IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

IEEE RELIABILITY SOCIETY SECTION

Received 19 July 2024, accepted 19 August 2024, date of publication 27 August 2024, date of current version 5 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3450673

RESEARCH ARTICLE

Piecewise Congruence Regressed Indexive Extreme Learning Classifier for Software Fault Prediction

SUREKA SIVAVELU AND VENKATESH PALANISAMY
School of Computer Science Engineering and Information Systems, Vellore Institute of Technology, Vellore 632014, India

Corresponding author: Venkatesh Palanisamy (venkatesh.palanisamy@vit.ac.in)

This work was supported by the School of Computer Science Engineering and Information Systems, Vellore Institute of Technology, Vellore, Tamil Nadu, India.

ABSTRACT Software fault prediction is a significant task in software development to discover faults early. It is the process of developing models that can be used by the software practitioners in the early phases of software development life cycle for detecting faulty constructs such as modules or classes. Therefore, early fault prediction is a critical and challenging task faced by the Project managers. Several kinds of approaches were utilized to predict software faults. In this work we propose a Piecewise Congruence Regressed Indexive Extreme Learning Classifier (PRILEC) to predict the software faults accurately. The process consists of two stages namely feature selection or software metric selection and classification. In feature selection process, congruence correlative piecewise regression method is utilized to extract the most relevant features from the given input dataset. In the next phase, statistical indexive levenberg extreme learning classifier is utilized to classify the fault prediction with better accuracy. The testing and training data analysis in extreme learning classifiers is evaluated using Camargo's statistical index. Hardlimit activation function is utilized to identify the faulty or non-faulty software code. The least square problem can be minimized using Levenberg–Marquardt algorithm and this algorithm can obtain the better classification results. The performance of the proposed approach is evaluated using software fault prediction data analysis dataset. The evaluation metrics such as precision, recall, F-measure, and specificity were used to assess the performance of the proposed algorithm. It was observed that compared with the state-of-the-art traditional methods (Linear regression), proposed technique increases data accuracy of software fault prediction. The system reduces the fault detection time by 4%, 2%, 2%, 2%, 29% and 21% respectively.

INDEX TERMS Software fault prediction, congruence coefficient piecewise regression, statistical indexive Levenberg extreme learning classifier, Camargo's statistical index, Hardlimit activation function.

I. INTRODUCTION

Software fault prediction models play a significant role in enhancing software quality. The software module features are extracted using Prediction algorithm. Traditionally, the features are designed from the qualitative or quantitative description of the module or its development process. However, the model performance is vulnerable to irrelevant and redundant metrics [1]. Conventional software fault prediction was based on features for discovering the fault code. The faults minimize the software quality and enhance its development cost. But, developing robust fault prediction is a challenging task in software prediction. To address these challenges, the PRILEC algorithm is developed.

For predicting the fault or non-fault software modules, a convolutional graph neural network for fault prediction (DP-GCNN) approach was developed [1]. However, the fault-prone software modules for source code with different sizes were not predicted. To address the challenge of predicting the software modules when considering different code sizes, a statistical indexive Levenberg extreme learning classifier is proposed with the inclusion of Camargo's index,

The associate editor coordinating the review of this manuscript and approving it for publication was Liudong Xing.

Hardlimit activation function, and Levenberg–Marquardt algorithm. The presence or absence of the disease is correctly identified with the Hardlimit activation function. Camargo's index is used for analyzing the testing and training data. The Levenberg–Marquardt algorithm reduce the error in the system.

Software faults were classified using a Deep Neural Network Defect Prediction (DNN-DP) model in [2]. But the time complexity was not minimized. To address the challenge of reducing time complexity, the congruence correlative piecewise regression technique is introduced. This feature selection technique is applied to pick relevant features and irrelevant features via congruence correlative.

The accuracy performance was enhanced by the cross-project fault prediction framework in [3]. Nevertheless, the semantic features were not extracted for improving the software fault prediction. A gated hierarchical long short-term memory network (GH-LSTM) was introduced in [4] for software fault prediction. However, the accurate prediction was not performed.

For software fault prediction, Nested-Stacking and heterogeneous feature selection were developed [5]. However, intelligent and automated prediction system was not focused. A Neural Network and Feature Selection methods were developed in [6] for Software fault Prediction. However, the designed method failed to predict the software fault with higher performance. A hybrid Deep Neural Network model was introduced in [7] to improve the prediction of software bugs. However, it failed to perform data pre-processing techniques to potentially improve the quality of available public datasets.

With higher recall, an Anomaly Detection Model Based on BiGAN was developed [8] for Software Defect Prediction. However, the anomaly detection method failed to solve the software fault prediction. For predicting the possible fault code of software modules, an attention-based GRU-LSTM was developed [9]. However, the designed model failed to extract more relevant features to develop the performance of fault prediction. The flexibility of the fault prediction was enhanced by the improved Elman neural network method [10]. However the prediction level of severity of software fault related to issues was not solved in software development.

An artificial neural network based prediction model was developed in [11] for software faults. The Deep Learning algorithm analysis the features of the dataset in depth. However, it failed to potentially improve the quality of available public datasets [12].

ReliefF-based clustering (RFC) method was developed in [13] based on the correlation between features to improve the performance of software fault prediction. However, it failed to focus on the redundant features of high-dimensional datasets. A new imbalanced ensemble learning was introduced in [14] for software fault prediction. But the time consumption of software fault prediction was not minimized efficiently. A Semantic Feature Learning via Dual Sequences (SFLDS) was developed in [15] for fault prediction. However, the accuracy of software fault prediction was not improved.

A. OBJECTIVE OF THE PAPER

The objective of the paper is to increase the prediction of software fault in accurate manner. To select the relevant software metrics in software fault prediction, a Congruence correlative piecewise regression based Feature Selection model is developed. Statistical indexive Levenberg extreme learning classifier is utilized to identify the fault or non-fault software code. The errors are minimized using Levenberg–Marquardt algorithm.

B. CONTRIBUTIONS OF THE WORK

The major contributions of PRILEC are listed below.

- Proposed PRILEC is introduced for enhancing software fault prediction analysis based on feature selection and classification.
- PRILEC uses Congruence correlative piecewise regression for performing feature selection. The congruence correlation coefficient is estimated for the similarity between two features. Based on the similarity value, relevant and irrelevant features are selected. In this way, time consumption of the software fault prediction is minimized.
- Statistical indexive Levenberg extreme learning classifier is developed in PRILEC method for accurate fault or non- fault software code. The novelty of Camargo's index is used to discover the similarity between the testing and training data. The novelty of Hardlimit activation function is applied for finding software fault. Levenberg–Marquardt algorithm is developed for minimizing the error and obtaining the final better classification results.
- The simulation result of the proposed PRILEC method achieves better software fault prediction when compared to prediction methods by using JAVA in the Software Defect Prediction Data Analysis dataset.

C. STRUCTURE OF THE PAPER

The paper is arranged into different sections as follows. Section II discusses the related works carried out in software fault prediction. Section III deliberates the implementation of the proposed PRILEC method. The detailed experimental setup and dataset description are presented in section IV. Performance results are discussed in Section V. Finally, Section VI provides the concluding remarks.

