DOI: https://doi.org/10.46632/eae/1/1/4

Electrical and Automation Engineering
Vol: 1(1), 2022
REST Publisher; ISBN: 978-81-956353-5-1
Website: http://restpublisher.com/book-series/electrical-and-automation-engineering/

Software Defect Prediction and Software Quality Assessment Using Dlr-Lvq and Fuzzy Rules

***V S. Prasad, K. Sasikala**
Vinayaka Mission's Kirupananda Variyar Engineering College (Vinayaka Mission Research Foundation (Deemed to be University)), Salem, Tamil Nadu, India.
*Corresponding author Email: prasadvs_83@yahoo.com

Abstract. Recently, Software development has been considerably grown. Fault in the software causes fault and interrupts the output. Characteristics like these make it much challenging to avert software flaws. Spontaneously forecasting the amount of flaws within the software modules is essential and also can assist developers to proficiently allot restricted resources. Recently, numerous Software Defect Prediction (SDP) techniques are developed. But, the accuracy and time-consuming challenges still remain to be solved. Also, a few top-notch techniques don't properly classify the software whereas it is a needed metric to ensure quality standards. This work proffers a novel Decaying Learning Rate – Learning vector Quantization (DLR-LVQ) classifier to forecast the software defect. The proposed methods consist of the following steps: redundant data removal, feature extraction (FE), feature oversampling, data normalization, defect prediction (DP), and quality prediction. The proposed DLR-LVQ's attained outcome is assessed with the existent methodologies. The outcomes exhibit that the methodology proposed attains efficient classification outcomes are examined.

Keywords: Software Defect Prediction (SDP), Non defective software quality prediction, BM-SMOTE, Decaying Learning Rate, Learning Vector Quantization, Fuzzy rules, HDFS and Map Reduce.

1. Introduction

Technical progressions in the software systems are created recently and they become progressively versatile and also powerful [1]. Software's quality is a crucial ingredient aimed at software triumph [2]. After numerous ages of research, SDP stays to be a significant research subject prevalent in the software engineering field [3]. The proficiency of a Software Development Life Cycle's (SDLC's) testing phase can be incremented utilizing SDP [4]. Whilst a software module's flaw can be forecasted in advance via examining its source code or else development procedure, the software developer (programmer) concentrates on those defective modules aimed at boosting the software's quality [5]. Software consequences, namely fault and failures, can decrement the software quality that directs towards customer's dissatisfaction [6]. The SDP identifies the defective software components. Software maintenance and as well evolution are crucial to guarantee a software system's quality [7]. The cost of even deprived software's quality is much high [8]. Even though its significance and widespread applicability, the defective software modules' detection is an intricate operation, principally in intricate and also large-scale software systems [9]. SDP will as well aid assigning testing resources expeditiously via ranking software system modules in handling their flaws [10]. Thus, automated prediction of software's defect-prone components is a vital and fundamental activity for highly dynamic software systems in the software development procedure course [11]. Diverse SDP are proffered to boost the software's quality over the previous few decades [12]. Aimed at boosting the software illness prediction, the code profiles' usage as function variables in the traditional metrics' region is required [13]. Since the 1970s, SDP technology has begun to develop gradually. The current design is primarily implemented centred on a machine learning (ML) technique [14]. ML has emerged as a dominant tool in material science aimed at building precise predictive material designs rapidly and also automatically aimed at the novel materials' rational design [15]. This paper proffered a novel classifier aimed at SDP and to determine software quality. This work's draft structure is schematized as: section-2 reviews the associated works concerning the protocol proposed; section-3 presented the proposed work's brief explication; section-4 emplicated the experiential outcome; section-5 reveals the conclusion.

