An Efficient Software Defect Prediction Model Using Rule-Based DLMNN Classifier

[*1]Prasad V. S., [2]Dr. Sasikala K.

[*1]Research Scholar, Department of Computer Science, Vinayaka Mission's Kirupananda Variyar Engineering College (Vinayaka Mission Research Foundation (Deemed to be University)), Salem, India

[2]Associate professor, Department of Computer Science and Engineering, Vinayaka Mission's Kirupananda Variyar Engineering College, (Vinayaka Mission Research Foundation (Deemed to be University)), Salem, India.

ABSTRACT

The primary objective is to resolve those problems by proposing a rule-centered Deep Learning Modified Neural Network (DLMNN) classifier for predicting the software defect (SD) effectively.

The proposed method comprises '7' steps. Pre-processing is the initial step; the repeated data are eliminated, missing values are restored, strings are converted, and the data are normalized here. Secondly, as of the pre-processed data, software project features are extracted, and also the essential features are chosen by utilizing the Hybrid Grass Hopper with Genetic Algorithm (HGHGA). After that, centered on threshold values (TV) of the chosen features, the rule is created and the generated rules are proffered to the DLMNN aimed at the SDP. For enhancing the accuracy and also decreasing the time, the Cockroach Swarm Optimization (CSO) algorithm is used for updating the neuron's weight value of the layers.

1. INTRODUCTION

The significance of software reliability has been augmented by the growing complexity of modern software. A substantial amount of testing and debugging is required for making extremely dependable software. Thus, SDP methods are used which predict the existence of bugs. It is extensively utilized for assisting developers in prioritizing their testing along with debugging efforts [1]. Attaining higher software quality along with reliability with the efficient usage of available restricted resources is the SDP's goal. In other terms, SDP includes detecting software modules or components that are susceptible to defects. SD is mostly resulting from the software's complex source code, which may result in software failure . Therefore, for

locating the defect-prone regions in the software system together with rectifying those defects at an initial stage for acquiring qualitatively enhanced software, there prevails a necessity to examine and implement a suitable technique Thus, SDP has emerged as one among the research topics in the data mining domain of software engineering. It will assist developers along with testers to find SD in time.

For building defect prediction methods, disparate ML methods are utilized by numerous existent researchers. Especially, random forest (RF), decision tree, Naive Bayes, support vector machine (SVM), nearest neighbor, neural network, logistic regression, ensemble method, etc are the several classification methods that are utilized as defect prediction model (DPM) . A promising means for effectual along with efficient SDP are ML techniques. These methods are classified as supervised along with unsupervised. For training the classifier, the supervised technique requires pre-classified training data. The rules are developed which are further utilized for categorizing the unseen data (test data) while training. For recognizing the classes and maintaining the software data, no training data is needed as these methods utilize a specific algorithm in unsupervised techniques . Firstly, for fault proneness, these techniques aim to detect a group of source code associated along with process-related features that can function as classifiers. Secondly, using various ML algorithms, these features are applied for training ML models. They are used for categorizing whether a recently seen software component is defect-prone or not after such models are trained.

Concerning several faulty units, SDP includes few fault datasets that may exhibit disparate quality. The faulty classes are very rare within the dataset therefore, it doesn’t indicate that these errors must be neglected, but rather it is vital to capture them . As there are disparate faultless units compared to faulty ones, this is called the class imbalance issue. Normally, for solving this issue, prevalent methods like sampling, cost-sensitive, along with ensemble learning (EL) methods were utilized. Disparate feature selection (FS) methods are utilized for choosing an optimum feature subset aimed at SDP as irrelevant along with redundant features in the defect data might degrade the classification model’s performance. Therefore, for the SDP, the optimum features are proffered to the classifier. But, owing to weight updation, the existing classifiers still possess accuracy issues. For overcoming this concern, the rule-centered DLMNN classifier is utilized by the proposed one for predicting the SD.

This paper is categorized as: Section 2 described the existent research methods of the SDP. Section 3 offers the proposed research method’s detail. Section 4 contrasted the proposed work’s performance with the prevailing methods. Section 5 concluded the paper along with future improvement is elucidated.