II. RELATED WORKS

A Federated Transfer Learning via Knowledge Distillation (FTLKD) approach was introduced in [16]. But it failed to improve the performance of heterogeneous fault prediction models. A new framework was developed in [17] based on

significant conditional dependency to detect software metrics. But the Just-In-Time fault prediction was not performed. An enhanced metaheuristic feature selection and a hybrid deep neural network were developed in [18] for software fault prediction. But the multi-source cross project or cross-version fault prediction was not performed.

A Collaborative Filtering based source Projects Selection (CFPS) approach was developed in [19] for cross project fault prediction. However, it failed to achieve better prediction performance. A bidirectional gated recurrent unit (BiGRU) and an attention mechanism were introduced in [20] for software fault detection.

Several feature selection and sampling methods were investigated in [21] with higher prediction accuracy for software quality. Fault identification efficiency was increased in [22] by the Least Square Support Vector Machine (LSSVM). Extreme learning machine was investigated in [23] based on the cost and efficiency via different kernel methods. Unrelated metrics were employed with nine feature selection techniques. The greatest set of source code metrics was chosen. Bayesian logistic regression was introduced in [24] to handle prediction issues. ANN and ensemble methods were discussed in [25] for minimizing immaterial features with maximum performance of the fault prediction. Fault prediction efficiency was estimated in [26] with a cost evaluation framework. Ensemble models were analyzed in [27] for optimizing testing resource allocation. Several machine learning techniques such as regression, Bayesian network, random forest, neural network, and naïve Bayes were developed in [28] for fault prediction. The machine-learning techniques used in [29] for achieving higher prediction accuracy. But it failed to consider performance of deep learning algorithms.

Effective feature selection is crucial for building robust fault prediction models, as it reduces dimensionality and enhances the predictive power of classifiers. Recent studies have explored various feature selection techniques to optimize model performance. The author [30] proposed a novel Congruence Correlative Piecewise Regression-based Feature Selection (CCPR-FS) model that leverages statistical correlations between features and fault occurrences. This method has shown to improve the accuracy of fault prediction by selecting the most relevant features while discarding the redundant ones. Regression models, particularly those employing piecewise regression, have been widely adopted for software fault prediction due to their ability to model complex, nonlinear relationships between software metrics and faults. The author [31] extended traditional regression approaches by introducing a piecewise regression model that better captures the non-linearities in large-scale software datasets, leading to improved prediction accuracy.

Extreme Learning Machines (ELMs) have been recognized for their speed and efficiency in classification tasks, including software fault prediction. The author [32] integrated ELM with a Statistical Indexive Levenberg (SIL) classifier to enhance the identification of fault-prone modules. The combination of ELM's rapid learning capabilities with SIL's statistical rigor has been shown to yield superior prediction results, outperforming traditional machine learning methods. The Levenberg–Marquardt (LM) algorithm is a well-established method for minimizing the error in nonlinear models, making it an ideal choice for optimizing software fault prediction models. The author [33] demonstrated the efficacy of the LM algorithm in reducing prediction errors in neural network-based fault prediction models. By fine-tuning model parameters, the LM algorithm significantly enhances the reliability and accuracy of fault prediction, as evidenced in recent empirical studies.

III. METHODOLOGY

A Software fault prediction is used to help the developers in predicting fault -prone parts prior to the testing stage. As the size and complication of codes continues to increase, software quality assurance becomes more and more essential. The main approach to improve software quality is software fault prediction in the field of software engineering. Software fault prediction helps to discover the fault-prone modules in software, which helps organizations to distribute the minimum resources and reduce the workload of software code testing. Based on the motivation, a novel technique called **P**iecewise Congruence **R**egressed **I**ndexive **E**xtreme **L**earning **C**lassifier (PRILEC) is introduced in this section for accurate software fault prediction with minimum error as well as time consumption. The proposed PRILEC is designed with Congruence Correlative Piecewise Regression and Statistical Indexive Levenberg Extreme Learning Classifier. Congruence Correlative Piecewise Regression is utilized for performing feature selection to pick significant features and eradicate immaterial features. Next, the Statistical Indexive Levenberg Extreme Learning Classifier is developed for detecting fault or non-fault products. As a result, the software fault prediction is accurately predicted with higher accuracy and less time.

Figure 1 depicts the architecture of the proposed PRILEC that includes two major processes namely ***feature selection***, and ***classification*** At first, the dataset consists of several software features (i.e. software metrics) $X_1, X_2, X_3, \ldots X_n$ and set of software entities (modules, classes, functions) $M_1, M_2, M_3, \ldots M_m$ denoting training instances.

First, the feature selection or metric selection process is carried out to choose the relevant features for efficient software fault prediction. The proposed PRILEC uses a Congruence Correlative Piecewise Regression for relevant feature selection and removal of redundant metrics. Congruence correlative piecewise regression is a machine learning technique to identify the relevant features with help of correlation measure between the features to enhance the software fault prediction with minimum time complexity.

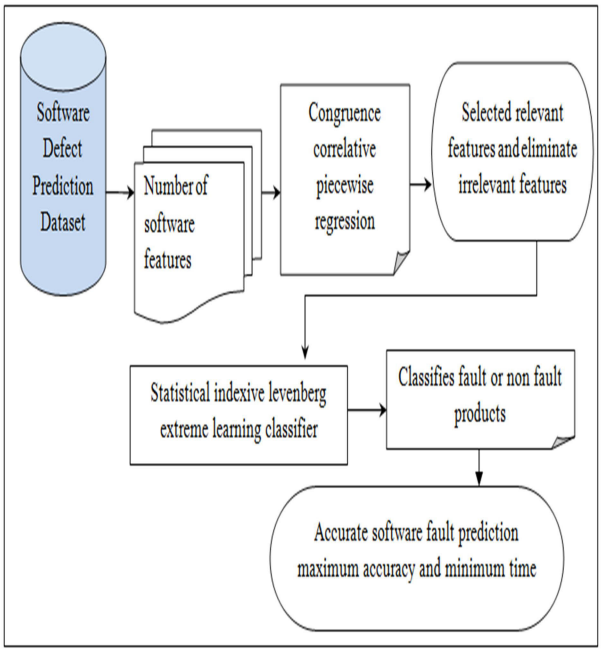

FIGURE 1. Architecture of proposed PRILEC.

Secondly, the Statistical Indexive Levenberg Extreme Learning Classifier is developed to enhance the performance of software fault prediction with higher accuracy and minimum error by estimating the testing and training parameters using Camargo's statistical index. Then the Hardlimit activation function is applied for analyzing the similarity value and predicting the software faults. Finally, the Levenberg–Marquardt algorithm is applied to minimize the error of the Extreme Learning Classifier. These two different processes of the proposed PRILEC are described in the following subsections.

A. CONGRUENCE CORRELATIVE PIECEWISE REGRESSION BASED FEATURE SELECTION

The proposed PRILEC first performs the feature or metric selection after the data collection process. Existing feature selection concepts are employed to select features. The handling dimensionality of the data is a challenging task. A large amount of storage requirements is employed to select features. If the dimensionality of the input dataset increases, any machine learning algorithm and model becomes more difficult. Also, the feature selection performance becomes poor. On the contrary to the conventional feature selection methods, the congruence correlative piecewise regression technique is introduced to perform feature selection. The number of software metrics or features is taken as input from dataset. With considered input metrics, the congruence correlative coefficient is measured among features. The measured value of the coefficient is compared with the threshold for selecting features. The considered dataset includes some unnecessary features that cause quality issues. For removing irrelevant features, a congruence correlative piecewise regression process is utilized. In this way, the dimensionality of the dataset was reduced with higher performance.