2. Related Work

Kechao Wang et al. [16] developed a minimal absolute contraction and selection operator centred on Support Vector Machine (LASSO–SVM) aimed at SDP. SVM's optimal value was acquired employing the cross-validation algorithm's parameter optimization capability. The simulation results' accuracy was 93.25%, the recall rate was 78.04%, and as well the f-metric was 72.72%. However, there exists an issue whilst the cross-validation chooses the SVM optimal parameters. Jinyin Chen et al. [17] introduced a Collective Training procedure aimed at DP (CTDP) that comprises '2' phases: source data expansion stage and then adaptive weighting stage. CTDP made the source and also target projects' feature distributions identical to one another via transfer learning and utilized the particle swarm optimization technique aimed at widely pondering the diverse source projects aimed at predicting the target project. Outcomes exhibited that CTDP efficiently boosted the performance. But, a few projects had negative transfer impacts on others, directing towards bad predictions analogized to the direct CPDP. Hongliang Liang et al. [18] developed vector sequences and also their labels (defective or else non-defective) for creating a Long Short Term Memory (LSTM) network. The LSTM design spontaneously could learn the programs' semantic information and also execute DP. The assessment outcomes on '8' open-source projects exhibited that the Seml had outshined '3' DP techniques on majority of the datasets aimed at within-project DP and also cross-project DP. But, the data flow information and semantic information were not captured. Qiao YU et al. [19] presented a feature selection (FS) technique centred upon a similarity measure (SM) aimed at SDP. Every feature subset was examined on a k-nearest neighbor (KNN) design and enumerated by an Area Under Curve (AUC) metric aimed at the classification performance. The experimentations were executed on 11 National Aeronautics and also Space Administration (NASA) datasets and the outcomes exhibited that this methodology performed efficient analogized to the compared FS approaches regarding classification performance. However, they evaluated this protocol just on the KNN design. Kapil Juneja [20] introduced a fuzzy-filtered neuro-fuzzy protocol aimed at predicting the software flaws intended for internal and also external software projects. Fuzzy rules were employed to these procedures aimed at the effective and high-impact features' selection. The approximation was employed to internal version-specific fault prediction and also external software projects examination. This prediction outcome signified that the protocol proposed has yielded a greater accuracy, low error rate, and also noteworthy AUC and GM aimed at inter-project and as well inter-version assessments. However, this system had high computation complexity.

3. Proposed Methodology

This work proposes a ML algorithm, namely DLR-LVQ for SDP and fuzzy rules aimed at predicting the software's quality. This methodology comprising '6' major steps that are: redundant data removal, FE, sampling, normalization, DP, and then software quality prediction. Initially, all the data sets, namely CM1, KC1, KC2, KC3, MC2, PC1, PC3, and also PC4, are collected as of the publicly available source. Next, the post-processing step was applied in all the datasets using HDFC and Map Reduce (MR). The attributes are extracted from the dataset. Then, sampling utilizing the BM-SMOKE procedure is done aimed at the features extracted. Sampled data are normalized to bring all the variables to the identical range with the log-scaling's help. Next, the SDP phase is done utilizing a novel DLR-LVQ algorithm. Finally, utilizing the fuzzy rules, the software's quality is predicted. Fig-1 exhibits the proposed method's block diagram.

