Indonesian Journal of Electrical Engineering and Computer Science
Vol. 30, No. 3, June 2023, pp. 1534~1547
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v30.i3.pp1534-1547

Gaussian kernelized feature selection and improved multilayer perceptive deep learning classifier for software fault prediction

Sureka Sivavelu, Venkatesh Palanisamy
School of Information Technology and Engineering, Vellore Institute of Technology, Vellore, India

Article Info

Article history:

Received Dec 26, 2022
Revised Jan 5, 2023
Accepted Jan 9, 2023

Keywords:

Classification
Gaussian kernelized locally linear feature embedding
Heaviside step activation function
Kaiser-Meyer-Olkin correlation test
Multilayer perceptive deep learning classifier
Software fault prediction

ABSTRACT

Software fault prediction is the significant process of identifying the errors or defects or faults in a software product. But, accurate and timely detection is the major challenging issue in different existing approaches to predicting software defects. A novel Gaussian linear feature embedding-based statistical test piecewise multilayer perceptive deep learning classifier (GLFE-STPMPDLC) is introduced to improve software fault prediction accuracy and minimize time consumption. First, the input data are collected from the dataset. Next, the software metrics selection is carried out to select the significant metrics using Gaussian kernelized locally linear embedding with lesser software fault prediction. Then classification is carried out by Kaiser Meyer piecewise multilayer perceptive deep learning classifier for software fault prediction. The novelty of Kaiser-Meyer-Olkin (KMO) correlation test analyzes testing and training instances. The innovation of the Heaviside step activation function is applied for analyzing the KMO correlation test results and providing the final software fault prediction results. Finally, accurate fault prediction outcomes are achieved at the output layer with lesser error. Simulation of proposed GLFE-STPMPDLC technique achieves better 5%, 3%, 3% and 3% enhancement of fault prediction accuracy, precision, recall, and f-measure and 13% faster prediction time compared to conventional methods.

Corresponding Author:

Venkatesh Palanisamy
School of Information Technology and Engineering, Vellore Institute of Technology
Vellore, India
Email: venkatesh.palanisamy@vit.ac.in

1. INTRODUCTION

The advance of source code defect forecasting process shows a vital role in developing software quality. In order to classify the defective software modules earlier are used to such defect forecasting and corrected them before the testing process. Software defect prediction is a vital part of software testing. It aids software practitioners to assign their limited resources for testing as well as enhancing software quality by identifying defect constructs in the early stages of enrichment life cycle. A software defect forecasting model classifies the software modules based on metrics. The software defect-forecasting process includes the extraction of metrics and structure of a defect-forecasting model. After that, software defect forecasting models are very helpful for testing engineers to take important conclusion likes accurate and timely detection. In order to, manufacture the software defect forecasting models, novel machine-learning algorithms and deep-learning algorithms are needed for accurately predicting software defects.

Semantic feature learning via defect prediction via stress-based forming limit diagrams (DP-SFLDS) method was developed in [1] to extracting the semantic and structural information using bi-directional long short-term memory (BiLSTM) based neural network. But the complexity of the algorithm was not reduced to further improve the software defect prediction. Lin and Lu [2], a convolutional graph neural network for defect prediction (DP-GCNN) method was performed to classify module as defective or not defective based on this information. However, it failed to analyze the fault-prone software modules for source code files of different sizes with higher accuracy.

The singular spectrum analysis (SSA) combined with a back propagation neural network (BPNN) was introduced in [3] to classifying the software faults. But, computational cost over the majority of data sets was improved. Tameswar *et al.* [4], a hybrid deep neural network model was developed into enhances the prediction of software bugs. However, data pre-processing techniques to potentially was not improve the quality of available public datasets.

Chen *et al.* [5], a nested-stacking and heterogeneous feature selection framework was performed to software defect prediction. However, it failed to manufacture a more intelligent and automated prediction system. A supervised deep learning technique was performed in [6] to software defect detection. But, performance of software defect detection was not improved. Statement-level software defect prediction was developed in [7] using a deep-learning model based on static code features. However, it failed to minimize the error rate of software defect prediction.

A three-stage weighting approach was introduced in [8] for detecting multi-source cross-project software faults. But the performance of defect prediction was not improved the minimum time. Diverse ensemble learning techniques (DELT) was developed in [9] to predict the project defects. The designed techniques increase the complexity of the defect prediction. A hybrid deep neural network was developed in [10] for predicting the software fault based on metaheuristic feature selection. However, it failed to perform the multi-source cross-project defect prediction.

In this section, different approaches have been introduced for defect prediction. Miholca *et al.* [11], a deep learning-based software defect prediction was performed. However, it failed to analyze the computation time of software defect prediction. The Hellinger net model was introduced in [12] for accurate software module defect prediction. But the designed model was not efficient in software defect prediction across larger datasets. Goal-oriented hyper-parameter optimization for scalable training model was introduced in [13] to classify the software defects. However, it failed to enhance the accuracy of software defect prediction.

Liu *et al.* [14], a flow learning-based geodesic cross-project software defect prediction approach was performed. But it was not efficient for software defect prediction and early warning of unknown malware variants. A stacked sparse denoising autoencoder and extreme learning machine were introduced in [15] for detecting software faults. But it failed to optimize the other classifiers for software defect prediction. A software defect prediction using method-call sequences was developed in [16]. But the relevant features were not extracted for cross-project defect prediction.

An improved Elman neural network method was introduced in [17] to improve the performance of defect prediction for time-varying characteristics. But it was not efficient for predicting the defects and solving the practical issues in software development. An attention-based gated recurrent unit long short term memory (GRU-LSTM) model was developed in [18] to predict the possible defective codes in the software. But it failed to predict fault within and between projects. Finding faults using ensemble learners (ELFF) was developed in [19] for predicting the defects in the latest software edition. But the accuracy of deep feature analysis was not developed to develop the performance of fault prediction. A least absolute shrinkage and selection operator support vector machine (LASSO–SVM) model was introduced in [20] to software defect prediction. But, it failed to improve the classification accuracy.

The software defect prediction ensemble approach was introduced in [21] to discover faulty components. But, the best-performing classifier was not identified. Novel variants of the whale optimization algorithm (WOA) were developed in [22] to eradicate unnecessary features. However, the student's performance prediction issues were not handled. Cross version model with data selection (CDS) was analyzed in [23] for choosing relevant data. Enhanced binary moth flame optimization (EBMFO) was introduced in [24] for forecasting software faults. The designed EBMFO failed to boost the accuracy. Fuzzy filtered neuro-fuzzy framework was investigated in [25] with higher accuracy. But, the feature selection stage was not enhanced.

Deep neural networks (DNN) prediction method was introduced in [26] with lesser dimensionality. Novel Feature Selection approach was developed in [27] for selecting vital software metrics. The performance of dissimilar classifiers was not detected. Two-stage data pre-processing method was discussed in [28] with higher prediction performance. But, the time was not minimized. Semi-supervised DFCM clustering was analyzed in [29] to address the class imbalance issue. Dynamic selection of learning techniques was introduced in [30] for forecasting the number of software faults. But, false positive rate was not minimized.

