that discipline and association are a higher priority than power. The test occupied by Alpha can be sent to the pack. As a rule, each person in a pack recognizes the alpha through keeping their ends down.

4.3.1 Mathematical Model

This dingo optimizer is containing different characteristics such as searching, encircling and attacking prey. This mathematical model of the dingo optimizer is presented in this section.

$$\vec{d}_D = |\vec{a}.\,\vec{p}_p(X) - \vec{P}(I)| \tag{21}$$

$$\vec{P}(I+1) = \vec{p}_p(I) - \vec{b}.\vec{d}(D) \tag{22}$$

$$\vec{a} = 2.\overrightarrow{A_1} \tag{23}$$

$$\vec{b} = 2\vec{b}.\overrightarrow{A_2} - \vec{b} \tag{24}$$

$$\vec{b} = 3 - \left(I * \left(\frac{3}{i_{max}}\right)\right) \tag{25}$$

However, at the point of inquiry indicated by the idea, experts usually do not calculate the condition of the prey (at best). Numerically planning a dingo hunting program, we acknowledge that complete pack individuals, counting alpha, beta in addition others, have excellent information around the possible area of prey.

$$\overrightarrow{d_\alpha} = |\overrightarrow{a_1}.\overrightarrow{p_\alpha} - \vec{P}| \tag{26}$$

$$\overrightarrow{d_\beta} = |\overrightarrow{a_1}.\overrightarrow{p_\beta} - \vec{P}| \tag{27}$$

$$\overrightarrow{0d_0} = |\overrightarrow{a_1}.\overrightarrow{p_o} - \vec{P}| \tag{28}$$

$$\overrightarrow{P_1} = |\overrightarrow{P_\alpha}.\vec{b} - \overrightarrow{d_\alpha}| \tag{29}$$

$$\overrightarrow{P_2} = |\overrightarrow{P_\beta}.\vec{b} - \overrightarrow{d_\beta}| \tag{30}$$

$$\overrightarrow{P_3} = |\overrightarrow{P_0}.\vec{b} - \overrightarrow{d_0}| \tag{31}$$

To compute the dingo intensity, the mathematical formulation is presented as follows,

$$\overrightarrow{i_\alpha} = log\left(\frac{1}{f_{\alpha-(ie-100)}} + 1\right) \tag{32}$$

$$\overrightarrow{i_\beta} = log\left(\frac{1}{f_{\beta-(ie-100)}} + 1\right) \tag{33}$$

$$\overrightarrow{i_o} = log\left(\frac{1}{f_{0-(ie-100)}} + 1\right) \tag{34}$$

where, $\overrightarrow{d_\alpha}$ can be described as dingo with best search, $\overrightarrow{d_\beta}$ is defined as second-best search in addition $\overrightarrow{d_0}$ is defined as search result.

4.3.2 Attacking Prey

If the condition worsens, it indicates that the dingo has completed the chase by following the prey. To develop the technique numerically, the value of $\vec{b}$ is directly reduced. Note that the modification purpose of $\overrightarrow{d_\alpha}$ is reduced by $\vec{b}$. It is also known as $\vec{b}$, which is an arbitrary value in the range $[-3b,\ 3b]$ where b is reduced from 3 to 0 during compression. When the →Dα is at irregular overlays [1, 1], the next level of a hunting expert may be in any situation between its current and prey status.

4.3.3 Searching

As mentioned by the package, dingo's often chase prey. They usually move forward to chase and attack predators. In the same way, $\vec{b}$ is used for arbitrary properties, where if the value is incorrect-1, it indicates the prey making a short distance away, but if the value is considered to be more significant than 1, it refers to the pack. Moving towards prey.

Step by step procedure

The dingo optimizer is utilized to select optimal weight parameters in CRF-LSTM for software bug detection.