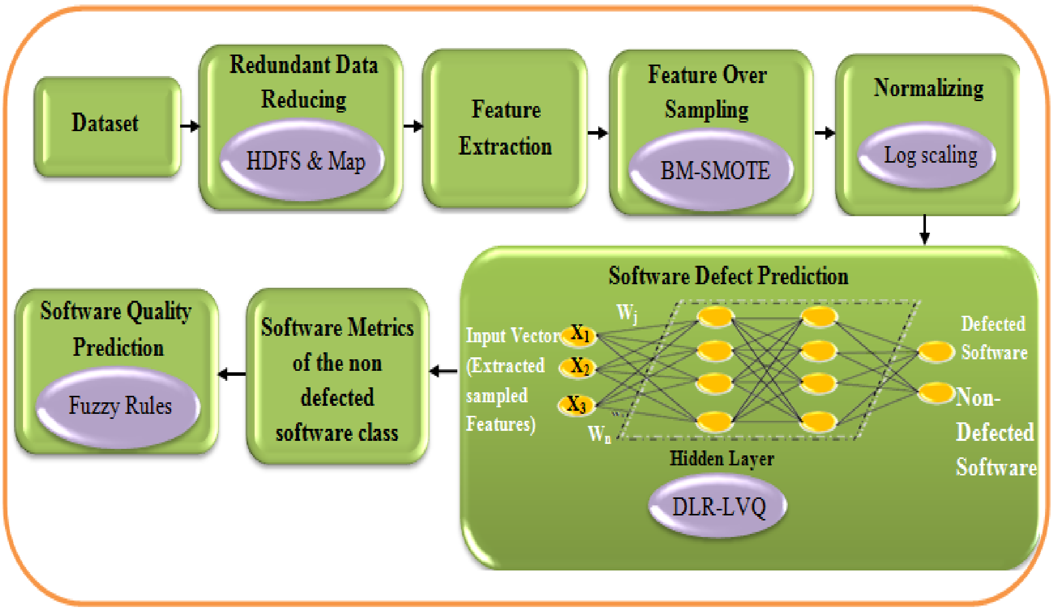


FIGURE 1. Architecture of proposed system

4. Hdfs and Map-Reduce

Herein, the data repeated in the dataset collected is eliminated utilizing the Hadoop Distributed File system (HDFS) and MR function. The HDFS and MR are the Hadoop ecosystem's core components. HDFS is for Distributed storage and MR is for distributed processing. HDFS is diverse as of the ordinarily distributed file systems wherein it comprises a greater storage capacity, and is modelled to stock the huge datasets. The dataset collected is stocked in HDFS. MR is the processing layer of Hadoop. MR programming design is modelled aimed at processing massive data volumes in parallel by portioning the work as independent tasks set. The MR technique encompasses '2' important operations, namely Map and then Reduce. In the Map stage, the HDFS data is processed utilizing the indicated Map function. Data to the Map stage is inputted as of HDFS. The Map function can be modelled aimed at counting the number of distinctive existences of a word within a preprocessed data. The output as of the reduce function is an MR application's output that signifies the collected dataset's reduced dataset. The output as of an MR application is typically stocked inside the HDFS.

5. Feature Extraction

FE is the term for the methodologies that choose and combine variables into features, efficiently decrementing the amount of data that must be processed, whilst yet accurately and fully elucidating the actual data-set. The attributes are extracted as of the reduced dataset, after performing HDFS and MR functions. Feature Oversampling using BM-SMOTE The features extracted are inputted to BM-SMOTE for over-sampling. Oversampling methodologies are utilized aimed at handling the class imbalance issue. SMOTE (Synthetic Minority Oversampling Technique) is amidst the well-known and most popular sampling techniques aimed at addressing the class imbalance learning issues. SMOTE's advantages exhibit that in analogy with the random oversampling methodology, it can administer the over-fitting issue to a huge level. Nevertheless, SMOTE's drawback is that it uses a random function to sample the features, which degrades the classification accuracy. Aimed at synchronously resolving this issue, a grouped SMOTE methodology with Brownian motion (BM-SMOTE) is proffered in this work. The steps are explained follow as,

Step 1: Setting the dataset A, input features $(y = y_1, y_2, \ldots y_n)$ aimed at every $y \in A$, the k-nearest neighbors of y is obtained by calculating the Euclidean distance between y and every other sample in set A.

Step 2: The sampling rate N is fixed regarding the imbalanced proportion. Aimed at every $y \in A$, N examples $(y_1, y_2, \ldots y_n)$ are chosen randomly as of its k-nearest neighbors, and they build the set A_1.