Regression is used for the decision making process for measuring the correlation between a dependent variable and one or more independent variables. The developed piecewise regression method is a regression analysis process. It is also called broken-stick regression. The Piecewise regression method makes it easier to support vector machines for identifying important features. During the regression process, independent features are separated into various segments by determining the relationship between two different variables. In multivariate statistics, the congruence coefficient is an index of the similarity between two features. The congruence coefficient is also called a monotonicity coefficient.

Here, a threshold value is set and compared for selecting significant relevant features. Piecewise works for finding the best set of breakpoints that reduces the error. Therefore, the proposed technique uses the congruence correlative piecewise regression technique for selecting significant features and removing irrelevant features with minimum time.

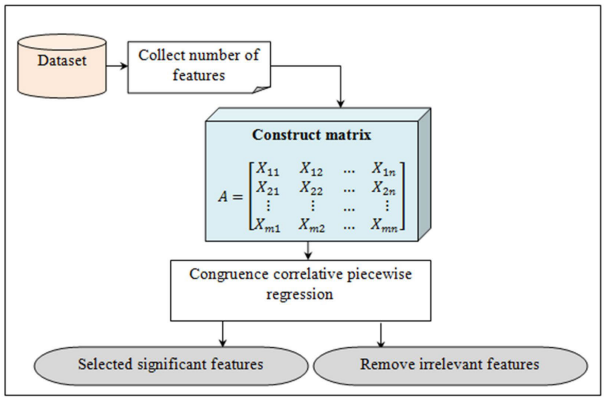

FIGURE 2. Flow diagram of congruence correlative piecewise regression based feature selection.

Figure 2 shows the Flow diagram of Congruence Correlative Piecewise Regression Feature Selection. With the dataset provided as input, the objective of the Congruence coefficient correlative piecewise regression is to select the significant feature in a computationally efficient manner. The congruence coefficient is defined when both matrices have the same number of rows and columns. These matrices can store factor scores (for observations). To start with the raw input dataset '***DS***' and formulated in the form of a matrix as given below.

$$A = \left[X_{11}\, X_{12}\, \dots\, X_{1n}\, X_{21}\, X_{22}\, \dots\, X_{2n} \vdots\vdots \dots \vdots X_{m1}\, X_{m2} \dots\, X_{mn} \right],\ m = rows,\ n = columns \tag{1}$$

From the above input matrix formulation as given in (1), 'n' column features are present with overall sample instances of 'm' row respectively.

With the above set of features matrix, first relevant features or software metrics for software fault detection is performed. To acquire the relevant features from the raw dataset, the regression function analyzes the relationship between the columns of the features in the matrix with the help of the Congruence correlation coefficient [34]. It is an index of the similarity between factors. Let X_i and X_j be column vectors of features with two samples. The formula for the congruence coefficient is given below:

$$\rho_c = \frac{\sum X_i X_j}{\sqrt{\sum X_i^2 \sum X_j^2}} \tag{2}$$

where, ρ_c denotes a congruence coefficient which returns the similarity output ranges from 0 to 1. The minimization of the above equation is then formulated with piecewise regression by setting the threshold as given below.

$$R = \{\rho_c > T, \textit{relevant features}; \rho_c < T, \textit{irrelevant features}\} \tag{3}$$

From the above equation (3), 'R'' represents the outcomes of the piecewise regression. The value of the coefficient greater than threshold 'T' indicates a perfect similarity and is considered as a relevant feature whereas the value of a coefficient lesser than threshold 'T' indicates an irrelevant feature and it is discarded for further processing.

With the obtained features selected, the overall algorithm representation of Congruence Coefficient Correlative Piecewise Regression is given below.

Algorithm 1 Congruence Correlative Piecewise Regression Based Feature Selection

Input: Dataset 'DS', metrics or features $X = \{X_1, X_2, X_3, \ldots X_n\}$

Output: Significant feature selection

Begin
1. **For** each Dataset 'DS' with Features 'X'
2. Formulate input vector matrix 'A' as given in (1)
3. Measure congruence correlation coefficient between the feature as given in (2)
4. Formulate piecewise regression function to obtain relevant features as given in (3)
5. **If** $(\rho_c' > T)$ **then**
6. **Return** relevant features
7. **else if** '$\rho' < 0.5$' **then**
9. **Return ir**relevant features
10. **End if**
11. **Select relevant** features
12. **Remove irrelevant** features
13. **End for**

End

The raw dataset is subjected to congruence correlative piecewise regression. First, with the data acquired in the form of input matrix, relationship between the features is determined by applying congruence correlation coefficient. Second, based on the correlation, piecewise regression is performed to determine the significant features by setting the threshold. Finally, highly correlated features are selected for software fault prediction in an accurate and timely manner. These selected significant features are utilized for classification.

Choosing the Congruence Correlation Coefficient (CCC) for feature selection in tasks like software fault prediction or other machine learning applications can be advantageous for several reasons. The CCC offers unique benefits that make it a valuable tool. Unlike traditional correlation coefficients (e.g., Pearson or Spearman), which primarily measure the strength and direction of a linear relationship, CCC is designed to assess the degree of agreement between two variables. This means that CCC considers both precision and accuracy. CCC considers both the correlation and the potential bias between the two variables, providing a more comprehensive measure of similarity. This is particularly important when the data has inherent variability or when there is a need to account for potential biases in the measurement process.

CCC can be applied to both continuous and categorical data, making it versatile across different types of datasets. This is particularly beneficial in feature selection when the dataset includes a mix of different types of features. CCC is less sensitive to differences in the scale of the two variables being compared. It adjusts for scale differences, which is beneficial when dealing with features that are measured on different scales. By combining the benefits of correlation analysis with an assessment of agreement, CCC offers a more holistic view of feature importance. It evaluates not only how strongly a feature is related to the target variable but also how closely it matches the target in terms of agreement. Features selected based on CCC are likely to generalize better to new, unseen data because CCC emphasizes agreement and consistency, not just correlation. CCC can be particularly effective when the distribution of the feature is aligned with that of the target variable. It ensures that selected features not only correlate but also conform to the distributional properties of the target, leading to better predictive performance.

Congruence Correlation Coefficient (CCC) can be highly sensitive to outliers in the dataset. Outliers can disproportionately affect the correlation coefficient, leading to misleading conclusions about the relationship between variables. CCC primarily measures the linear relationship between variables. If the relationship between features and the target variable is nonlinear, CCC may not accurately capture this relationship. Calculating CCC for many features can be computationally demanding, particularly in high-dimensional datasets. The need to compute pairwise correlations for all features increases the computational load.