Conventionally, features were physically considered from qualitative or quantitative description of the module or its growth procedure. But, these features disregard both the unmistakable syntax as well as semantics that describe a programming language employed for software expansion as well as which offer extra information on the software modules. As well, conventional methods are illustrated in major problems including minimum software defect prediction, better time consumption, lesser precision, recall and F-measure, and it failed to provide accurate predictions. In order to, motivated by this fact, the novel Gaussian linear feature embedding-based statistical test piecewise multilayer perceptive deep learning classifier (GLFE-STPMPDLC) technique is developed. The strengths of proposed technique are to precisely forecast the software defect, precision, recall, and F-measure as well as reduce the time.

In order to overcome the existing issues, major objectives of research work is contributed as:

- To develop the software fault prediction accuracy, the GLFE-STPMPDLC is comprised two different processes namely feature or metrics selection and classification.
- To reduce the software fault prediction time, Gaussian kernelized locally linear embedding technique is employed in GLFE-STPMPDLC to select the more relevant software metrics from the dataset. The Gaussian kernel function is applied to a locally linear embedding technique to find the relevant metrics based on the nearest neighbor concept. The other irrelevant features are removed from the dataset.
- To increase fault prediction accuracy and minimize the error rate, the Kaiser Meyer piecewise multilayer perceptive deep learning classifier is applied with the selected metrics. The KMO correlation test is applied to a multilayer perceptive deep learning classifier to determine software faults through testing and training data analysis. The Heaviside step activation function is used to evaluate the correlation test and provide the final prediction results. After that, the weight updating of the deep learning classifier minimizes the prediction error.
- Finally, a comprehensive experimental assessment is carried out with a variety of performance parameters to illustrate the improvement of the GLFE-STPMPDLC technique over conventional deep learning methods.

The paper is organized by: In section 2, provides a brief explanation of the proposed GLFE-STPMPDLC technique with a neat architecture diagram. In section 3, describes the Gaussian kernelized feature selection and improved multilayer perceptive deep learning classifier for software fault prediction. In section 4, presents the performance results of the proposed GLFE-STPMPDLC technique and conventional deep learning methods are discussed with different metrics and dataset description. Finally, section 5 is concluding the paper.

2. PROPOSED METHOD

Software fault prediction aims to identify the defective modules of software programs earlier to the testing stage of the development process. The early faults prediction system helps to remove software defects and obtains cost-efficient and better-quality software products. Software faults are logic or execution errors of defects or bugs that cause the system to produce incorrect testing outcomes. As a result, an early forecasting system of software defects is important. The predicting faults-proneness of a module also reduces the time, effort, manpower, and consequently the cost to develop better quality software projects. A module represents a software component or section of a program that includes one or more source codes written in a particular language consisting of subprograms and functions. To predict faults in software modules are performed for wide range of statistical and machine learning models. However, the model performance is vulnerable to irrelevant and redundant features. In addition, the previous model mainly uses data mining techniques, but the accurate prediction performance is still a challenging issue.

Based on this motivation, this paper proposes an improved multilayer perceptive classifier to detect the software fault in source code lines and enhance the software quality. Software quality analysis is a main concern in software testing with relevant code metrics or features. Initially, data set comprises many irrelevant or redundant features, which affects the accuracy of software fault forecasting. As a result, proposed GLFE-STPMPDLC technique is used to selecting a subset of the relevant features to the defects.

Figure 1 illustrates the architecture of the proposed GLFE-STPMPDLC technique which includes two different processes such as feature selection and classification. Let us consider a dataset 'D' is a set of software entities (modules, classes, functions) $M_1, M_2, M_3, \dots M_m$ is a training instance. The software entities are illustrated as numerical vectors and showed by a set of software features (i.e. software metrics). $X_1, X_2, X_3, \dots X_n$. A training data set including both positive and negative samples is used for constructing the software defect predictor to classify instances in order to predict the Defected and non-defected instances. Therefore, the proposed technique improved the accuracy of review or testing.

First, the feature selection also called metric selection is carried out in GLFE-STPMPDLC by selecting only relevant features for minimizing the time complexity of fault prediction. Software metric is quantity of software characteristics to evaluate software performance. In the proposed GLFE-STPMPDLC technique, Gaussian kernelized locally linear feature embedding is applied for a relevant feature or metric selection and removes the other features to enhance the performance of fault prediction with minimum time.

A locally linear feature embedding is a dimensionality reduction technique used to find the relevant feature or metric based on the nearest neighbor concept with the help of the Gaussian kernel function.

Finally, the Kaiser Meyer piecewise multilayer perceptive deep learning classifier is performed in GLFE-STPMPDLC technique for predicting the performance of accurate fault prediction with lesser time by deeply learning the testing and training data by using KMO correlation test. The KMO correlation test is a statistical measure to find out the well-matched results for software fault prediction through the analysis of testing and training data. The Heaviside step activation function is applied to examine the correlation test results. After that, the weights are updated until the algorithm finds the minimum error. The process of proposed GLFE-STPMPDLC technique is briefly described in following sub-sections.

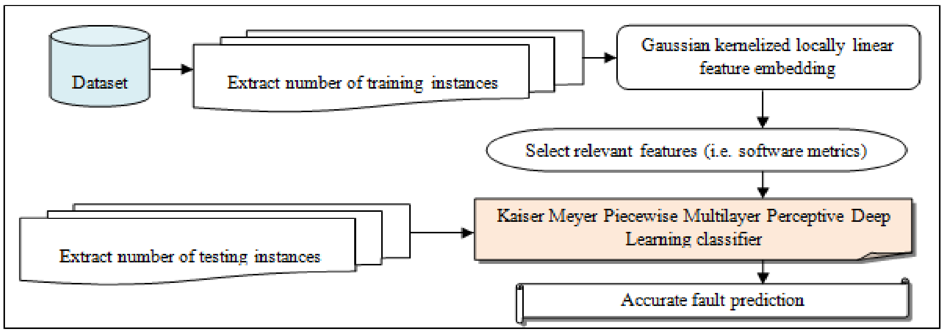
Dataset
Extract number of training instances
Gaussian kernelized locally linear feature embedding
Select relevant features (i.e. software metrics)
Kaiser Meyer Piecewise Multilayer Perceptive Deep Learning classifier
Extract number of testing instances
Accurate fault prediction

Figure 1. Architecture diagram of the proposed GLFE-STPMPDLC technique

3. METHOD DESCRIPTION

3.1. Gaussian kernelized locally linear embedding-based feature selection

Feature selection is a fundamental process of the proposed GLFE-STPMPDLC technique to minimize the complexity of fault prediction. While building a deep learning model for a large dataset, a lot of features are presented and not all these features are significant for every time. These unnecessary features lead to minimizing the overall accuracy of the model and increase its complexity. Therefore, feature selection is a significant process while building a machine-learning model for a large dataset. The important aim of feature selection is to determine the preeminent feasible set of features. The definition software fault prediction is estimating the errors (sometimes called defects) in a software product based on previously defined metrics.