Step 3: Aimed at every example $y_k \in A_1$, $(k = 1,2,3, \ldots N)$ the subsequent formula is utilized to create a novel example: wherein rand (0, 1) signifies the random number betwixt 0 and 1. Wherein, y'_s signifies the sampling data that are freshly synthesized in eqn. (1).

$$y'_s = y + rand(0,1) * (y - y_k) \tag{1}$$

Instead of rand (0, 1) introduces Brownian motion $BM(t)$. The purpose aimed at picking the Brownian motion technique is that its isotropic methodology (utterly direction-independent) increments the discovery proficiency. BM-SMOTE procedure boosts the linear interpolation process, increments the randomness prevalent in the interpolation procedure. The normal sequence can be chosen to be computable that are exhibited in eqn. (2).

$$y'_s = y + BM(t) * (y - y_k) \tag{2}$$

Where

$$BM(t) = H * rand() * K_g \tag{3}$$

$$H = \sqrt{T / N} \tag{4}$$

$$N = 100 * T \tag{5}$$

$$K_g = 1 / H\sqrt{2\prod} exp(-d - f)^2 / 2H^2) \tag{6}$$

Wherein, T implies the motion time-period in seconds; N signifies the number of sudden motions that is in proportion to time; the term d symbolizes the dimension; f implies the features; y'_s signifies the output sampled features and $y'_s = y'_1, y'_2, y'_3, \ldots, y'_n$ aimed at the subsequent iteration.

6. Data Normalization

Herein, the sampled features y'_s are normalized utilizing the log scaling methodology that is offered in eqn. (7). The normalization's goal is to transform features to be on a similar scale. This boosts the model's performance and its training stability.

$$N_d = log(y'_s) \tag{7}$$

Log scaling calculates the log of the values aimed at compressing a comprehensive range to a narrow range. Log scaling is beneficial whilst a handful of values comprise numerous points, whilst most other values comprise limited points. This data distribution is termed the power-law distribution. Log scaling varies the distribution, assisting to boost the linear design performance. Defect Prediction with DLR-LVQ The normalized data N_d are inputted to the DLR-LVQ classifier for identifying the software flaw. Learning Vector Quantization (LVQ) is an Artificial Neural Network type that is as well enthused by the neural systems' biological designs. It can as well deal with the multiclass classification issue. LVQ comprises '2' layers: the Inputted layer and then the Outputted layer. The normal unsupervised training stage's procedure, akin to the Kohonen self-organized feature map, is detailed as, The conventional LVQ comprises the highest learning error issue that direct towards the data's misclassification. So this work establishes a new learning mechanism termed DLR into the conventional LVQ that boosts the LVQ's learning rate and boosts the classification accuracy. Fig-2 explicates DLR-LVQ's architecture. This DLR centred LVQ is termed DLR-LVQ. DLR-LVQ's steps are:

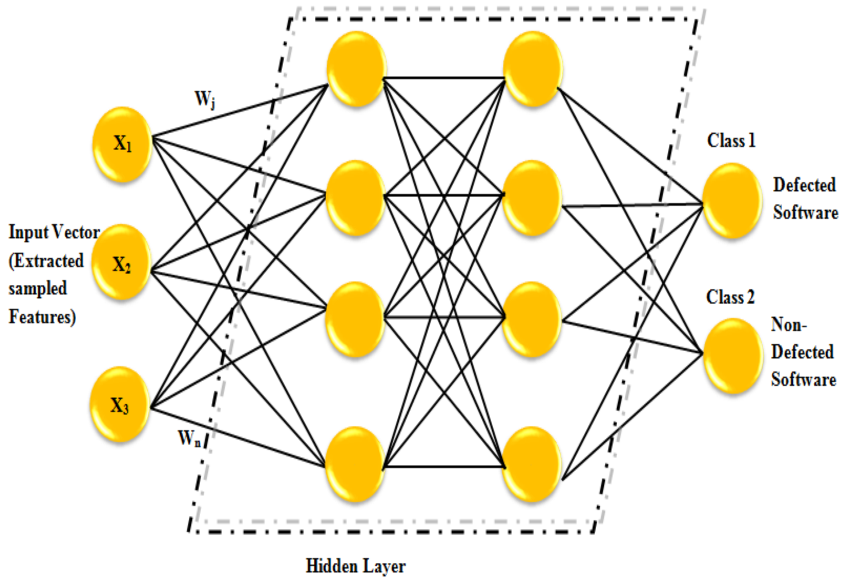


FIGURE 2. Architecture of DLR-LVQ

Step 1: Initialize every input feature $N_d = (N_1, N_2, N_3, ..., N_n)$ that is uniquely attributed to a codebook vector $W_j(0)$; $j = (0,1,2,...,m)$, t =time, at $t = 0$ (m the network's size), with the weights' magnitudes inside the definite range regarding the inputs. The weight vector $(W_j(t))$ is articulated by eqn. (8) as,