Step 1: Initial Population

In the initial condition, the random weighting parameters are initialized. Additionally, the population of the dingo optimizer is initialized.

$$d_n(n = 1,\ 2,\ ..,\ n) \tag{35}$$

Step 2: Fitness Function evaluation

After that, the fitness function is evaluated based on Mean Absolute Error (MAE). The MSE should be reduced with the help of DO. The fitness function is formulated as follows,

$$FF = Min(MAE) \tag{36}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n} |Y_i^* - Y_i| \tag{37}$$

Related on the fitness function, the optimal weighting values can be chosen in the proposed classification.

Step 3: Updating Process

The updating process is essential to update the best solutions based on the fitness function evaluation. The dingo positions are updated based on the Eqs. (22)–(28).

Step 4: Termination Condition

Finally, the maximum iteration is checked. This algorithm meets the termination condition, the results are saved. With the help of the DO, the optimal weight values can be chosen which empowers the optimal software bug detection. The performance of the projected technique is assessed in the underneath unit.

5 Outcome Evaluation

The reliability of the proposed technique is evaluated in this area and legalized. The proposed strategy is validated by considering individual estimates such as accuracy, precision, review, awareness, transparency, and F_Measure individually. The proposed technique is different and the current strategies, for example, SVM, ANN, DNN and DBNN separately. The proposed strategy can be carried out by thinking about the collected information base. The handling limits of the proposed technique are given in the Tab. 1. The proposed technique is verified by factual estimates. The implementation estimates of the true estimates are given below in this section.

Table 1: Parameters of proposed approach

S no	Description	Value
1	Inertia factor	0.7298
2	Maximum iteration	100
3	Initial population	50
4	Momentum	0.9
5	Learn rate drop period	5
6	Initial learn rate	0.2
7	Max epochs	0.05
8	Minimum batch size	15
9	Initial learn rate	500

5.1 Dataset Description

These days bug global stabilization structures like Bugzilla are common. Used in programming efforts to store and monitor error information in the framework Error reports. Upgrading programming programs, these error global stabilization structures contain the largest error reports. There are currently about 137 companies, associations and missions. Use Bugzilla to track the confusion of items registered on the Bugzilla site [26]. As pointed out by Intelligence, until March 2019, Bugzilla oversaw More than 546,000 error reports for Eclipse and 1,543,000 separately for Mozilla. When an error is detected, an error report is installed. An analyzer to portray the client or its nuances, e.g., expressions, related properties and steps to repeat this

error. Error in the global stabilization framework, an Error One crash, another component, document update, or a restructuring on the other hand. Evaluation metrics of proposed approach is presented below;

Accuracy:

It is classified from an absolute number to a number of accurately recognized information examples. The recipe for precision was introduced as follows,

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN} \tag{38}$$

***Precision*:** It is classified as part of the positive difference, which is actually correct. Precision successfully implemented the total probability and implementation scale. The definition of review introduced is as follows,

$$Precision = \frac{TP}{TP + FP} \tag{39}$$

***Recall*:** It can be categorized as a ratio of precisely recognized positive examples, to add positive events detected as follows:

$$Recall = \frac{TP}{(TP + FN)} \tag{40}$$

***Specificity*:** It is classified as a ratio of precisely recognized adverse events to include planned negative events in the following,

$$specificity = \frac{TN}{(TN + FP)} \tag{41}$$

***F_Measure*:** F_Measure 0 to 1 should be introduced. Worst value is 0 and best value is 1. F_Measure can be specified as follows,

$$F_{Measure} = \frac{2TP}{(2TP + FP + FN)} \tag{42}$$

where, TP represent the true positive, TN represent the true negative, FP represent the false positive and FN represent the fault negative.