Figure 2 illustrates the architecture diagram of Gaussian kernelized locally linear embedding for selecting the significant features (i.e. software metrics) for predicting the defective or non-defective modules to develop the quality of software products. The training instances are collected from dataset. After the data collection process, the significant features are selected by applying a Gaussian kernelized locally linear embedding technique. The proposed method is a dimensionality reduction technique used for finding a set of the nearest neighbors of each point (i.e. features). The nearest neighbor's points are identified by using Gaussian kernel functions. A Gaussian kernel is a localized similarity measure between two random variables (i.e. two features).

In Figure 2, consider the number of metrics or features $X = \{X_1, X_2, X_3, \dots, X_n\}$ gathered from dataset. Then, Gaussian kernel function is applied for finding nearest neighbor's features in two-dimensional space. Therefore, nearest neighbor's features selection process as shown in (1).

$$N_{ij} = \sum_{i=1}^{n} = \sum_{j=1}^{n} \left[\exp\left[-\frac{1}{2^{V^2}}\left(\left|\left|X_i - \beta X_j\right|\right|^2\right)\right]\right] \tag{1}$$

According to (1), N_{ij} indicates a Gaussian kernel function between the two features X_i and the nearest features 'X_j' and 'β' denotes a weight vector. As a result, the sum of each row of the weight vector is normalized into 'β =1'. The relevant and irrelevant features are identified through the Gaussian kernel as stated (2).

$$N_{ij} = \begin{cases} X_i \sim X_j \text{ ; } neighbor\ features \\ 0 \text{ ; } not\ a\ neighbor\ feature \end{cases} \tag{2}$$

According to (2), $\boldsymbol{N_{ij}}$ indicates an output of a Gaussian kernel locally-linear feature embedding technique. As a result, the nearest neighbor feature is said to be a relevant feature for classification, and the remaining features are said to be irrelevant features. These irrelevant features are removed from the dataset and the relevant features are specified to input of a deep learning classifier for fault prediction. The algorithmic process of the Gaussian kernel locally-linear embedding-based feature selection is given in Algorithm 1.

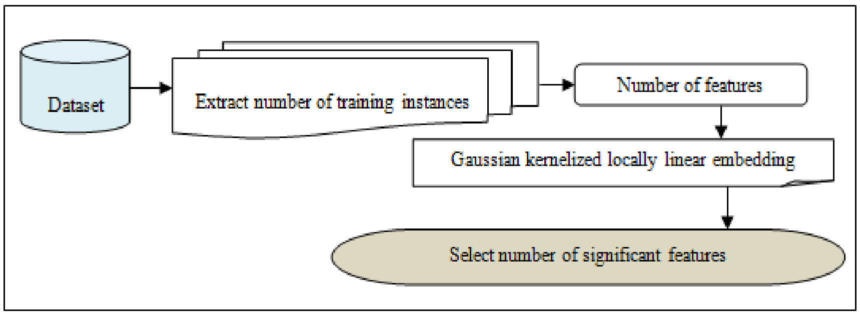
Dataset
Extract number of training instances
Number of features
Gaussian kernelized locally linear embedding
Select number of significant features

Figure 2. Architecture diagram of Gaussian kernelized locally linear embedding-based feature selection

Algorithm 1. Gaussian kernel locally-linear feature embedding
```
Input: Dataset 'D', features or software metrics $X_1, X_2, X_3, \dots X_n$
Output: Select significant features
Begin
1. Collect the features or software metrics $X_1, X_2, X_3, \dots X_n$ from the dataset
2. For each feature 'X'
3. Apply Gaussian kernel to find neighboring features
4. If ($N_{ij} > 0$) then
5. Two features are said to be a neighbor
6. Selected as a relevant feature
7. else
8. Two features are not a neighbor
9. Selected as an irrelevant feature
10. end if
9. Select relevant features
10. Remove the irrelevant features
11. end for
End
```

Algorithm 1, represents the process of the GLFE technique for selecting the significant software metrics. The number of features and data are collected from the dataset. After that, the Gaussian kernel is applied to finding neighboring features. These selected neighboring features are used for fault prediction and other metrics are removed from the dataset to minimize the time complexity.

3.2. Kaiser Meyer piecewise multilayer perceptive deep learning classifier for software fault prediction

After the significant metric selection, the GLFE-STPMPDLC technique performs software fault prediction using Kaiser Meyer piecewise multilayer perceptive deep learning classifier. The proposed classifier takes the input as relevant software metrics also called training instances. Then the proposed deep learning classifier analyzes the training instance with the testing instances to identify the defects in the module of software code for improving the quality of products. The main advantage of the improved multilayer perceptive classifier is to include less feature compatibility and it also handles large sizes of data handling and provides accurate results with minimum error.

Figure 3 illustrates a flow process of classifications for predicting the software fault based on the set of metrics. A multilayer perceptron (MLP) is a type of deep learning feed-forward artificial neural network that comprised numerous layers of perceptrons (with threshold activation). A perceptron is an artificial neuron that includes some numerical inputs along with the weights and a bias. A Multilayer in deep learning classifier including the three layers of nodes namely an input layer, more than one hidden layer, and an output layer. Starting with the input layer, the training, and testing data forward to the hidden layers. This process is the feed-forward propagation. The input and output layers are always single layers, whereas the hidden layer contains numerous sub-layers for analyzing the given training data instances. Each layer generally includes small individual units called artificial neurons or perceptrons or nodes. Input training instances and testing instances are given to the perceptrons and transferred into the other layer from a previous layer's neurons. The connection between the perceptrons or neurons is called a synapse. The model is constructed as shown in Figure 3.

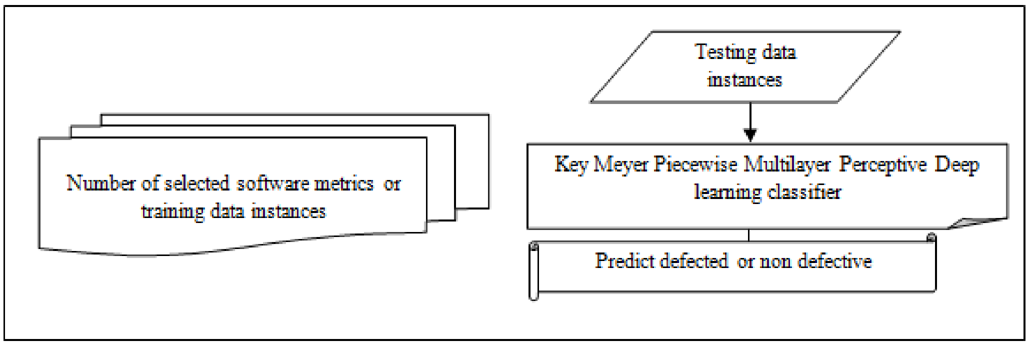
Testing data instances
Number of selected software metrics or training data instances
Key Meyer Piecewise Multilayer Perceptive Deep learning classifier
Predict defected or non defective

Figure 3. Flow process of multilayer perceptive classifer

Figure 4, shows the schematic illustration of deep multilayer perceptron involves the input layer, hidden layers, and output layer. Let us consider the training sets $\{X_i, Z_i\}$ where X_i denotes an input or training data i.e. $A_1, A_{2,}A_3, \dots. A_n$ is a source code and Z_i is a final deep-learning classification result. The artificial neuron in the input layer receives input and applies to weight along with bias as shown in Figure 4.