$$W_j(t) = [W_{j1}(t), W_{j2}(t), W_{j3}(t)...W_{jn}(t)]^T \tag{8}$$

Herein, n = the vector's dimension, T = maximum epoch number.

Step 2: A sample N is to be taken as of the data that is signified in this case vector.

Step 3: For the input $N(t) = [N_1(t), N_2(t), N_3(t), ...N_n(t)]$, the finest matching (winning) neuron is attained utilizing Euclidean distance as,

$$Y(N) = arg_j\, min\{\| N - W_j \|\} \tag{9}$$

All neurons' synaptic weight vectors are updated utilizing the update formula as,

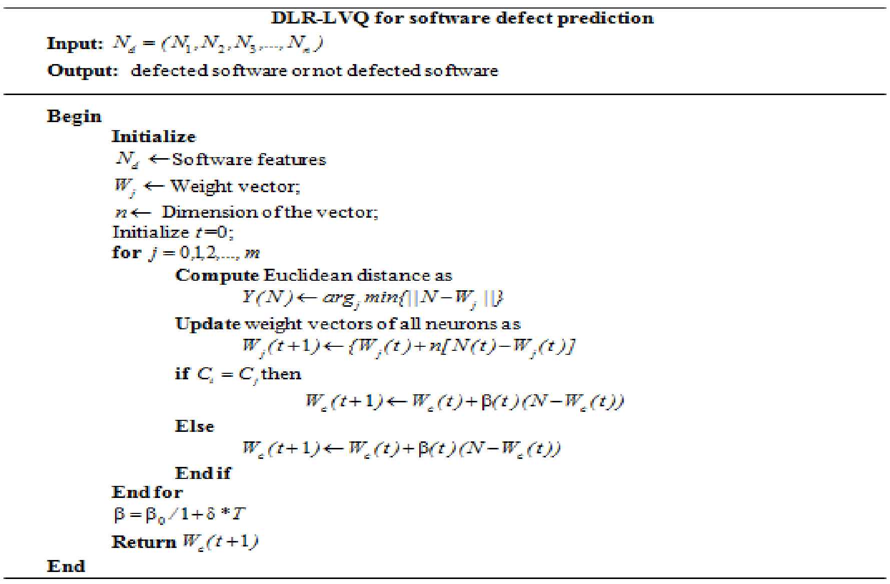
$$\begin{cases} W_j(t+1) = \{W_j(t) + n[N(t) - W_j(t)] & if \quad j \in C \\ W_j(t) & otherwise \end{cases} \tag{10}$$

Herein, n = learning rate and C signifies that winner.

Step 4: Alter the winner unit's weight:

$$\begin{cases} W_c(t+1) = W_c(t) + \beta(t)(N - W_c(t)) & ifC_i = C_j \\ W_c(t+1) = W_c(t) - \beta(t)(N - W_c(t)) & ifC_i \neq C_j \end{cases} \tag{11}$$