Statistical Indexive Levenberg Extreme Learning Classifier based Software Fault Prediction: With the selected

significant features, software fault prediction is a major barrier to attain effective software testing resource allocation. Hence, it becomes necessary to design effective models for software fault prediction at an early stage. In this work, a new model Statistical Indexive Levenberg Extreme Learning Classifier for software fault prediction is designed. On the contrary to the conventional deep learning algorithm, extreme learning machines are feed-forward neural networks having straightforward solutions that do not require iteration. Such a solution is also linear and very fast to compute the analysis and provide a linear output. The main advantage is the strong generalization ability and fast training speed. Therefore, the proposed technique uses the statistical indexive levenberg extreme learning classifier technique for accurate fault prediction with minimum time since it avoids the iterative training process.

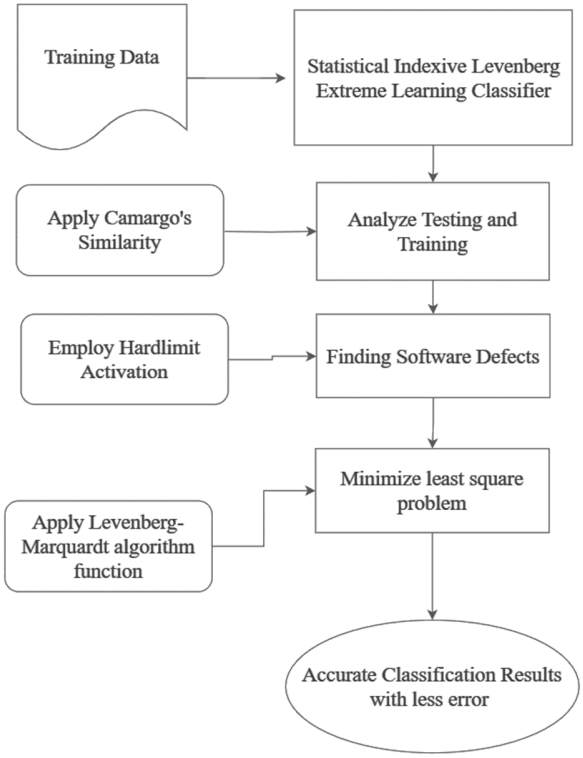

FIGURE 3. Structure of statistical indexive levenberg extreme learning classifier model.

Figure 3 illustrates the flow process of Ensemble learning based classification for software fault prediction. The input training data is given to the proposed classifier. Then the testing and training data analysis is carried out with the help of the statistical index functions called Camargo's index. With the output of the statistical index, the Hardlimit activation function is applied to validate the statistical index results. Finally, the error rate is minimized by applying a Levenberg–Marquardt algorithm for solving the least square problem. The basic structure of the statistical indexive levenberg extreme learning classifier is given below:

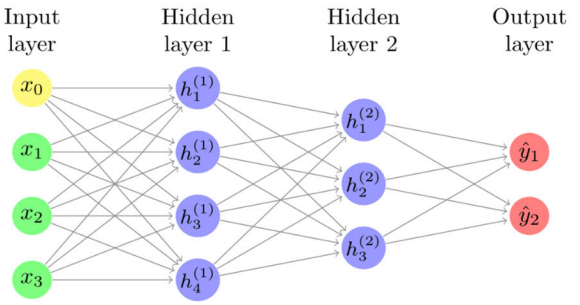

FIGURE 4. Structural of statistical indexive levenberg extreme learning classifier.

Figure 4 depicts the structure of Extreme Learning classifier a type of feed-forward neural network used for classification and feature learning with a single layer or multiple layers of hidden nodes 'h'. The ELM structure includes input layer, multiple hidden layers, and output layer. Compared with the conventional deep neural network, it has two characteristics such as hidden layer parameters (i.e., input weights and the biases) are randomly initialized and another one is output layer weights solved as the least squares problem. As shown in the above figure, let us consider that the training set $\{D, Y\}$ where D denotes a training data with the selected features '$\{X_1, X_2, \ldots, X_k\}$' and a label or output 'Y' representing its category which belongs to the dissimilar classes.

As shown in Figure 2, an Extreme Learning machines classifier receives 'n' training data as input ($D_i = D_1, D_2, \ldots, D_n$), and classifier randomly set a weight '$\alpha_1, \alpha_2, \ldots, \alpha_m$' associated with the weight in the input layer and added bias 'k' in the hidden layer that stored the value is '1'. The input layer offers only training data but it does not perform any computations, whereas the output layer is linear with no transformation function and no bias. The input weights are fixed and have a straightforward solution that does not require a different iteration process.

$$Q_i = \sum_{i=1}^{m} [D_i(t) \times \alpha_{ij}] + K \tag{4}$$

From equation (4), the activity of neurons at the input layer 'Q_i' denotes that the weighted input data, weight' and bias function, 'α_{ij}' denotes a weight between the j^{th} input layer neuron and the i^{th} hidden layer neuron, K denotes a bias. The input is transferred into the first hidden layer. In that layer, training data and testing data are analysed with the help of Camargo's index. The Camargo's index is used [35] to find the similarity between the testing and training data for identifying the fault or non-fault software products. Therefore, the Camargo's index-based testing and testing data are analyzed as given below,

$$CI = 1 - \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{|D_i - D_j|}{n} \tag{5}$$

where 'CI' denotes similarity outcomes, D_i denotes testing data, D_j indicates the training data, 'n' indicates total number of training data samples. The similarity value provides the

outcomes in the ranges from 0 to 1. Therefore, the hidden layer output is given below.

$$H = \sum_{i=1}^{L} \alpha_{ij}\, \beta\, (\alpha_{jk}\, h_o + K) \tag{6}$$

where, 'H' represents the result of the output layer, 'β' indicates an activation function, 'α_{jk}' denotes the j^{th} hidden layer neuron and k^{th} output layer neuron, 'α_{ij}' denotes a weight between input and output layer, 'h'_o denotes an output of previous hidden layer, 'L' denotes the number of hidden units. The Hardlimit activation function is applied [34] for finding the software faults.

$$\beta = \{1, CI > 0.50, CI < 0.5\} \tag{7}$$

The activation function 'β' returns '1' denotes a fault, 'β' returns '0' denotes a non- fault. Based on activation function results, software faults are correctly identified.

To obtain the better solution with minimum-error, the least square method is applied. The method of least squares is a standard approach in classification analysis to obtain better solutions by decreasing the sum of the squares of the residuals. The residual is measured between the difference between a target result and the observed result provided by the proposed classifier. It is given below:

$$R = [T - Y]^2 \tag{8}$$

where 'R' denotes a least square method output which is defined as a squared difference between the target results 'T' and output predicted by the activation function 'Y'.

The Levenberg–Marquardt algorithm is used [36] to minimize the least squares problems 'R'.

$$F = arg\ arg\, [T - Y]^2 \tag{9}$$

where, F denotes output of the Levenberg–Marquardt algorithm, *arg arg* denotes an argument of minimum function to minimize least squares problems. Finally, the accurate classification results are obtained at the output layer with minimum error. A statistical indexive levenberg extreme learning classifier algorithm is given below:

Algorithm 2 given above illustrates the algorithmic process of software fault prediction using statistical indexive levenberg extreme learning classifier with higher accuracy and minimum time consumption. The proposed learning classifier consists of many layers to analyze the given training data. The selected features with the training data are given to the input layer. In that layer, the random weights are assigned to each training set and added to the bias. Then the hidden layer performs feature ma through the testing and training data analysis with help of Camargo's index. Then the activation neuron in the hidden node evaluates the index value and identifies the fault or non- fault products. Finally, the Levenberg–Marquardt algorithm is applied to minimize the least square problem and obtain the final better classification results at the output layer.