2. RELATED WORK

C. Manjula and Lilly Florence [20] introduced a hybridized system for software prediction by software metrics centered upon a deep neural network (DNN). From the feature set, the features were initially chosen. Next, utilizing the Genetic Algorithm (GA), the attained features were optimized. Next, the DNN was implemented for categorization. The approach executed better when contrasted with prevailing techniques as shown by the experimental outcome. The recurred fitness function on a complex problem that degraded the SDP performance was not predicted by the GA algorithm.

Lei Qiao et al. [21] specified deep learning (DL) methods for anticipating the total defects in software systems. An openly accessible dataset was initially pre-processed, incorporated with log transformation together with data normalization. Secondly, for preparing the data input for the DL model, data modeling was conducted. Thirdly, aimed at classification, the modeled data was transferred to a specifically designed DNN. The method was more precise along with enhanced upon the top-notch approach as exhibited by the assessment result. However, via a restricted number of source codes, the approach performance was tested. Higher accuracy wasn't offered by the DNN approach for a few source codes that affected the system's effectiveness.

Hamad Alsawalqah *et al*. [22] expressed a SDP by Heterogeneous Ensemble Classification centered upon Segmented Patterns. A clustering method was implemented to the training data for segmenting it into disparate groups of similar occurrences in the 1st stage. Centred upon the created groups as of the 1st phase, disparate classifiers were trained in the 2nd phase. The developed models were evaluated by the 3rd phase and utilized for anticipating the unrepresented instances. The prediction power was considerably boosted by the approach's ensemble version analogized to the ensemble along with basic classifiers in many datasets as revealed by the experimental result. The SDP was affected because the ensemble classification method was computationally costly.

Tianchi Zhou et al. [23] introduced a deep forest model for building the DPM. The EL along with DL's idea was acquired by the framework. The forest classifiers were transformed by the method into a layer-by-layer structure and merged the RF classifiers in every layer centered upon Defect Prediction Deep Forest (DPDF), for performing the role of SD categorization. Using a layer-by-layer training method to attain the DL's advantage and merge the RF classifiers to encourage diversity and augment the model's robustness for identifying more significant defect features was the DPDF model's major idea. Superior performance when weighted against the top conventional ML algorithms was displayed by the approach's

experimental result. Some irrelevant information was incorporated in the input data, which was applied straightly to the classifier. It affected the classifier's computational time.

Thanh Tung Khuat and My Hanh Le [24] stated an SDP system centered ensemble of Classifiers upon imbalanced data. Initially, the EL's classification efficiency was severely impeded by the imbalance property of SD data. The consequence of sampling regarding ensembles of different classifiers on unbalanced data in SDP problems was empirically evaluated by the approach. Extensive experiments were performed over the datasets with the amalgamation of '7' disparate classification algorithms, '3' sampling techniques, along with '2' balanced data learning scheme. The positive impacts of merged sampling techniques together with the EL model upon the defect prediction's performance concerning datasets with unbalanced class distributions were denoted by the experimental outcome. A better result was proffered by the approach for fewer software projects only. It wasn't well-matched with the modified software project.

Haijin J et al. [25] introduced an SDP centred upon the Naive Bayes (NB) method. '6' weight assignment techniques were examined by the approach for setting the feature weights. Next, the most apt one centered upon the F-measure was chosen. The information diffusion model (IDM) was employed for the calculation of probability density of every feature in place of the compliant probability density function (PDF) of the normal distribution (ND) aimed at the ND assumption. For comparison, numerous familiar classifiers, and also ensemble methods, were incorporated. The effectiveness along with practicability was exhibited by the last experimental outcomes. The software project possessed more features, and the NB did not obtain a precise result at all times owing to its complexity.

3. PROPOSED SOFTWARE DEFECT PREDICTION USING DLMNN

In software systems, defects are frequent and can possibly cause different concerns to software users. For rapidly predicting the most probable locations of defects in huge codebases, disparate methods are developed. A complex task is an initial prediction of defects that must be addressed and could be enhanced by obtaining a greater classification rate of defect prediction. For solving this issue, a hybridized approach is introduced by this research methodology by amalgamating optimization with neural networks for classification. Dataset collection, pre-processing, feature extraction (FE), FS, rule generation, SDP, along with K-fold testing are the proposed work's 7 phases. From the publically accessible resources, all the data are initially gathered. Afterward, the pre-processing is executed for the available dataset. Then, line count of code, cyclomatic complexity, design complexity, essential complexity, program length, etc.,

are the source code's features, which are extracted as of the data (pre-processed). Next, using HGHGA, the selection of significant features as of the extracted features is done. Afterward, centred on the TV, the rule is generated from the features (selected). Then, the feature's generated rules are inputted into the DLMNN algorithm. The software's source code is classified by this algorithm as defected or non-defected. Figure 1 exhibits the block diagram for the research technique,