Herein, C is once more utilized to signify the winning neuron; C_i implies the class allotted to the winner; C_j symbolizes the class; the input N originating from, $C_i = C_j$ input features are within the class. Hence, the software was defeated by those features. $C_i \neq C_j$ input features aren't in the assigned class. So, the software wasn't defected. β implies the proposed learning rate. Additionally, the subsequent DLR is elected aimed at the earning rate in equation (12). Figure 3 elucidates the proposed DLR-LVQ algorithm's pseudo-code.



```
DLR-LVQ for software defect prediction
Input: $N_d = (N_1, N_2, N_3, ..., N_n)$
Output: defected software or not defected software

Begin
    Initialize
    $N_d$ ← Software features
    $W_j$ ← Weight vector;
    $n$ ← Dimension of the vector;
    Initialize $t$=0;
    for $j = 0,1,2,..., m$
        Compute Euclidean distance as
            $Y(N) \leftarrow arg_j\, min\{||N - W_j||\}$
        Update weight vectors of all neurons as
            $W_j(t+1) \leftarrow \{W_j(t) + n[N(t) - W_j(t)]$
        if $C_i = C_j$ then
                $W_c(t+1) \leftarrow W_c(t) + \beta(t)(N - W_c(t))$
        Else
            $W_c(t+1) \leftarrow W_c(t) + \beta(t)(N - W_c(t))$
        End if
    End for
    $\beta = \beta_0 / 1 + \delta * T$
    Return $W_c(t+1)$
End
```

FIGURE 3. pseudo-code for the proposed DLR-LVQ algorithm

In the DLR methodology, the learning rate decrements with an increment in epochs/iterations. This technique overcomes the error learning rate issue in the conventional technique.

$$\beta = \beta_0 / 1 + \delta * T \tag{12}$$

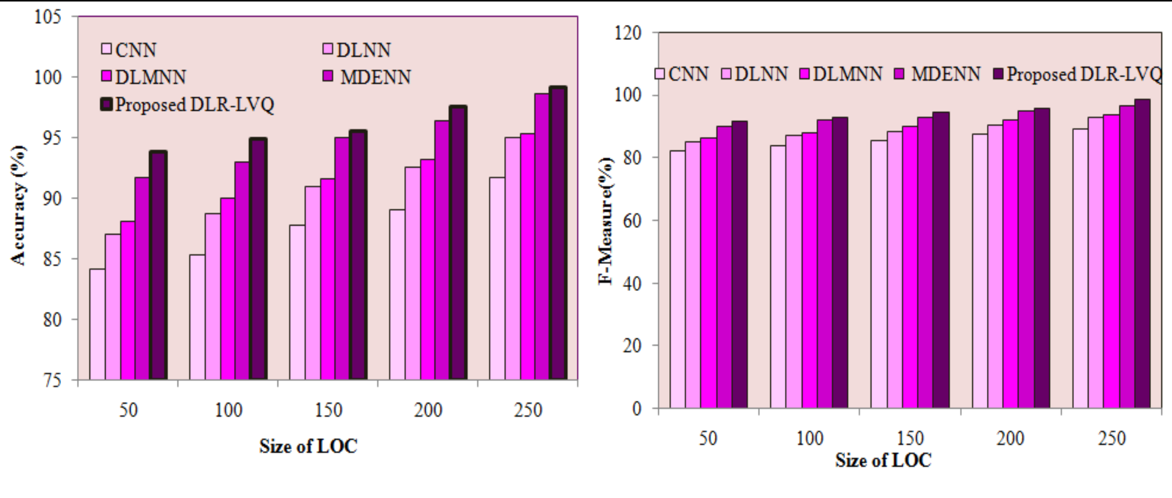
Herein, 0 is the initial learning rate; δ implies the hyper-parameters; β signifies the decay rate and is the learning rate at a certain maximal iteration number (T).Quality Prediction using Fuzzy Rules Pondering all chosen software metrics one at an instance, it is needed to describe numerous rules aimed at forecasting the non-defected software's quality. Therefore, the software metrics engaged from phase to phase are pondered in the design proposed. It has decremented the essential number of rules. Eqn. (13) exhibits the fuzzy rules to forecast the software's quality.

$$\textbf{Rule:}\ \text{IF}\ F_1\ is\ K_1^i,...,F_m\ is\ K_m^i,\ \text{THEN}\ L\ is\ O^i \tag{13}$$

Herein, m implies the number of software input metrics. $K_j^i\,(j = 1,2,3,...,m)$. O^i are the software quality's output labels that indicates the software quality as high, medium or else low. i signifies each fuzzy rule's index, $F \in R^m$ symbolizes the input vector and $L \in R^m$ implies the output variable.