Algorithm 2 Statistical Indexive Levenberg Extreme Learning Classifier

Input: selected features (i.e. training data)
D_i and testing data
Output: Increase the attack detection accuracy
Begin

1. **Number of significant features** taken as input at the input layer
2. **For each** training data D_i // **[hidden layer]**
3. **For each** testing feature D_j
4. Apply Camargo's index 'CI'
5. **end for**
6. **end for**
7. **Apply** Hardlimit activation function
8. **if** $(CI > th)$**then// [output layer]**
9. R returns '1'
10. Software fault is predicted
11. **else**
12. R returns '0'
13. Software fault not predicted
14. **end if**
15. Apply Levenberg–Marquardt algorithm to minimize the least square
16. Obtain the classification results at the output layer

End

IV. EXPERIMENTAL SETUP

The performance evaluation of the proposed PRILEC technique is carried out by the implementing the proposed approach using JAVA language. The results are analyzed using Software Defect Prediction Data Analysis dataset. This data is collected from [37]. This is a PROMISE repository and publicly available dataset for software engineering. The dataset includes 22 attributes or features or software metrics, and 10885 instances. The attribute information is given in Table 1. First, the relevant feature selection process is carried out through the correlation measure. With the selected relevant metrics, the deep learning technique is applied whether the module has software fault or not based on the last attribute 'faults' [38]. The results of the proposed PRILEC technique are compared with conventional DP-GCNN [1], and DNN-DP [2] methods. Performance parameters considered to examine the results of both proposed and existing methods are fault prediction accuracy, precision, recall, f-measure, prediction time, specificity, and software fault prediction time and error rate.

The confusion matrix of proposed OR-HREC method is demonstrated in Table 2.

Confusion matrix is a fashionable measure to address classification issues. It is used for binary classification as well as for multi-class classification issues. A confusion matrix is estimated from predicted and actual values. The row values of confusion metrics represented the corresponding true label

TABLE 1. Feature information.

S. No	Features or attributes or metrics	Description
1	loc	line count 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	10Code	line count
14	10Comment	count of lines of comments
15	10Blank	count of blank lines
16	10CodeAndComment	line count and count of lines of comments
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

TABLE 2. Confusion matrix.

Number of instances =1000	Classified: Yes	Classified: No	Total
Actual: No	TN=60	FP=7	67
Actual: Yes	FN=6	TP= 927	934
Total	66	934	1000

and column values are indicated the corresponding predicted labels. The value that appears in each cell shows the prediction labels. The number of instances (i.e, 1000) is taken in the dataset. TN represents the true negative, FP denotes false positive, FN indicates the false negative, and TP denotes the true positive.

V. PERFORMANCE RESULTS AND DISCUSSION

In this section, performance analysis of PRILEC technique and existing DP-GCNN [1], and DNN-DP [2] are discussed with the different parameters such as fault prediction accuracy, precision, recall, f-measure, prediction time, specificity, software fault prediction time and error rate. Performance results are assessed with the help of tables and graphical illustrations.

Software Fault Prediction Accuracy: It is measured as the number of instances that are correctly predicted as fault or not fault through the classification [2]. The fault prediction accuracy is mathematically stated as given below.

$$SFPA = \left[\frac{T_p + F_p}{T_p + F_p + T_n + F_n}\right] * 100 \qquad (10)$$

where $SFPA$ indicates a software fault prediction accuracy, T_p indicates true positive, F_p denotes false positive, T_n indicates true negative, F_n represents false negative. The accuracy is measured in percentage (%).

TABLE 3. Comparison of software fault prediction accuracy.

Number of instances	Software fault prediction accuracy (%)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	98.7	94	96.56
2000	98.5	93	96
3000	98.33	93	96
4000	98.12	92.75	96
5000	97.9	92.6	95.6
6000	97.5	92.5	95.5
7000	97.42	92.42	95.28
8000	97.25	92.25	95.25
9000	97.22	92.22	95.22
10000	97.2	91.5	95.1

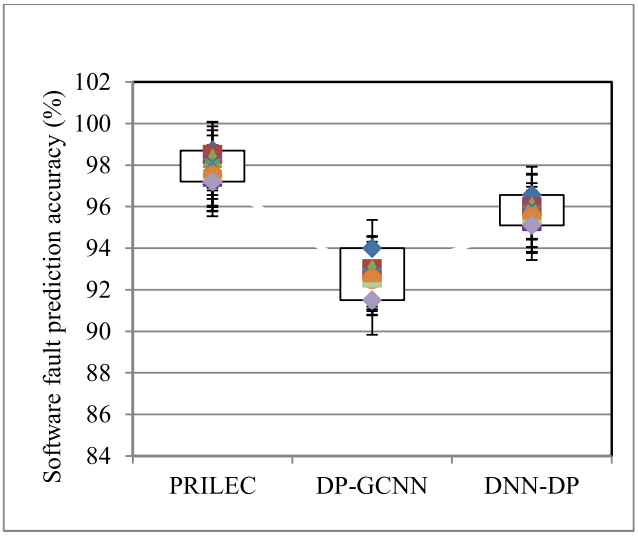

FIGURE 5. Comparison of software fault prediction accuracy.

Table 3 and Figure 5 illustrate the graphical illustration of the software fault prediction accuracy using proposed PRILEC technique and existing DP-GCNN [1], and DNN-DP [2]. In the above figure, the horizontal axis represents

the number of instances and the vertical axis indicates the measure of software fault prediction accuracy. The observed result indicates that the proposed PRILEC method outperforms other methods [1] and [2]. However, with simulations conducted with 1000 instances, the accuracy was observed to be 98.7% using proposed PRILEC and 94% and 96.56% using existing methods DP-GCNN [1], and DNN-DP [2] respectively. The overall performance of ten results indicates that the fault prediction accuracy using proposed PRILEC technique is considerably improved by 6% when compared to [1] and 2% when compared to [2] respectively. With this measure, the software fault prediction accuracy is improved using PRILEC upon comparison with the two other existing methods. The reason behind the improvement was due to the application of statistical indexive Levenberg extreme learning classifier with higher accuracy and minimum time consumption. The Camargo's index is measured by testing and training data. The Hardlimit activation function is applied to find the fault and non- fault modules. This helps to improve the true positive rate. The Levenberg–Marquardt algorithm is applied for minimizing the least square problem hence it reduces the false positive as well as false negative rate.

Precision: It is measured as the ratio of number of true positives as well as false positives [2]. Therefore, the accurate precision is mathematically formulated as given below,

$$Pr = \left(\frac{T_p}{T_p + F_p}\right) \times 100 \tag{11}$$

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

TABLE 4. Comparison of precision.