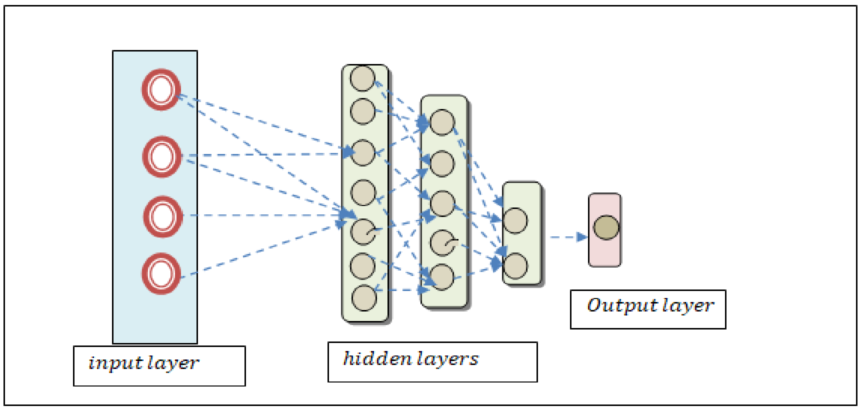

Figure 4. Schematic illustration of the multilayer perceptron

As shown in Figure 5, an artificial neuron receives the weighted sum of input training data with bias as input. Therefore, the activity of the neuron '$R(t)$' is obtained as mentioned in (3).

$$R(t) = \sum_{i=1}^{n} A_i(t) * t \tag{3}$$

Where the activity of neuron in the layer '$R(t)$' indicates that the weighted 'ϑ_i' sum of the input '$A_i(t)$' and add to the bias function 'K' that stored the value is '1'. The weight takes numerical values and controls the level of significance of each input. The main purpose of including a bias term is to transfer the activation function of each perceptron not get a zero value. In the proposed deep learning classifier and the significant parameters are weights and biases. The optimal values for those parameters are determined during the learning process of the deep neural network.

Then the input is transferred into the hidden layers. A random number of hidden layers and sub-layers are positioned with input and output layers. In hidden layer, data analysis is performed by using KMO correlation test with training and testing data as,

$$\rho_{kmo} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|A_i - T_j|^2}{\sum_{i=1}^{n}\sum_{j=1}^{n}|A_i - T_j|^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} V_{ij}^2} \tag{4}$$

where,'ρ_{kmo}' indicates a KMO correlation test coefficient, A_i indicates training data, T_j denotes testing data, V_{ij} denotes the partial correlation between the training and testing data and it is calculated using (5).

$$V_{ij} = \frac{A_i - T_j}{\sqrt{(1-A_i)^2 (1-T_j)^2}} \tag{5}$$

The correlation coefficient ‘ ρ_{kmo}’ is provides output value between 0 and 1. The correlation test coefficient results are given to the Heaviside step activation function and it provides the final classification results.

Figure 6, represent the flow process of Heaviside step activation. The similarity coefficient results are given to the Heaviside step activation for producing the final classification results. The advantage of the activation function is to learn the complex testing and training data. The Heaviside step activation also called the piecewise function provides the output of the best-normalized function with 1 and 0, it makes an accurate software fault prediction. Another advantage is to provide an exact value from the model's output.

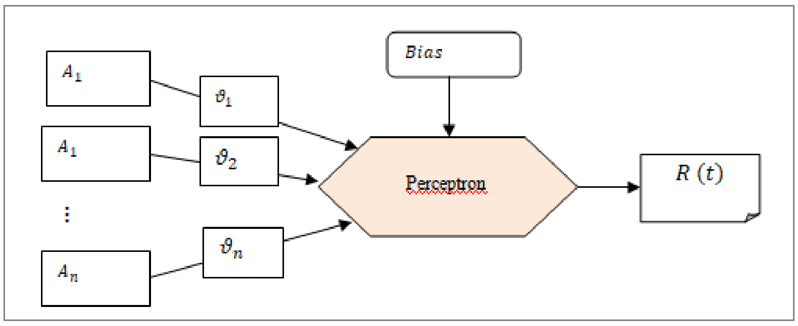

Figure 5. Flow process of perceptron or artificial neuron

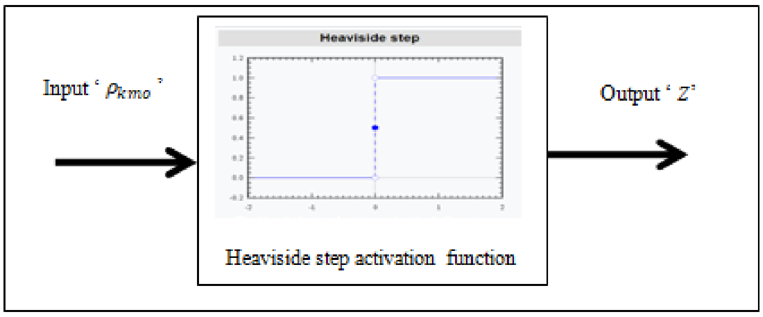

Figure 6. Flow process of Heaviside step activation

As shown in (6), f denotes a Heaviside step activation, ‘ρ_{kmo}’ indicates the correlation test coefficient results. The Heaviside step activation function provides ‘1’ indicating that the software fault or defects are correctly predicted based on the correlation test coefficient between testing and training data. The Heaviside step activation function returns ‘0’ indicating that the non-defects are correctly predicted.

$$f = \begin{cases} \rho_{kmo} = 1; defects \\ \rho_{kmo} = 0; non - \ defects \end{cases} \tag{6}$$

After classification results, the error rate is measured for each learning process is measured as (7).

$$ER = \frac{1}{2}\,(f_a - f_0)^2 \tag{7}$$

Where the error rate ‘ER’ is computed as a squared difference between the actual classification results ‘f_a’ and output produced by the perceptron ‘f_o’. For each iteration, weights get updated as,

$$V_i(t+1) = V_t * \tau \left[\frac{\partial ER}{\partial V_i}\right] \tag{8}$$

where, $\vartheta_i(t+1)$ indicates an updated weight, ϑ_t indicates a current weight, ER denotes an error rate, τ indicates a learning rate ($\tau < 1$), '$\frac{\partial ER}{\partial \vartheta_t}$' indicates a partial derivative of the error 'ER' with respect to current weight 'ϑ_t'. This process is repeated until finding the lesser error. Finally, the software fault prediction result with a minimum error rate is obtained at the output layer. The algorithmic process of the Kaiser Meyer Piecewise multilayer perceptive deep learning classifier is given in Algorithm 2.