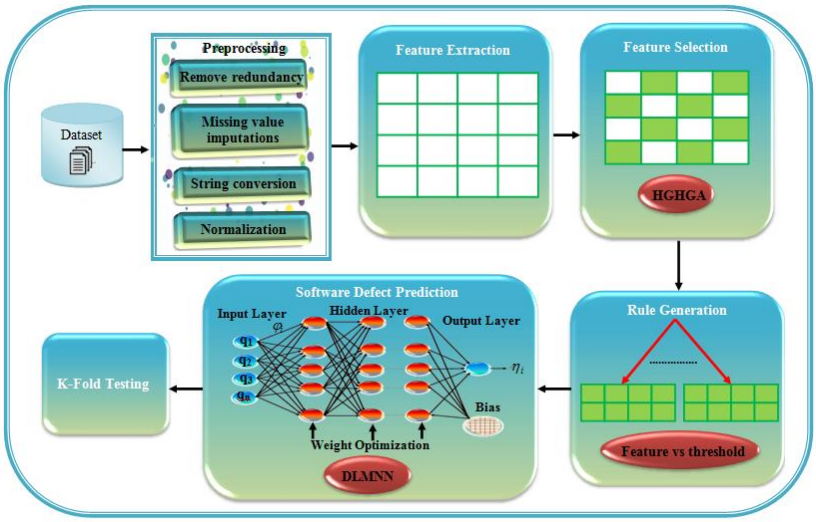

Figure 1: Block diagram for the proposed software defection prediction methodology

3.1 Data Collection

From the '5' PROMISE dataset namely KC1, KC2, PC1, PC3, and also PC4, the input data are initially gathered. The gathered data are specified as,

$$S_i = \{b_i, i \in n\}$$

Wherein, S_i specifies the collected dataset and n indicates the n-number of input data proffered in the taken dataset.

3.2 Pre-processing

By deeming the removal of redundancy, missing value imputation, string conversion, along with normalization, the gathered data are pre-processed in this section. Firstly, the repeated data are eliminated. Then, by the computation of the other value's mean value, the missing values are changed, which is stated as,

$$B_i = \frac{\sum_{i=1}^{n} b_i}{n}$$

Wherein, B_i signifies the missing values, b_i symbolizes the i^{th} value of the data collection (input), along with n denotes the total data. If the dataset includes any text that shall be changed by numbers after missing value imputation. Next, since the raw data contains attributes with differing scales, the normalization process is performed. Therefore, it normalized numeric attributes as of the range of 0 to 1. The normalization function is derived as,

$$S_i^{'} = \frac{b_i - m_n(b_i)}{m_x(b_i) - m_n(b_i)} * (U - D) + D$$

Wherein, $S_i^{'}$ specifies the normalized output, $m_n(b_i)$ signifies the data's minimum value, $m_x(b_i)$ indicates the data's maximum value, and U, D symbolizes the predefined boundary.

3.3 Feature Extraction

The extraction of features from the pre-processed data is performed here. Therefore, the features are code's line count(z_1), Cyclomatic complexity(z_2), Essential Complexity(z_3), Design Complexity(z_4), Halstead total operators along with operands(z_5), Halstead Effort(z_6), Halstead Line Count(z_7), Flow graph branch count(z_8), Program length(z_9), etc. Hence, the feature set (Z_s) is exhibited as,

$$Z_s = z_i, i \in n$$

3.4 Feature Selection

The vital features are selected from the extracted features after FE. The HGHGA is utilized for the FS here. The insect swarm behavior of grasshoppers is simulated by the Grasshopper optimization algorithm. For finding a new habitat with food, the grasshopper swarm migrates over a huge distance. The interaction amongst grasshoppers affects each other within the swarm in this process. The grasshopper's trajectory is impacted by the wind's power and the gravity outside the swarm. An essential influence factor is the food target. Easily falls into local optimal along with slow convergence speed are the few problems of the general grasshopper algorithm. The crossover mutation process is used by this research technique before the updation strategy for the improvement of performance. Social communication, power of gravity, and also wind advection is the '3' major elements affecting the grasshopper's individual movement. The mathematical form of their swarm behaviors is indicated by the subsequent formula:

$$H_l = C_l + G_l + A_l$$

Wherein, H_l defines the l^{th} grasshopper's position, C_l implies the social interplay defined in Eq. (6), G_l implies the gravity force upon the l^{th} grasshopper in Eq. (10), and A_l exhibits the air advection Eq. (11). The C_l is indicated as,

$$C_l = \sum_{k=1,k\neq l}^{N} c(\delta_{lk})\hat{\delta}_{lk}$$

Where, c implies a function which indicate the social force's strength betwixt these grasshoppers, δ_{lk} signifies

.

3.5 Rule Generation

The rule is created for the selected features after FS. Initially, the threshold is set for the feature i.e. the TV is allotted for every feature. Then, the amalgamation of those allotted TV is generated. Therefore, the generated rule is specified as,

$$R_s = \{q_{1,}q_{2,}q_{3,}.....,q_n\}$$

Where, R_s signifies the generated rule set, and q_n defines the n-number of rules.

3.6 Defect Prediction by DLMNN Algorithm

Therefore, the created rules are provided as input to the DLMNN algorithm. The n-number of the hidden layers (HL) is deemed, and it possesses the weight updation issue in a normal artificial neural network (ANN). The Crow Search Optimization (CSO) is used by this research methodology for overcoming the issue. The better weight value is selected by this algorithm together with lessens the iteration of the weight selection. In a smaller time, an ANN can execute perceptual and recognition tasks. A problem's non-linearity is exploited by a neural network (NN) for defining a set of desired inputs. The ANN comprises '3' basic components, like input layer (IL), HL, along with output layer (OL). The information that is fed on the network is possessed by the IL. Initially, this information is rather raw. Processing the raw information attained from the IL into something that might be utilized by the OL is the HL's fundamental job. More than one HL can be possessed by the ANN architecture. The information acquired from the HL is submitted to the OL and also it is processed for generating the desired results. Figure 2 exhibits the proposed DLMNN's structure,

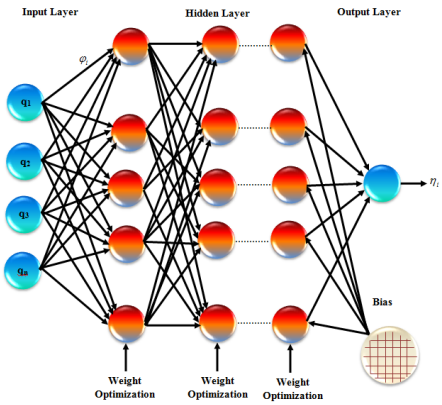

Figure 2: Structure of the DLMNN algorithm

The produced rules are presented as input to the IL. Thereafter, from the obtained IL, the raw information is inputted to the HL. The hidden unit is measured for the input values in the HL. The hidden unit formula is articulated as,

$$dd_i = \chi + \sum_{i=1}^{n} q_i . \varphi_i$$

Wherein, dd_i denotes the HL, φ_i symbolizes the particular layer's weight value, χ denotes the bias value along with q_i signifies the input data. Next, by adding every weight of the input values, the output unit is computed. This is basically the calculation for obtaining the neuron's value in the OL.

$$\eta_i = \chi + \sum_{i=1}^{n} dd_i . \varphi_i$$

Where, η_i characterizes the output unit, and then estimate the loss function (LF) using the equation (18),

$$error = (ob_t - \eta_i)$$

Wherein, η_i implies the output unit, $error$ defines the LF along with ob_t indicates the NN's target output. Herein, this research methodology utilizes the CSO algorithm for lessening the LF. The cockroach's social behavior inspires the CSO. Here, the layer's weight value is deemed as the cockroach. Figure 3 depicts the DLMNN classifier's pseudocode,

Input: Generated rules $R_s = \{q_1, q_2, q_3, \ldots\ldots, q_n\}$
Output: software defected (or) not