Number of instances	Precision (%)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	99.25	95.55	97.86
2000	98.93	95	97.26
3000	98.93	94.87	97.47
4000	98.67	94.80	97.31
5000	98.52	95.43	97.43
6000	98.25	95.48	97.52
7000	98.33	95.64	97.40
8000	98.41	95.64	97.32
9000	98.35	95.65	97.27
10000	98.31	95.11	97.33

A comparison of precision is portrayed in Table 4 and Figure 6. The proposed PRILEC technique and existing DP-GCNN [1], and DNN-DP [2] are utilized in Figure 6. The PRILEC technique of precision is higher compared to existing methods. For the 1000 data considered for simulation, the precision was observed as 99.25% using the PRILEC technique and 95.55% and 97.86% using the DP-GCNN [1], and DNN-DP [2] respectively. The precision was better with the statistical indexive levenberg extreme learning classifier for fault prediction. The Levenberg–Marquardt algorithm is utilized to an extreme learning classifier to diminish the least square problem and false positive rate. The results designate that the precision is increased by 3% and 1% than DP-GCNN [1], and DNN-DP [2].

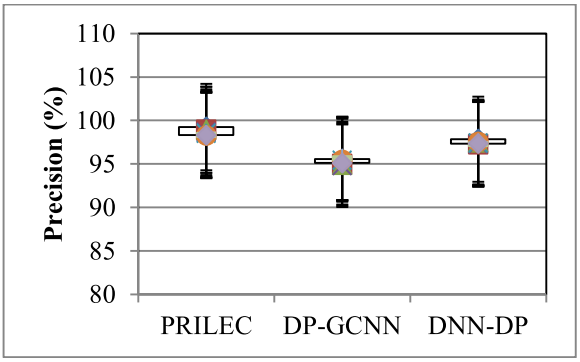

FIGURE 6. Comparison of precision.

Recall/ Sensitivity: It is estimated as a percentage of fault modules properly predicted by the classifier. The recall is calculated in percentage (%).

$$R_c = \left(\frac{T_p}{T_p + F_n}\right) \times 100 \tag{12}$$

where 'R_c 'denotes recall, T_p indicates true positive, F_n represents the false negative.

TABLE 5. Comparison of recall.

Number of instances	Recall (%)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	99.35	97.72	98.38
2000	99.46	97.15	98.34
3000	99.28	97.36	98.18
4000	99.33	97.19	98.36
5000	99.25	96.48	97.84
6000	99.11	96.35	97.69
7000	98.93	96.09	97.54
8000	98.67	95.90	97.59
9000	98.70	95.88	97.62
10000	98.73	95.63	97.44

Comparison of recall demonstrated in Table 5 and Figure 7 for PRILEC technique and existing methods. In Figure 7, number of instances indicated in the horizontal axis and performance of recall rate dented as the vertical axis. PRILEC technique of recall was enhanced than comparison with [1] and [2]. Assume 1000 instances to experiment. The performance of recall using the proposed PRILEC technique is 99.35% and the recall of existing [1], [2] are 97.72% and

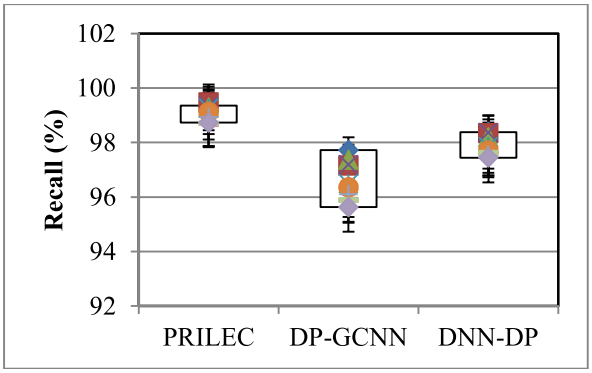

FIGURE 7. Comparison of recall.

98.38% respectively. The results show that the recall of the proposed technique is improved by 3% and 1% than [1] and [2] respectively. The reason for enhancing recall is to apply Extreme learning machines classifier uses Hardlimit activation function. The fault or non fault modules precisely classified with higher true positive and lesser true negative.

F-Measure: It is estimated by precision as well as recall [2]. F-measure is calculated in percentage (%).

$$MES_F = \left[2 * \frac{P_r * R_c}{P_r + R_c}\right] \times 100 \tag{13}$$

where MES_F designate an F-measure, 'P_r' indicates precision and 'R_c' denotes recall.

TABLE 6. Comparison of F-measure.

Number of instances	F-measure (%)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	99.3	96.62	98.11
2000	99.19	96.06	97.79
3000	99.10	96.09	97.82
4000	98.99	95.98	97.83
5000	98.88	95.95	97.63
6000	98.67	95.91	97.60
7000	98.62	95.86	97.46
8000	98.53	95.76	97.45
9000	98.52	95.76	97.44
10000	98.51	95.36	97.38

Table 6 and Figure 8 demonstrate the graphical illustration of the F-measure for a distinct number of instances taken from 1000 to 10000. From the observed results, F-measure was found to be 99.3% using the PRILEC technique, 96.62% using DP-GCNN [1] and 98.11% using DNN-DP [2]. From this result, the performance of the F-measure using the proposed PRILEC technique is improved when compared to existing methods. By applying this PRILEC technique, the performances of precision as well as recall were improved

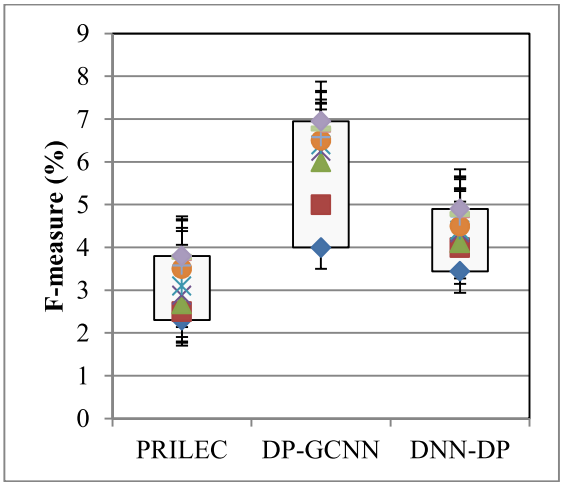

FIGURE 8. Comparison of F-measure.

during the software fault prediction. Followed by which the actual classifications were made in extreme learning classifier, therefore improving the F-measure using the PRILEC technique. The overall performance results indicate that the F-measure of PRILEC techniqueis considerably improved by 3% as compared to [1] and 1% as compared to [2] respectively.

Specificity: It is measured as the ratio of number of true negatives as well as false positives during the fault prediction in the software modules. It is calculated as given below,

$$Spe = \left(\frac{T_n}{T_n + F_p}\right) \times 100 \tag{14}$$

where '*Spe*' indicates a specificity, 'T_n' denotes a true negative, 'F_p' denotes the false positive. The performance of specificity is measured in percentage (%).

TABLE 7. Comparison of specificity.