Algorithm 2. Kaiser Meyer piecewise multilayer perceptive deep learning classifier

```
Input: Selected relevant features or software metrics and training data
Output: Increase the software fault prediction accuracy
Begin
    1.  Number of training data at the input layer
    2.   For each training data $A_i$
    3.   Assign weight '$\vartheta_i$' and bias '$K$'
    4.   Obtain the perceptron activity at the input layer '$R(t)$'
    5.   end for
    6.   For each training data with testing disease data - [hidden layers]
    7.   Perform Kaiser-Meyer-Olkin (KMO) correlation test '$\rho_{kmo}$'
    8.   Apply Heaviside step activation function '$f$'
    9.   If ( $\rho_{kmo} = 1$ ) then
    10.  $f$ returns '1'
    11.  Correctly predicted as defects or faults
    12.  else
    13.  $f$ returns '0'
    14.  Correctly predicted as non-defects
    15. End if
    16.  For each classification results
    17.  Measure the error rate '$ER$'
    18.  Update the weight '$\vartheta_i(t+1)$'
    19.  Find minimum error
    20.  Obtain the final classification results with minimum error at the output layer
End
```

Algorithm 2, represents the process of software fault prediction with better accuracy and lesser time consumption. Kaiser Meyer piecewise multilayer perceptive deep learning classifier includes many layers to learn the given input software metrics. The selected software metrics are given to the input layer. For each input, the weights and biases are assigned to identify the activity of the neurons. Then the input is transferred into the neuron of the hidden layer. The KMO correlation test is applied to analyze the training data with the testing faults data. After that, the correlation test results are given to the Heaviside step establishment function at hidden layer. Activation function analyzes the correlation test results and provides final classification results either '1 or '0'. If activation function returns '1', then software fault is correctly predicted. If not, activation function returns '0'. Following the classification, error rate for each classification result is measured based on squared difference between the actual and predicted output results. Then, initial weight gets updated and measures the error rate. This process is continuously iterated until the algorithm reaches minimum error. Finally, software fault prediction results are displayed at output layer resulting in improving accuracy and reduced the error rate.

4. PERFORMANCE ANALYSIS AND DISCUSSION

The proposed GLFE-STPMPDLC technique and existing DP-GCNN [1], DP-SFLDS [2] are discussed in Java language using software defect prediction data analysis taken from [31]. This is a PROMISE repository and is publicly available for software engineering. This data set is producing highly precise predictors for defects. The dataset includes 10,885 instances and 22 attributes or features or metrics. The attribute information is given in Table 1. The dataset class value is discrete. The Static code measures are mechanically as well as cheaply gathered. First, the metric selection process is performed using the Gaussian kernelized locally linear embedding technique to select relevant features for defect prediction. With the selected features, the Kaiser Meyer piecewise multilayer perceptive deep learning classifier is applied for predicting the software defects base on the final attributes and it indicates true and false. True indicates software modules have defects and false indicates modules have no defects.

Table 1. Attributes information

S. No	Features or attributes or metrics	Description
1	Loc	Line of code
2	$V(g)$	Cyclomatic complexity
3	$eV(g)$	Essential complexity
4	$iV(g)$	Design complexity
5	n	Total operators + operands
6	v	volume
7	l	Program length
8	d	Difficulty
9	i	Intelligence
10	e	Error approximation
11	b	Effort approximation
12	t	Time estimator
13	locode	Count of Line of code
14	locomment	Count of Line of comment
15	loblank	Count of blank lines
16	locodeandcomment	Count of Line of code and comment
17	Uniq_Op	Unique operators
18	Uniq_Opnd	Unique Operands
19	Total_Op	Total operators
20	Total_Opnd	Total Operands
21	Branch count	Flow graph
22	Defects	{False, True} indicates whether the module has defects or not

4.1. Software fault prediction accuracy

The number of instances that are correctly predicted as defects or not are used to find the software fault prediction accuracy. The prediction accuracy is mathematically as shown in (9).

$$SFPA = \left(\frac{T_p+F_p}{T_p+F_p+T_n+F_n}\right) * 100 \tag{9}$$

Where, $SFPA$ is a Software fault prediction accuracy, T_p indicates a true positive, F_p denotes a false positive, T_n is a true negative, F_n is a false negative. Accuracy is measured in terms of percentage (%).

- true positive: Defects module suitably predicted as defects
- true negative: Non-defects module correctly identified as Non-defects.
- false positive: Non-defects modules incorrectly identified as defects.
- false negative: Defects modules incorrectly identified as Non-defects.

Figure 7, graphical illustration of software fault prediction accuracy for ten various numbers of instances taken from datasets. As shown in Figure 5, the number of instances are collected and taken as input in the 'x ' directions and performance analysis of prediction accuracy of different methods is observed at the 'y' directions. The accuracy of proposed GLFE-STPMPDLC technique is better than other two existing methods. This is because of Kaiser Meyer piecewise multilayer perceptive deep learning classifier. As a result, higher true positives and lesser false negatives, false positives, and true negatives results are obtained. The average of ten comparison results specifies that the proposed GLFE-STPMPDLC technique improves the performance of software fault prediction accuracy by 4% and 5% when compared to existing [1] and [2] respectively.

4.2. Precision

Precision is measured as number of true positives and false positives. Therefore, precision is calculated in (10).

$$P_r = \left(\frac{T_p}{T_p+F_p}\right) * 100 \tag{10}$$

Where, Pr is a Precision, T_p symbolizes the true positive, F_p is a false positive. The Precision is measured in percentage (%).

Figure 8, provide the performance assessment of precision with three methods namely GLFE-STPMPDLC technique and existing [1] and [2]. The performance of precision is improved by 2% and 3% when compared to [1] and [2] respectively. This is due to the application of an improved multilayer perceptive deep learning classifier. The deep learning technique accurately analyzes the selected software metrics with the testing metrics and is classified into two different classes such as defective or non-defective modules. In addition, the false positive rate is minimized by updating the weight and finding a minimal error. This in turn increases the accuracy.