Begin
Initialize neurons, input layer, hidden layer, output layer, weight value φ_t, and loss threshold as_r.
Calculate the hidden unit and output unit by,
$dd_t = \chi + \sum_{t=1}^{n} q_t . \varphi_t$ and $\eta_t = \chi + \sum_{t=1}^{n} dd_t . \varphi_t$
Check loss function
if $(loss \geq as_r)$ {
Select the weight value of neurons by CSO
// Weight updation by CSO
Set iteration $tr = 0$
Evaluate fitness function
While $(tr < \max)$ **do**
// Chase-swarming behaviour
if $X_{G,t} \neq w_{G,t}$ **then,**
$X_{G,t} + \omega.\psi_1[0,1].(x_{b,t} - X_{G,t})$
else
$X_{G,t} + \omega.\psi_2[0,1].(x_{g,t} - X_{G,t})$
end if
// Dispersion behaviour
for all cockroach **do**
$X_b = X_b + \gamma(1, D)$
end for
// Ruthless behaviour
For all cockroach **do**
$X_y = x_g$
End for
Set $tr = tr + 1$
end while
Return optimal solution

} **else** {
Denote the output is the final output
}
end if
End

Figure 3: Pseudo code for the DLMNN classifier

The algorithm concentrated on forming a sequence of probable solutions in its first step. Generally, the initial solutions are arbitrarily produced in the search space. By imitating the behaviors, which are chase-swarming, dispersing, ruthless behavior, its model is produced. The chase-swarming behaviour is executed as,

$$X_{G,t+1} = \begin{cases} X_{G,t} + \omega.\psi_1[0,1].(x_{b,t} - X_{G,t}), & X_{G,t} \neq w_{G,t} \\ X_{G,t} + \omega.\psi_2[0,1].(x_{g,t} - X_{G,t}), & X_{G,t} = w_{G,t} \end{cases}$$

Wherein, $X_{G,t}$ signifies the cockroach's existing position at the t^{th} generation, ω indicates the constant value, ψ_1 and ψ_1 are a arbitrary number in [0, 1] and also x_b implies the personal best position, which can well be gauged as,

$$x_b = Opt_j \left\{ x_j \middle| x_b - x_j \right| \leq visual \right\}$$

Where, visual implies the perception constant, $x_{g,t}$ signifies the global top position at the t^{th} generation.

$$x_g = Opt\{X_b\}$$

The dispersion of individuals is another procedure. For preserving the cockroach's diversity, it is conducted from time to time. The procedure contains every cockroach executing a random step in the search space. The dispersion process is signified as,

3.7 K-Fold Testing

In the designed DLMNN classifier, the K-Fold testing will be finally performed. The system shall be checked by some test data after training to verify if the system was trained properly. For training, 80% of data was provided along with 20% of data was presented for testing from the dataset.

4. RESULT AND DISCUSSION

The proposed method's performance is examined in this section. The proposed SDP system utilizing rule generation, and also the DLMNN classifier was applied in JAVA.

4.1 Database Description

Herein, for experimental analysis, the PROMISE open-source datasets are deemed. '5' datasets as of the PROMISE repository are taken by this research methodology which is named KC1, KC2, PC1, PC3, along with PC4. In C or C++ language, these datasets were developed. It is basically a National Aeronautics and Space Administration- Metrics Data Program (NASA-MDP) software projects as of satellite flight control, instrumentation spacecraft, storage management of ground data, along with scientific data processing.

4.2 Performance Analysis

Concerning precision, recall, sensitivity, accuracy, specificity, and also F-Measure, the DLMNN classifier's performance is examined with the prevailing Adaptive Neuro-Fuzzy Inference System (ANFIS), SVM, ANN, and also K-Nearest Neighbor (KNN). The method's accuracy performance is deemed, which is tabularized below.