Number of instances	Specificity (%)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	89.55	66.66	77.77
2000	85.714	62.5	73.68
3000	85	58.82	72
4000	80.76	55.81	68.75
5000	75.86	53.33	65.71
6000	69.69	50.98	60
7000	73.80	53.33	63.82
8000	73.91	52.23	62.26
9000	72	50.68	60.34
10000	69.23	45.78	59.67

Table 7 and Figure 9 depict the performance results of the Specificity against the number of instances taken from the dataset. The Specificity is calculated with respect to the true negative as well as false positive rate. The observed

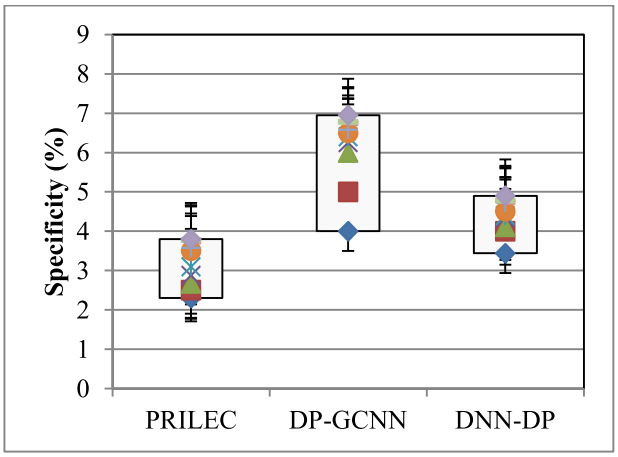

FIGURE 9. Comparison of specificity.

result states that the PRILEC technique provides improved performance when compared to existing methods. Let us consider 1000 instances taken from the dataset in the first iteration. The observed performance of the specificity using PRILEC is found to be 89.55% whereas the results of existing DP-GCNN [1] and DNN-DP [2] are found to be 66.66% and 77.77% respectively. Finally, the overall observed results of the proposed PRILEC technique are compared to the results of existing methods. The average of ten results indicates that the specificity is significantly improved by 41% and 17% when compared to existing methods.

Prediction Time: It is defined as the amount of time taken by the algorithm for predicting the fault or non- fault software modules. The overall prediction time is calculated as given below,

$$Pt = n \times [t\,(POI)] \tag{15}$$

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).

TABLE 8. Comparison of prediction time.

Number of instances	Prediction time (ms)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	20	28	24
2000	22	30	26
3000	25.5	36	27.6
4000	28	40	32
5000	30	42.5	35
6000	33	45	36
7000	36.4	49	42
8000	40	52	48
9000	43.2	58.5	51.3
10000	47	62	56

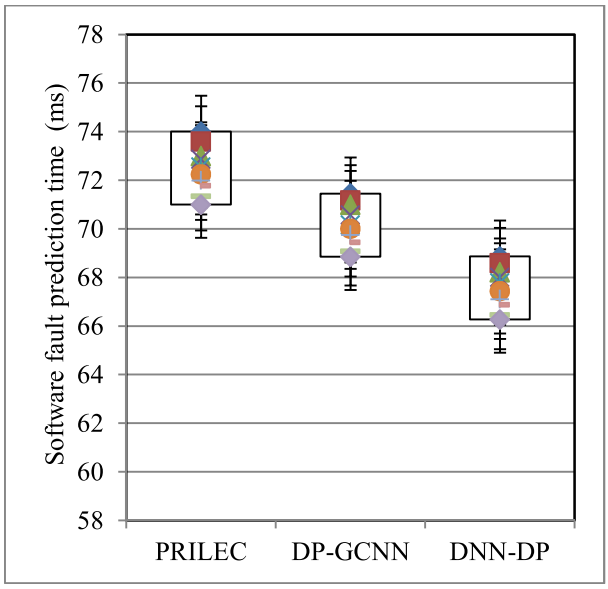

FIGURE 10. Comparison of prediction time.

Table 8 and Figure 10 demonstrates the graphical depiction of prediction time for software fault prediction using the three methods: PRILEC technique and existing DP-GCNN [1], and DNN-DP [2] respectively. From the figure it is inferred that the prediction time increases with the increase in the number of instances. This is due to the reason that with a larger number of instances involved during simulation, a large amount of time is said to be consumed during analyzing of different modules; this in turn increases the prediction time also. however, with experiments performed with 1000 instances, the time consumed in software fault prediction being '20ms', the overall prediction time using DP-GCNN [1] method was 28ms, the time consumed in detecting the software faults being '24 ms'. From this result it is inferred that the software fault prediction time using PRILEC was minimized by 27% and 14% when compared to [1] and [2]. The improvement is due to the application of congruence correlative piecewise regression. As given in the above algorithm with the objective of selecting the significant features for software fault prediction. The raw dataset is subjected to congruence correlative piecewise regression. First, with the input features acquired in the form of matrix, relationship between the features is determined by applying congruence correlation coefficient. Then the piecewise regression is applied to determine the important features for software fault prediction. This helps to minimize the prediction time.

Error Rate: It is one of the software metric and it is measured as the number of instances that are incorrectly predicted as fault or not fault through the classification. It is measured as given below.

$$Error\ rate = \left[1 - \frac{T_p + F_p}{T_p + F_p + T_n + F_n}\right] \times 100 \tag{16}$$

From equation (15),the Error rate is computed and it is measured in percentage (%).

TABLE 9. Comparison of error rate.

Number of instances	Error rate (%)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	2.3	4	3.44
2000	2.5	5	4
3000	2.67	6	4.11
4000	2.88	6.25	4.34
5000	3.1	6.4	4.4
6000	3.5	6.5	4.5
7000	3.58	6.58	4.72
8000	3.75	6.75	4.75
9000	3.78	6.78	4.78
10000	3.8	6.95	4.9

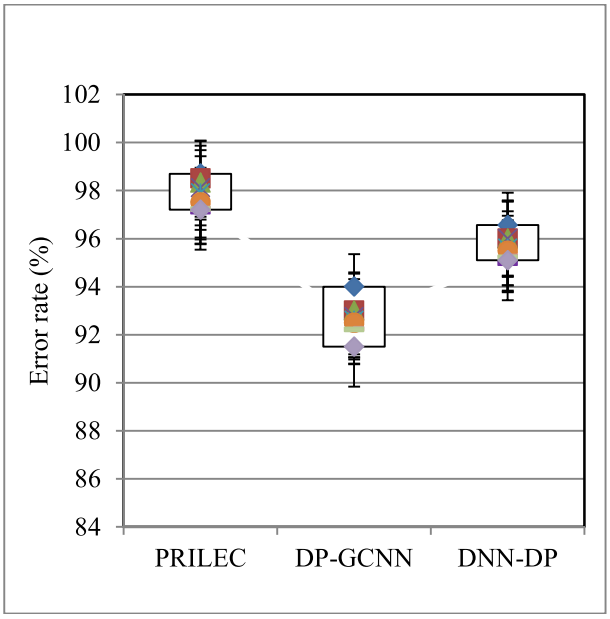

FIGURE 11. Comparison of error rate.