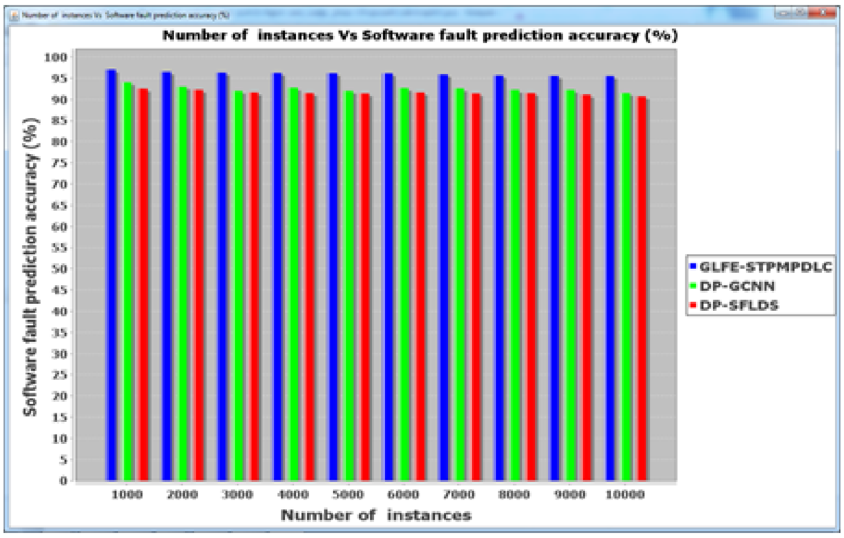

Figure 7. Perfomance comparison of software fault prediction accuaracy

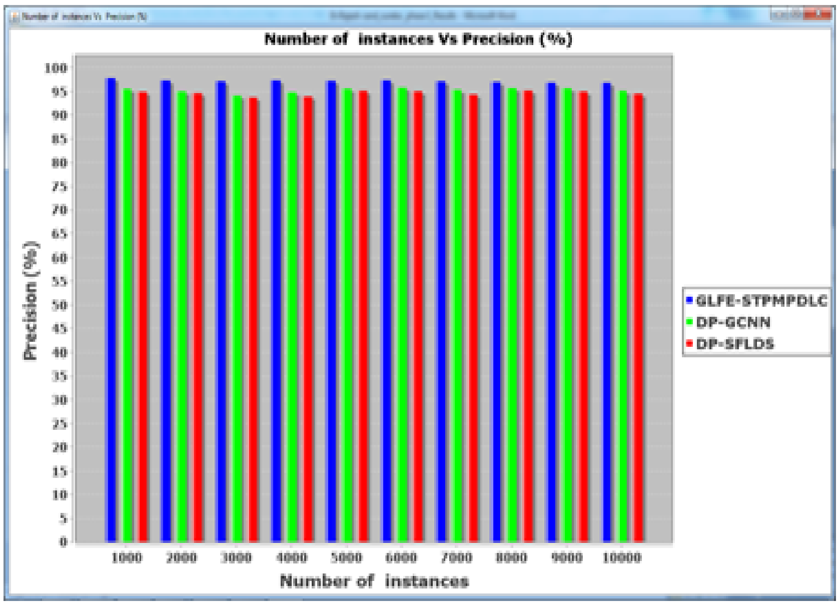

Figure 8. Perfomance comparison of precision

4.3. Impact of recall

The recall is measured to determine the number of true positives as well as false negatives during the prediction. It is formulated as given in (11).

$$R_c = \left(\frac{T_p}{T_p+F_n}\right) * 100 \tag{11}$$

Where R_c is a recall, T_p is a true positive, F_n is a false negative. The recall is measured in percentage (%).

Figure 9 exhibit the overall performance of recall using three different classifiers such as GLFE-STPMPDLC technique and existing DP-GCNN [1], DP-SFLDS [2]. The performance of recall using the GLFE-STPMPDLC technique is improved when compared to existing methods [1], [2] respectively. The performance of recall using GLFE-STPMPDLC is improved by 2% when compared to [1] and 3% when compared to existing [2] respectively.

4.4. Impact of F-measure

F-measure is measured as the mean value of both precisions as well as recall. It is computed using the mathematical formula given in (12).

$$MES_F = \left[2 * \frac{P_r * R_c}{P_r + R_c}\right] * 100 \quad (12)$$

Where MES_F indicates an F-measure computed based on based on precision P_r and recall R_c. F-measure is measured in percentage (%).

Figure 10, represents the performance results of F-measure Vs number of instances taken from the dataset. The F-measure is computed with respect to precision as well as recall. Finally, the overall results of the proposed GLFE-STPMPDLC technique are compared to the results of existing methods. The average of ten comparison results indicates that the overall performance of F-measure is significantly improved by 2% and 3% when compared to existing [1] and [2] respectively.

4.5. Impact of prediction time

Time is defined as amount of time taken by algorithm to accurately predict defective or non-defective software modules. Therefore, overall time consumption of fault prediction is measured as (13).

$$P_t = [n] * t(POI) \quad (13)$$

Where Pt indicates a prediction time, n denotes the number of instances, t denotes a time for predicting one instance (POI). Prediction time is measured in milliseconds (ms).

In Figure 11, shows the graphical design of software fault prediction time using three different methods namely GLFE-STPMPDLC and existing DP-GCNN [1], DP-SFLDS [2]. From the figure, a prediction time gets increased while increasing the number of instances in experiments. However, the GLFE-STPMPDLC decreases the time consumption fault prediction when compared to other existing methods. This is owing to GLFE-STPMPDLC technique performing the software metric selection using the Gaussian kernel locally linear embedding technique. Hence it minimizes the time consumption of software fault prediction. The overall performance is software fault prediction time of GLFE-STPMPDLC technique is minimized by 9% and 16% when compared to existing [1], [2] respectively.