5.2 Experimental Results

To prove the efficiency of our proposed approach, we compare our approach with existing methods. The result obtained from the proposed approach is listed below;

The complete examination of the projected methodology is illustrated in Fig. 3. The comparison validation of the accuracy is given in Fig. 4. In Fig. 4, the proposed methodology has been attained 100 for 10 documents. Similarly, the SVM, ANN and DNN, DBNN has been achieved 98.45, 96, 94 and 92 for 10 documents. Based on the analysis, the projected technique is attained optimal accuracy. The comparison validation of the specificity is given in Fig. 5. In Fig. 5, the proposed methodology has been achieved 100 for 10 documents. Similarly, the SVM, ANN and DNN, DBNN has been achieved 98.45, 96, 94 and 92 for 10 documents. Based on the validation, the projected technique is attained optimal specificity. The comparison validation of the sensitivity is given in Fig. 4. In Fig. 5, the proposed methodology has been achieved 99.87 for 10 documents. Similarly, the SVM, ANN and DNN, DBNN

has been achieved 99.23, 97, 92 and 91 for 10 documents. From the analysis, the projected methodology has been attained optimal sensitivity. The comparison analysis of the F_Measure is presented in Fig. 6. In Fig. 6, the proposed methodology has been achieved 100 for 10 documents. Similarly, the SVM, ANN and DNN, DBNN has been achieved 95.14, 93.15, 97.15 and 95.12 for 10 documents. Based on the analysis, the projected technique is attained optimal F_Measure. The comparison analysis of the computation time is given in Fig. 7. In Fig. 7, the projected methodology has been attained 1 s for 10 documents. Similarly, the SVM, ANN and DNN, DBNN has been achieved 2, 3.5, 5 and 7 s for 10 documents. Based on the validation, the projected technique is attained optimal computation time.