7. Result and Discussion

Herein, the proposed DLR-LVQ classification's outcomes are examined aimed at the SDP. A novel classifier's performance is examined in the performance examination section and the work proposed is applied in JAVA's functioning platform. performance analysis Herein, DLR-LVQ's performance is assessed with diverse existent methodologies. The proposed DLR-LVQ's outcome is analogized with the prevalent techniques, namely Convolutional Neural Network (CNN), Deep Learning Modified Neural Network (DLMNN), Deep Learning Neural Network (DLNN), and Modified Deep Elman Neural Network (MDENN). The outcomes are analogized regarding a few performance metrics centred upon accuracy and F-Measure. The graph is employed to recognize the efficient performance that is exhibited below. Figure 4(a) exhibits the proposed DLR-LVQ's classification accuracy. DLR-LVQ outcomes are analogized with the existent CNN, DLNN, DLMNN, and also MDENN. Aimed at 200 size of Line Of Code (LOC), the proposed DLR-LVQ has 97.53% accuracy, whereas the existing methods CNN, DLNN, DLMNN, and MDENN attains 89.08, 92.55, 93.21, and 96.35. Thus, for all sizes of LOC, the proposed DLR-LVQ attains enhanced classification accuracy analogized to all existent techniques. Figure 4(b) exhibits the proposed DLR-LVQ's F-Measure. The outcomes are analogized with the prevalent methodologies, namely CNN, DLNN, DLMNN, and also MDENN. Aimed at 250 sizes of LOC, the existent CNN, DLNN, DLMNN, and also MDENN attained 89.09, 92.7, 93.89, and 96.78 F-Measure, whereas the proposed DLR-LVQ has 98.78. Thus, for all size of LOC, the proposed DLR-LVQ attains high F-Measure value than other techniques. The results validate the proposed DLR-LVQ's superior performance



(a) (b)

FIGURE 4. Classification acuuracy of the proposed system

8. Conclusion

A novel DP is proposed in this work. The methodologies proposed utilize ‘6’ steps to forecast the defective software and assess the non-defective software’s quality. Thus, the system has decremented the redundant data and increments the classification accuracy, and decrements the learning rate. The DLR-LVQ proposed is analogized with existent CNN, DLNN, DLMNN, and also MDENN concerning a few performance metrics, namely accuracy and also F-Measure. The PROMISE dataset is utilized aimed at examining the proposed methodology’s performance. The proposed DLR-LVQ attains 99.12 accuracy, and also 98.78 F-Measure aimed at 250-size of LOC. The analogized outcomes exhibits the proposed system’s effective classification accuracy. Mining procedure is employed aimed at boosting DP’s accuracy in future.

.

References

[1]. Xuan Huo, and Ming Li, “On cost-effective software defect prediction: Classification or ranking?”, Neurocomputing, vol. 363, pp. 339-350, 2019.

[2]. Hadeel Alsolai, and Marc Roper, “A systematic literature review of machine learning techniques for software maintainability prediction”, Information and Software Technology, vol. 119, pp. 106214, 2020.

[3]. Diego PP Mesquita, Lincoln S. Rocha, João Paulo P. Gomes, and Ajalmar R. Rocha Neto, “Classification with reject option for software defect prediction”, Applied Soft Computing, vol. 49, pp. 1085-1093, 2016. 10.1016/j.asoc.2016.06.023

[4]. Rituraj Singh, Jasmeet Singh, Mehrab Singh Gill, and Ruchika Malhotra, “Transfer Learning Code Vectorizer based Machine Learning Models for Software Defect Prediction”, In IEEE International Conference on Computational Performance Evaluation (ComPE), pp. 497-502, 2020. 10.1109/ComPE49325.2020.9200076

[5]. Zhiguo Ding, and Liudong Xing, “Improved software defect prediction using Pruned Histogram-based isolation forest”, Reliability Engineering & System Safety, vol. 204 107170, 2020. 10.1016/j.ress.2020.107170