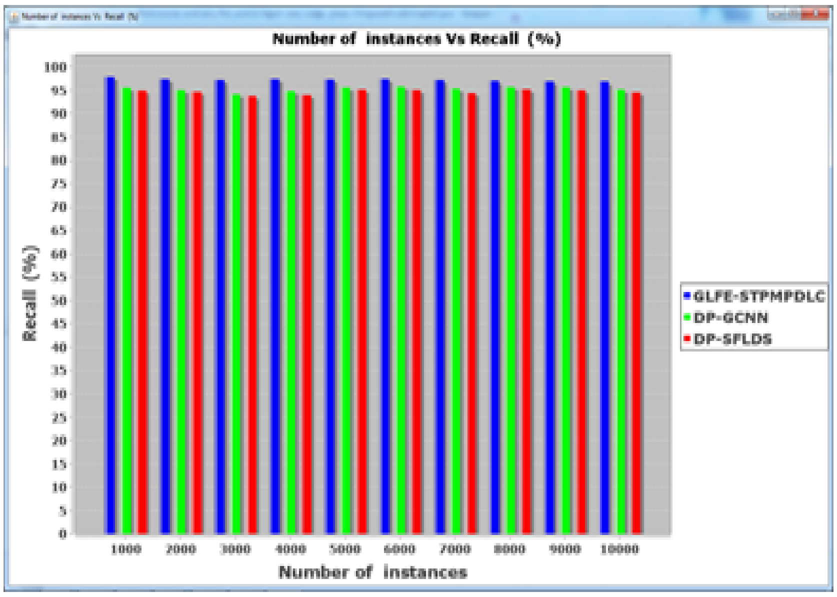

Figure 9. Perforamnce comparison of recall

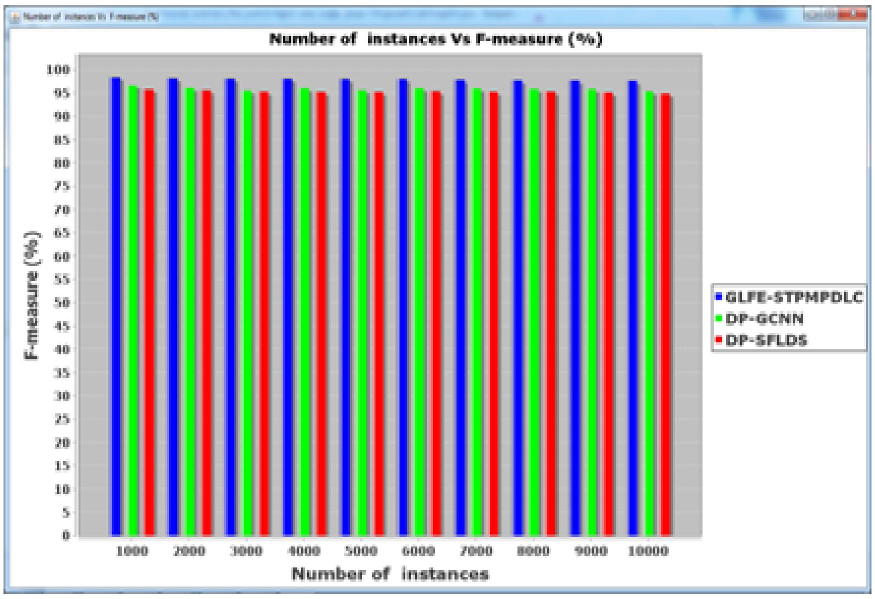

Figure 10. Performance comparison of f-measure

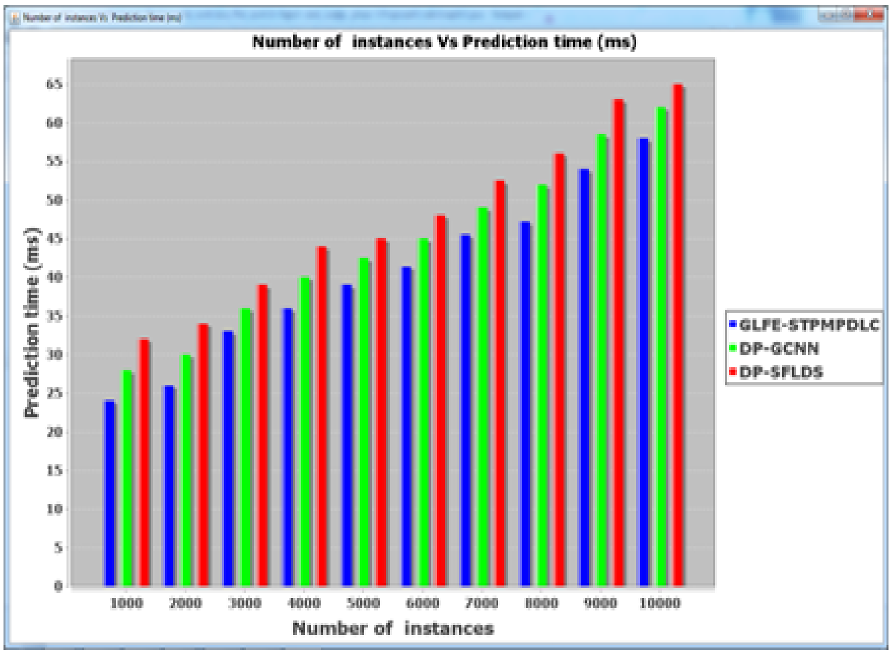

Figure 11. Performance comparison of prediction time

5. CONCLUSION

To overcome this paper, rapid expansion of larger and more complex software systems, quick and accurate detection of potential defects are essential in the source code of the software. In this paper, the GLFE-STPMPDLC technique is introduced. The main aim of proposed work is improved accurate and quick software defect prediction by enhance the prediction accuracy. First, the GLFE-STPMPDLC technique finds the semantic features or metrics by using Gaussian kernel locally linear embedding. Followed by this, a Kaiser Meyer piecewise multilayer perceptive deep learning classifier is developed for deeply learning the training and testing data using the KMO correlation test. The Heaviside step activation function analyzes the correlation outcomes and classifies the defective or non-defective software modules. A comprehensive experimental evaluation is carried out with different performance metrics likes software fault prediction accuracy, precision, recall, F-measure, and prediction time with respect to the number of instances. The overall performance results illustrate the presented GLFE-STPMPDLC technique achieves better prediction accuracy, precision, recall, and f-measure up to 5%, 3%, 3%, and 3% with minimum time by 13% than the existing deep learning methods. The proposed work is further suggested to use a new convolution deep neural network for identifying the software faults by using regression method. In future works should concentrate on evaluating the performance of the proposed methods against different parameters, such as false positive rate and memory consumption.

REFERENCES

[1] L. Sikic, A. S. Kurdija, K. Vladimir, and M. Silic, "Graph neural network for source code defect prediction," *IEEE Access*, vol. 10, pp. 10402–10415, 2022, doi: 10.1109/ACCESS.2022.3144598.

[2] J. Lin and L. Lu, "Semantic feature learning via dual sequences for defect prediction," *IEEE Access*, vol. 9, pp. 13112–13124, 2021, doi: 10.1109/ACCESS.2021.3051957.

[3] S. Kassaymeh, S. Abdullah, M. A. Al-Betar, and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 3365–3378, Jun. 2022, doi: 10.1016/j.jksuci.2021.01.015.

[4] K. Tameswar, G. Suddul, and K. Dookhitram, "A hybrid deep learning approach with genetic and coral reefs metaheuristics for enhanced defect detection in software," *International Journal of Information Management Data Insights*, vol. 2, no. 2, p. 100105, Nov. 2022, doi: 10.1016/j.jjimei.2022.100105.

[5] L. Q. Chen, C. Wang, and S. L. Song, "Software defect prediction based on nested-stacking and heterogeneous feature selection," *Complex and Intelligent Systems*, vol. 8, no. 4, pp. 3333–3348, Aug. 2022, doi: 10.1007/s40747-022-00676-y.

[6] T. Hai, J. Zhou, N. Li, S. K. Jain, S. Agrawal, and I. B. Dhaou, "Cloud-based bug tracking software defects analysis using deep learning," *Journal of Cloud Computing*, vol. 11, no. 1, p. 32, Aug. 2022, doi: 10.1186/s13677-022-00311-8.