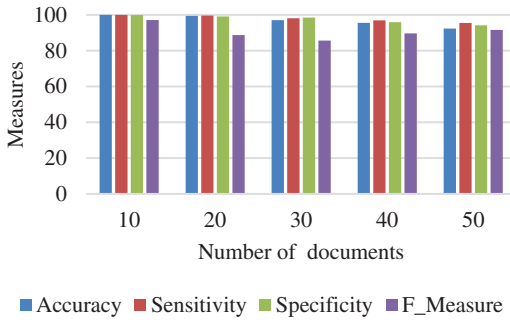

Figure 3: Analysis of proposed method

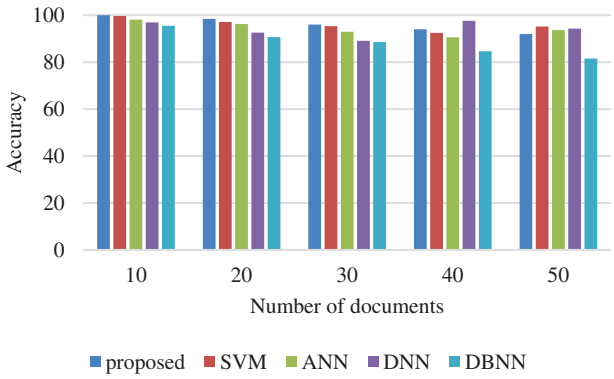

Figure 4: Comparison analysis of accuracy

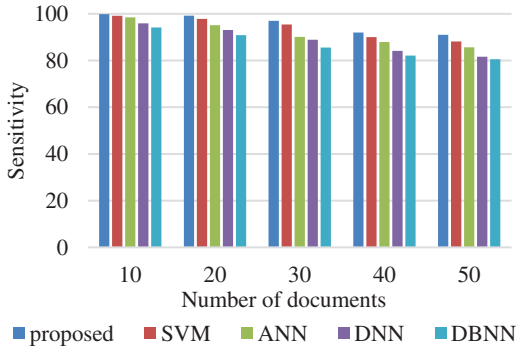

Figure 5: Comparison analysis of sensitivity

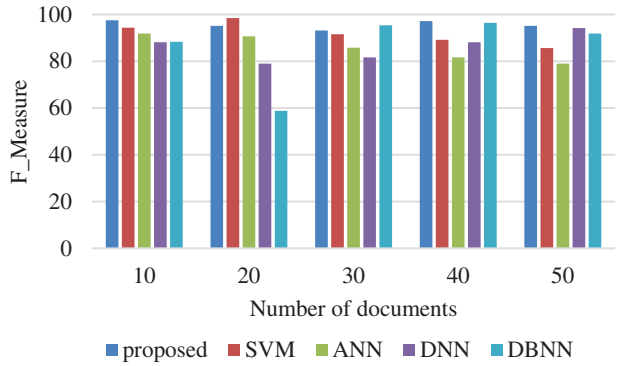

Figure 6: Comparison analysis of F_Measure

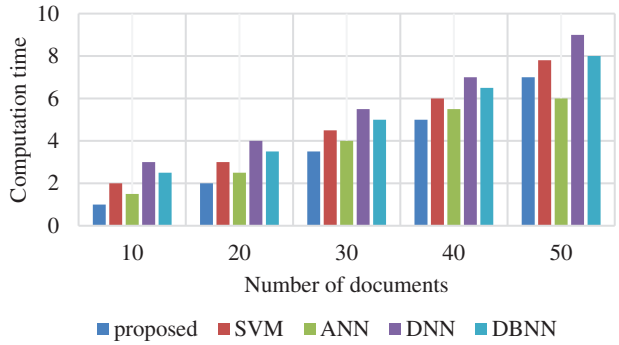

Figure 7: Comparison analysis of computation time

6 Conclusion

In this paper, ADLM for automatic duplicate bug report detection and classification process. The proposed ADLM is a combination of CCRF-LSTM and DO. In the CRF, the DO has been used to choose the optimal weight values in network. The proposed automatic bug report detection will be proceeding with three stages such as pre-processing, feature extraction in addition bug detection with classification. Initially, the bug report database has been gathered from the online system. In the pre-processing phase, the unwanted information from the input data is removed by using cleaning text, convert data types and null value replacement. The pre-processed data has been sent into the feature extraction phase. In the feature extraction phase, the four types of feature extraction method have been utilized such as contextual, categorical, temporal and textual. Finally, the features have been sent to the proposed ADLM for automatic duplication bug report detection and classification. The proposed methodology has been proceeding with two phases such as training and testing phases. Based on the working process, the bugs have been detected and classified from the input data. The proposed methodology has been evaluated by analyzing performance metrices such as accuracy, precision, Recall, F_Measure and kappa. Our method attained the maximum accuracy of 100%, sensitivity of 100%, specificity of 98.7% and F-measure of 97.5% which is high compared to SVM, ANN, DNN and DBNN.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

Reference

[1] C. Savaglio and G. Fortino, "A Simulation-driven methodology for IoT data mining based on edge computing," *ACM Transactions on Internet Technology (TOIT)*, vol. 21, pp. 1–22, 2021.

[2] Z. Parsons and S. Banitaan, "Automatic identification of Chagas disease vectors using data mining and deep learning techniques," *Elsevier, Ecological Informatics*, vol. 62, pp. 1–15, 2021.

[3] S. Hosseini and S. R. Sardo, "Data mining tools-a case study for network intrusion detection," *Springer Multimedia Tools and Applications*, vol. 80, pp. 4999–5019, 2020.

[4] D. Gibert, C. Mateu and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Elsevier Journal of Network and Computer Applications*, vol. 153, pp. 1–22, 2020.

[5] T. A. Tuan, H. V. Long and L. H. Son, "Performance evaluation of botnet DDoS attack detection using machine learning," *Springer Evolutionary Intelligence*, vol. 13, pp. 283–294, 2019.

[6] A. K. Sandhu and R. S. Batth, "Software reuse analytics using integrated random forest and gradient boosting machine learning algorithm," *Wiley Journal of Software Practice and Engineering*, vol. 51, pp. 1–13, 2020.