Table 1: Performance analysis of the proposed DLMNN classifier with the existing classifiers based on accuracy metric

Datasets	Proposed DLMNN	ANFIS	KNN	SVM	ANN
KC1	95.89785	89.21454	90.54774	91.78456	93.87541
KC2	96.12458	89.65785	90.32567	92.45784	93.58741
PC1	95.87452	88.32458	91.45784	92.54782	93.57484

PC3	95.32458	88.65875	90.78451	91.65874	94.65898
PC4	96.23458	89.32657	91.65785	92.54785	93.23659

Centered on accuracy metrics for '5' datasets in the PROMISE database, table 1 exhibits the proposed DLMNN's performance and the ANFIS, KNN, SVM, along with ANN classifiers. The division of the overall number of correct predictions by the total predictions made for a dataset is defined by the accuracy metric. Herein, for the KC1 dataset, the DLMNN classifier's accuracy is 95.89%, 96.12% for KC2, 95.87% for PC1, 95.32% for the PC3 dataset, and also 96.23% for the PC4, which is the maximum amongst every other method. The existent ANFIS algorithm's accuracy is 89.21% for the KC1 dataset, 89.65% for the KC2, 88.32% for the PC1 dataset, 88.65% for the PC3 dataset, along with 89.32% for the PC4 dataset. Amongst all other techniques, this has lesser performance. The ANN has lesser performance contrasted to the proposed classifier. Nevertheless, it is superior contrasted to other existent research methods. So, it is inferred that greater accuracy is acquired by the DLMNN classifier-centered SDP system analogized to the prevailing research methodologies. Figure 3 exhibits the graphical depiction of the accuracy analysis,

. The proposed DLMNN's precision value is 95.78%, 94.89%, 95.12%, 95.32%, and 94.87% for every dataset i.e., KC1, KC2, PC1, PC3, and also PC4, correspondingly. The prevailing method's precision value is lesser when analogized to the DLMNN proposed for the same datasets. Figure 4 (b) examines the performance regarding the recall metric. It obtains the ratio of accurate positive predictions to the overall positives. The DLMNN classifier possesses 95.65% recalls in the PC1 dataset, although for the same PC1, the existing method's recall value has 88.68% for ANFIS, 91.48% for KNN, 92.45% for SVM, and also 93.78% for the ANN methods. This performance is lower when analogized with the DLMNN proposed classifier. Likewise, the existent research methods possess a lower recall value analogized to the proposed one for the rest of the dataset also. Therefore, the discussion of figure 4 affirmed that the proposed DLMNN-centered SDP is extremely appropriate.

The proposed HGHGA's fitness value (FV) with the prevailing Particle Swarm Optimization (PSO), GA, Cuckoo Search Optimization (CSO), and Grass Hopper Algorithm (GHA) concerning several iterations is examined in figure 8. For examining the performance, 5, 10, 15, 20, and 25 iterations are deemed. The HGHGA has 107 FV in the 25^{th} iteration. In the same iteration, the prevalent algorithms, like GHA, GA, CSO, along with PSO have 97, 87, 77, and

also 72 FV. Herein, a higher FV is possessed by the proposed HGHGA than the prevalent algorithms. Superior performance is obtained by the proposed when analogized to the prevailing algorithms for the rest of the iterations. It is affirmed that better results are attained by the proposed HGHGA centered FS process than the existent algorithms.

Table 2: Analysis of rule generation time of the proposed methodology

Datasets	Rule generation time (in ms)
KC1	25273
KC2	23378
PC1	24471
PC3	18767
PC4	21744

Table 2 exhibits the proposed one's rule generation time in different datasets. Lower time is taken by PC3 dataset in a generation that is 18767 ms, secondly, the PC4 dataset requires 21744ms time, thirdly, 23378ms time is needed by the KC2, fourthly, the PC1 takes 24471ms time. Finally, 25273ms time is taken by the KC1 dataset for the generation of rules.

5. CONCLUSION

The historical data are utilized by the DPM that is acquired from software projects for training along with testing the model upon the software's future release. For SDP, the rule-centered DLMNN classifier is used here. The research technique comprises data collection, pre-processing, FS, FE, rule generation, classification, and also K-fold testing. In the pre-processing stage, the data are neatly organized and the features, namely code's line count, Cyclomatic complexity, design complexity, essential complexity, etc., are extracted. Afterward, the major features are elected by HGHGA. Then, for the elected features, the rule is produced and the rule output is inputted to the DLMNN. The software project is predicted by the classifier whether it has defected or not defected. Using '5' datasets of PROMISE, the system's performance is deemed. Centered on precision, accuracy, F-measure, recall, sensitivity, training time, along with specificity, the proposed work's performance is analogized with the existent ANFIS, SVM, KNN, and also ANN. For datasets KC1, KC2, PC1, PC3, along with PC4, the proposed DLMNN's accuracy is 95.89%, 96.12%, 95.87%, 95.32%,

and 96.23%, correspondingly. This accuracy is greater when analogized to the prevalent methods. In addition, centered on several iterations, the proposed HGHGA's fitness is analogized to optimization methods. The proposed one has superior fitness results analogized to the methods in this comparison also. Therefore, superior results are attained via this proposed DLMNN centered SDP. The proposed DLMNN could be enhanced by improvement of prediction accuracy utilizing improved classified methods with a few additional intermediate steps. The non-defected software's quality is attained in the upcoming future.