[7] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: Statement-level software defect prediction using deep-learning model on static code features," *Expert Systems with Applications*, vol. 147, p. 113156, Jun. 2020, doi: 10.1016/j.eswa.2019.113156.

[8] J. Bai, J. Jia, and L. F. Capretz, "A three-stage transfer learning framework for multi-source cross-project software defect prediction," *Information and Software Technology*, vol. 150, p. 106985, Oct. 2022, doi: 10.1016/j.infsof.2022.106985.

[9] U. S. Bhutamapuram and R. Sadam, "With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 8675–8691, Nov. 2022, doi: 10.1016/j.jksuci.2021.09.010.

[10] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *Journal of Systems and Software*, vol. 180, p. 111026, Oct. 2021, doi: 10.1016/j.jss.2021.111026.

[11] D. L. Miholca, V. I. Tomescu, and G. Czibula, "An in-depth analysis of the software features- impact on the performance of deep learning-based software defect predictors," *IEEE Access*, vol. 10, pp. 64801–64818, 2022, doi: 10.1109/ACCESS.2022.3181995.

[12] T. Chakraborty and A. K. Chakraborty, "Hellinger Net: A hybrid imbalance learning model to improve software defect prediction," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 481–494, Jun. 2021, doi: 10.1109/TR.2020.3020238.

[13] R. Yedida and T. Menzies, "On the value of oversampling for deep learning in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 3103–3116, Aug. 2022, doi: 10.1109/TSE.2021.3079841.

[14] W. Liu, B. Wang, and W. Wang, "Deep learning software defect prediction methods for cloud environments research," *Scientific Programming*, vol. 2021, pp. 1–11, Nov. 2021, doi: 10.1155/2021/2323100.

[15] N. Zhang, S. Ying, K. Zhu, and D. Zhu, "Software defect prediction based on stacked sparse denoising autoencoders and enhanced extreme learning machine," *IET Software*, vol. 16, no. 1, pp. 29–47, Feb. 2022, doi: 10.1049/sfw2.12029.

[16] F. Yang, Y. Huang, H. Xu, P. Xiao, and W. Zheng, "Fine-grained software defect prediction based on the method-call sequence," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1–15, Aug. 2022, doi: 10.1155/2022/4311548.

[17] K. Song, S. K. Lv, D. Hu, and P. He, "Software defect prediction based on elman neural network and cuckoo search algorithm," *Mathematical Problems in Engineering*, vol. 2021, pp. 1–14, Nov. 2021, doi: 10.1155/2021/5954432.

[18] H. S. Munir, S. Ren, M. Mustafa, C. N. Siddique, and S. Qayyum, "Attention based GRU-LSTM for software defect prediction," *PLoS ONE*, vol. 16, no. 3 March, p. e0247444, Mar. 2021, doi: 10.1371/journal.pone.0247444.

[19] E. A. Felix and S. P. Lee, "Predicting the number of defects in a new software version," *PLoS ONE*, vol. 15, no. 3, p. e0229131, Mar. 2020, doi: 10.1371/journal.pone.0229131.

[20] K. Wang, L. Liu, C. Yuan, and Z. Wang, "Software defect prediction model based on LASSO–SVM," *Neural Computing and Applications*, vol. 33, no. 14, pp. 8249–8259, Jul. 2021, doi: 10.1007/s00521-020-04960-1.

[21] S. Huda *et al.*, "An ensemble oversampling model for class imbalance problem in software defect prediction," *IEEE Access*, vol. 6, pp. 24184–24195, 2018, doi: 10.1109/ACCESS.2018.2817572.

[22] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, "Boosted whale optimization algorithm with natural selection operators for software fault prediction," *IEEE Access*, vol. 9, pp. 14239–14258, 2021, doi: 10.1109/ACCESS.2021.3052149.

[23] J. Zhang *et al.*, "CDS: A cross-version software defect prediction model with data selection," *IEEE Access*, vol. 8, pp. 110059–110072, 2020, doi: 10.1109/ACCESS.2020.3001440.

[24] I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, "Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction," *IEEE Access*, vol. 8, pp. 8041–8055, 2020, doi: 10.1109/ACCESS.2020.2964321.

[25] K. Juneja, "A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation," *Applied Soft Computing Journal*, vol. 77, pp. 696–713, Apr. 2019, doi: 10.1016/j.asoc.2019.02.008.

[26] W. Geng, "Cognitive deep neural networks prediction method for software fault tendency module based on Bound Particle Swarm Optimization," *Cognitive Systems Research*, vol. 52, pp. 12–20, Dec. 2018, doi: 10.1016/j.cogsys.2018.06.001.

[27] H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction," *Expert Systems with Applications*, vol. 122, pp. 27–42, May 2019, doi: 10.1016/j.eswa.2018.12.033.

[28] S. Riaz, A. Arshad, and L. Jiao, "Rough noise-filtered easy ensemble for software fault prediction," *IEEE Access*, vol. 6, pp. 46886–46899, 2018, doi: 10.1109/ACCESS.2018.2865383.

[29] A. Arshad, S. Riaz, L. Jiao, and A. Murthy, "Semi-supervised deep fuzzy c-mean clustering for software fault prediction," *IEEE Access*, vol. 6, pp. 25675–25685, 2018, doi: 10.1109/ACCESS.2018.2835304.

[30] S. S. Rathore and S. Kumar, “An approach for the prediction of number of software faults based on the dynamic selection of learning techniques,” *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 216–236, Mar. 2019, doi: 10.1109/TR.2018.2864206.
[31] M. Cevik, “Software defect prediction data analysis,” www.kaggle.com, 2020, https://www.kaggle.com/code/semustafacevik/software-defect-prediction-data-analysis/data (accessed Jan. 6, 2022).

BIOGRAPHIES OF AUTHORS

Prof. Sureka Sivavelu is Assistant Professor (Sr) at School of Information Technology and Engineering, Vellore Institute of Technology, Vellore from 2007 to till date. She holds an MSC(CS) with university rank. She holds her M.Tech. (CSE) in VIT. She used to hold some administrative posts with the School of Information Technology and Engineering. Her Research area is Software Testing. She has supervised more than 20 UG and PG Projects. She can be contacted at email: ssureka@vit.ac.in.

Dr. Venkatesh Palanisamy is currently working as Professor, School of Information Technology and Engineering, Vellore Institute of Technology, India. He received MS (By Research) in 2008 and PhD in 2013 from Anna University, Chennai, India in Computer Science domain. He has over 21 years of teaching experience and currently guiding 2 PhD students. He has published papers in reputed peer-reviewed national and international journals. His research interests are in machine learning, data analytics, Internet of Things and healthcare analytics. Currently he holds the post of Assistant Director (Software Development Cell), involving in software product design and delivery. He can be contacted at email: venkatesh.palanisamy@vit.ac.in.