[6]. Punitha K and Chitra S, “Software defect prediction using software metrics-A survey”, In proceeding of IEEE International Conference on Information Communication and Embedded Systems (ICICES), pp. 555-558, 2013. 10.1109/ICICES.2013.6508369

[7]. Diana-Lucia Miholca, Gabriela Czibula, and Vlad Tomescu, “COMET: A conceptual coupling based metrics suite for software defect prediction”, Procedia Computer Science, vol. 176, pp 31-40, 2020. 10.1016/j.procs.2020.08.004

[8]. Pandit, Mahesha Bangalore Ramalinga, and Nitin Varma, “A Deep Introduction to AI Based Software Defect Prediction (SDP) and its Current Challenges”, In proceeding of TENCON 2019-2019 IEEE Region 10 Conference (TENCON), pp. 284-290, 2019, 10.1109/TENCON.2019.8929661

[9]. Diana-Lucia Miholca, Gabriela Czibula, and Istvan Gergely Czibula, “A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks”, Information Sciences, vol. 441, pp. 152-170, 2018. 10.1016/j.ins.2018.02.027

[10]. Lakshmi P and Latha Maheswari T, “An effective rank approach to software defect prediction using software metrics”, In proceeding of IEEE 10th International Conference on Intelligent Systems and Control (ISCO), pp. 1-5, 2016. 10.1109/ISCO.2016.7727030

[11]. Shamsul Huda, Sultan Alyahya, Md Mohsin Ali, Shafiq Ahmad, Jemal Abawajy, Hmood Al-Dossari, and John Yearwood, "A framework for software defect prediction and metric selection", IEEE access, vol. 6, pp. 2844-2858, 2017.

[12]. Ning Li, Martin Shepperd, and Yuchen Guo, "A systematic review of unsupervised learning techniques for software defect prediction", Information and Software Technology, vol. 122, pp. 106287, 2020.

[13]. Nalini C and Murali Krishna T, "An Efficient Software Defect Prediction Model Using Neuro Evalution Algorithm based on Genetic Algorithm", In proceeding of IEEE Second International Conference on Inventive Research in Computing Applications (ICIRCA), pp. 135-138., 2020.

[14]. Yan Zhou, Chun Shan, Shiyou Sun, Shengjun Wei, and Sicong Zhang, "Software Defect Prediction Model Based On KPCA-SVM", In 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 1326-1332, 2019.

[15]. Liang Tian, Yue Fan, Lin Li, and Normand Mousseau, "Identifying flow defects in amorphous alloys using machine learning outlier detection methods", Scripta Materialia, vol. 186, pp. 185-189,. 2020.

[16]. Chinnasamy, Sathiyaraj, M. Ramachandran, Kurinjimalar Ramu, and P. Anusuya. "Study on Fuzzy ELECTRE Method with Various Methodologies." REST Journal on Emerging trends in Modelling and Manufacturing 7, no. 4 (2021).

[17]. Kechao Wang, Lin Liu, Chengjun Yuan, and Zhifei Wang, "Software defect prediction model based on LASSO–SVM", Neural Computing and Applications, pp. 1-11, 2020 10.1007/s00521-020-04960-1

[18]. Jinyin Chen, Keke Hu, Yitao Yang, Yi Liu, and Qi Xuan, "Collective transfer learning for defect prediction", Neurocomputing, vol. 416, pp. 103-116, 2020. 10.1016/j.neucom.2018.12.091

[19]. Hongliang Liang, Yue Yu, Lin Jiang, and Zhuosi Xie, "Seml: A semantic lstm model for software defect prediction", IEEE Access, vol. 7, pp. 83812-83824, 2019. 10.1109/ACCESS.2019.2925313

[20]. Qiao Yu, Shu-juan Jiang, Rong-cun Wang, and Hong-yang Wang, "A feature selection approach based on a similarity measure for software defect prediction", Frontiers of Information Technology & Electronic Engineering, vol. 18, no. 11, pp. 1744-1753, 2017.

[21]. Kapil Juneja, "A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation", Applied Soft Computing, vol. 77, pp. 696-713.

.