[7] K. Shi, Y. Lu and G. Liu, "MPT-Embedding: An unsupervised representation learning of code for software defect prediction," *Wiley Journal of Software Evaluation and Process*, vol. 33, pp. 1–20, 2021.

[8] A. Majd, M. V. Asl and A. Khalilian, "SLDeep: Statement-level software defect prediction using deep-learning model on static code features," *Elsevier Expert System with Application*, vol. 147, pp. 1–14, 2019.

[9] T. Lin, X. Fu and F. Chen, "A novel approach for code smells detection based on deep learning, "*Springer EAI Int. Conf. on Applied Cryptography in Computer and Communications*, Cham, vol. 386, pp. 171–174, Jul 2021.

[10] Y. M. Galvao, J. Ferreira and V. A. Albuquerque, "A multimodal approach using deep learning for fall detection," *Elsevier Journal of Expert System and Application*, vol. 168, pp. 1–9, 2020.

[11] X. Cai, Y. Niu and S. Geng, "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," *Wiley Concurrent and Competency Practice and Experience*, vol. 32, pp. 1–14, 2019.

[12] A. Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," *Springer Empirical Software Engineering*, vol. 19, pp. 154–181, 2014.

[13] V. U. B. Challagulla, F. B. Bastani and I. L. Yen, "Empirical assessment of machine learning based software defect prediction techniques," *Scientific World International Journal on Artificial Intelligence Tools*, vol. 17, pp. 389–400, 2007.

[14] M. Li, H. Zhang, R. Wu and Z. H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Springer Automated Software Engineering*, vol. 19, pp. 201–230, 2011.

[15] R. A. Khurma, H. Alsawalqah and I. Aljarah, "An enhanced evolutionary software defect prediction method using island moth flame optimization," *Science Gate Mathematics*, vol. 9, pp. 1–20, 2021.

[16] S. K. Pandey and R. B. Mishra, "BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques," *Elsevier Expert System with Application*, vol. 144, pp. 1–33, 2019.

[17] W. Rhmann, B. Pandey and G. Ansari, "Software fault prediction based on change metrics using hybrid algorithms: An empirical study," *Elsevier Journal of King Saud University-Computer and Information Sciences*, vol. 32, pp. 419–424, 2020.

[18] R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with android software," *Elsevier Applied Soft Computing*, vol. 49, pp. 1034–1050, 2016.

[19] B. Khan, R. Naseem and M. A. Shah, "Software defect prediction for health care big data: An empirical evaluation of machine learning techniques," *Hindawi Journal of Health Care Engineering*, vol. 2021, pp. 1–16, 2021.

[20] M. Faseeha, S. Aftab and A. Iqbal, "A framework for software defect prediction using feature selection and ensemble learning techniques," *Readera International Journal of Modern Education and Computer Science*, vol. 11, pp. 14–20, 2019.

[21] H. A. Almuzaini and A. M. Azmi, "Impact of stemming and word embedding on deep learning-based arabic text categorization," *IEEE Access*, vol. 8, pp. 127913–127928, 2020.

[22] B. S. Neysiani, S. M. Babamir and M. Aritsugi, "Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems," *Elsevier Information and Software Technology*, vol. 126, pp. 1–27, 2020.

[23] Y. Jin, J. Xie and W. Guo, "LSTM-CRF neural network with gated self-attention for Chinese NER,"*IEEE Access*, vol. 7, pp. 136694–136703, 2019.

[24] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Elsevier Physica D: Nonlinear Phenomena*, vol. 44, pp. 1–43, 2020.

[25] A. K. Bairwa, S. Joshi and D. Singh, "Dingo optimizer: A nature-inspired metaheuristic approach for engineering problems," *Hindawi Mathematical Problems in Engineering*, vol. 2021, pp. 1–12, 2021.

[26] A. Ahmed, https://github.com/ansymo/msr2013-bug_dataset, Apr 2015.