ACKNOWLEDGEMENTS

The authors would like to thank the Deanship of Vinayaka Mission's Kirupananda Variyar Engineering College for supporting this work.

DATA AVAILABILITY STATEMENT

All the data is collected from the simulation reports of the software and tools used by the authors. Authors are working on implementing the same using real world data with appropriate permissions.

FUNDING

No fund received for this project

COMPLIANCE WITH ETHICAL STANDARDS

- **CONFLICTS OF INTEREST**

The authors declare that they have no conflict of interest.

- **ETHICAL APPROVAL AND HUMAN PARTICIPATION**

No ethics approval is required.

REFERENCES

[1] C. Pan, M. Lu, B. Xu, & H. Gao, 2019. An improved cnn model for within-project software defect prediction. *Applied Sciences*, *9*(10), 2138.

[2] A.O. Balogun, S. Basri, S.J. Abdulkadir, & A.S. Hashim, 2019. Performance analysis of feature selection methods in software defect prediction: a search method approach. *Applied Sciences*, *9*(13), 2764.

[3] L.A.F. Gomes, R. da Silva Torres, & M.L. Côrtes, 2019. Bug report severity level prediction in open-source software: A survey and research opportunities. *Information and software technology*, *115*, 58-78.

[4] W. Zheng, C. Feng, T. Yu, X. Yang, & X. Wu, 2019. Towards understanding bugs in an open-source cloud management stack: An empirical study of OpenStack software bugs. *Journal of Systems and Software*, *151*, 210-223.

[5] T. Shippey, D. Bowes, & T. Hall, 2019. Automatically identifying code features for software defect prediction: Using ast n-grams. *Information and Software Technology*, *106*, 142-160.

[6] S. Ghosh, A. Rana, & V. Kansal, 2019. Statistical assessment of nonlinear manifold detection-based software defect prediction techniques. *International Journal of Intelligent Systems Technologies and Applications*, *18*(6), 579-605.

[7] G. Fan, X. Diao, H. Yu, K. Yang, & L. Chen, 2019. Software defect prediction via attention-based recurrent neural network. *Scientific Programming*, *2019*.

[8] S. Zheng, J. Gai, H. Yu, H. Zou, & S. Gao, 2020. Software Defect Prediction Based on Fuzzy Weighted Extreme Learning Machine with Relative Density Information. *Scientific Programming*, *2020*.

[9] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, et.al. 2019. Software defect prediction based on kernel PCA and weighted extreme learning machine. *Information and Software Technology*, *106*, 182-200.

[10] X. Chen, D. Zhang, Y. Zhao, Z. Cui, & C. Ni, 2019. Software defect number prediction: Unsupervised vs supervised methods. *Information and Software Technology*, *106*, 161-181.

[11] W. Fu, & T. Menzies, 2017. Revisiting unsupervised learning for defect prediction. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering* (pp. 72-83).

[12] M.F. Sohan, 2019. *The Impact of Multi-structured Datasets on Cross-project Software Defect Prediction* (Doctoral dissertation, Daffodil International University).

[13] F. Wang, J. Ai, & Z. Zou, 2019. A cluster-based hybrid feature selection method for defect prediction. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)* (pp. 1-9). IEEE.

[14] V. Bolón-Canedo, & A. Alonso-Betanzos, 2019. Ensembles for feature selection: A review and future trends. *Information Fusion*, *52*, 1-12.

[15] S. Qiu, L. Lu, S. Jiang, & Y. Guo, 2019. An investigation of imbalanced ensemble learning methods for cross-project defect prediction. *International Journal of Pattern Recognition and Artificial Intelligence*, *33*(12), 1959037.