Table 9 and Figure 11 illustrate the comparison error rate for the proposed PRILEC technique and existing DP-GCNN [1], and DNN-DP [2]. The result indicates that the proposed PRILEC method outperforms other methods [1] and [2]. However, with simulations conducted with 1000 instances, the error rate was observed to be 2.3% using the proposed PRILEC and 4% and 3.44% using the existing two methods DP-GCNN [1], and DNN-DP [2] respectively. The average value of error rate using the proposed PRILEC technique is considerably reduced by 50% when compared to [1] and 28% when compared to [2] respectively. The error rate is minimized using PRILEC than the two other existing methods. The reason for lesser error rate is to apply a statistical IndexiveLevenberg extreme learning classifier. The testing and training data is investigated with Camargo's index. Fault and non-fault modules are determined by the Hardlimit activation function. The least square issue is handled via Levenberg–Marquardt algorithm which minimizes the false positive and false negative rates. Hence, the error rate is decreased.

Hypothesis Test: The statistical test for software fault prediction is done in our work by using the McNemar test. McNemar's test is also called as the paired or matched chi-square. The McNemar test is utilized as a non-parametric test for paired nominal data. In order to evaluate the McNemar test, the number of instances is said to be placed into a 2×2 contingency table.

TABLE 10. Tabulation for McNemar test for proposed PRILEC method.

	Wrongly predicted	Correctly predicted	
Wrongly predicted	7 (a)	60 (b)	67
Correctly predicted	6 (c)	927 (d)	933
	13	987	1000

In table, cells b and c are employed to estimate the McNemar test statistic and it is as given below.

$$\chi^2 = \frac{(b-c)^2}{b+c} \tag{17}$$

Table 10 offers the tabulation outcomes of the McNemar test involved in software fault prediction for three methods, PRILEC and existing DP-GCNN [1], and DNN-DP [2]. From the comparison, it is found that the proposed PRILEC performs better performance results in terms of the McNemar test than [1] and [2].

TABLE 11. Comparison of McNemar test.

Number of instances	MCNEMAR TEST (M-TEST)		
	PRILEC	**DP-GCNN**	**DNN-DP**
1000	74	71.45	68.86
2000	73.6	71.18	68.59
3000	73	71	68.22
4000	72.86	70.57	68
5000	72.55	70.16	67.78
6000	72.24	70	67.44
7000	72	69.76	67.11
8000	71.78	69.45	66.88
9000	71.35	69.08	66.46
10000	71	68.85	66.27

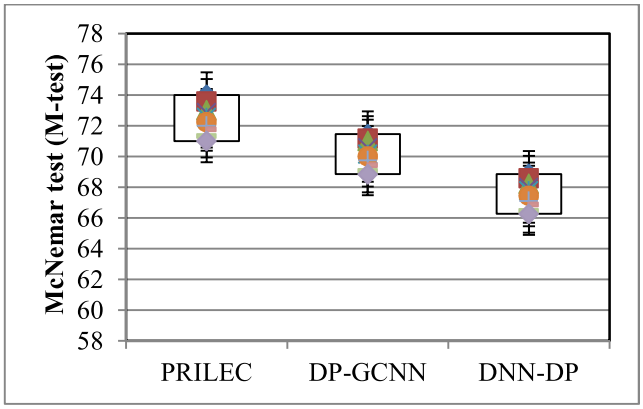

FIGURE 12. Comparison of McNemar test.

Table 11 and Figure 12 demonstrate the McNemar test (M-test) using the proposed PRILEC and existing DP-GCNN [1], and DNN-DP [2]. A hypothesis test is conducted based on a sample between 1000 and 10000. The reason behind the improvement was the application of initially obtaining highly correlated features with the aid of Camargo's index among two sample data. M-test is better performance for PRILECby 3% and 7% than the existing methods respectively.

Comparative Analysis of the Proposed Prilec Technique With Existing Methods: The analysis of the proposed PRILEC and existing DP-GCNN [1], and DNN-DP [2] are compared and implemented in Java. Table 11 provides a detailed comparison result of the proposed PRILEC technique with the existing methods.

TABLE 12. Comparative analysis of proposed PRILEC technique with existing methods.

METRICS	METHODS		
	Proposed PRILEC	Existing DP-GCNN [1]	Existing DNN-DP [2]
Software fault prediction Accuracy (%)	97.81	92.62	96.56
Precision (%)	98.59	95.31	97.41
Recall (%)	99.08	96.57	97.89
F-measure(%)	98.83	95.93	97.65
Specificity (%)	77.55	55.01	66.40
Software fault prediction time (ms)	32.51	44.3	37.79
Error rate	3.18	6.12	4.39

Table 12 demonstrates the comparative analysis of proposed PRILEC and existing methods. From the experimental results, the following summary key finds are achieved: Proposed PRILEC achieved higher accuracy, precision, recall, F-measure, and specificity by 97.81%, 98.59%, 99.08%, 98.83%, and 77.55% when compared to DP-GCNN [1], and DNN-DP [2]. PRILEC also minimizes the prediction time and error rate by 32.51 msand 3.18% when compared to existing methods. The reason for maximum accuracy and minimal time is to apply congruence correlative coefficient piecewise regression and statistical indexive levenberg extreme learning classifier. In comparative analysis, the PRILEC provides better performance for software fault prediction and attains overall objectives in terms of accuracy, precision, recall, F-measure, and time.

VI. DISCUSSION

The aim of the proposed PRILEC technique is to determine fault or non-fault with higher accuracy and minimize time. Based on this objective, the proposed PRILEC technique and existing DP-GCNN [1], and DNN-DP [2] are evaluated through modules written in Java with Software Defect Prediction Data Analysis dataset. The reason for less time is to apply congruence correlative coefficient piecewise regression for chosen pertinent features and immaterial features. The proposed PRILEC technique saves time efficiently as compared with the conventional methods. The reason for higher accuracy, precision, recall, and F-measure is by applying a statistical indexive levenberg extreme learning classifier for accurately classifying the fault. From the experimental results, the following summary key finds are achieved: The proposed PRILEC technique achieved higher accuracy by 4%, precision by 2%, recall by 2%, F-measure 2%, and specificity by 29% when compared [1] and [2]. PRILEC technique also minimizes the time by 21% when compared to existing methods. In comparative analysis, the PRILEC technique provides better performance for software fault prediction and attains overall objectives in terms of accuracy, and time.

VII. THREATS TO VALIDITY

We determined and assessed a set of threats to guarantee the quality of this empirical investigation. Every threat is explained, along with the methods taken to reduce the threats. Four validity are considered such as internal validity, construct validity conclusion validity, and external validity. An internal threat is determined in the metrics employed as independent variables to forecast software faults associated with the trustworthiness of results. In this experiment, construct validity is the evaluation metrics employed to estimate the prediction performance. The evaluation measures such as the F-measure as one threshold-dependent measure for enhancing the performance. The commonly used measures are considered in software fault prediction for comparison with proposed and other studies. Several statistical comparison tests were used to compare proposed and existing

methods. The underlying assumptions influence the statistical test that should be applied. We utilized the McNemar test for paired nominal data. In this work, ten cross-validations are used for the feature selection method and classification. We tested our study on the Software Defect Prediction Data Analysis dataset with 22 attributes and 10885 instances. Second, extreme learning models are mainly affected by the data. Features are chosen by using characteristics of the dependent variable. Numerous trendy performance metrics are employed for feature selection and classification problems, such as accuracy, recall, precision, F-measure, and error rate. Lastly, external validity is the ability to simplify the results of the study for the dataset.

VIII. CONCLUSION

Software fault prediction is utilized for finding faults in the software. But, time and accurate detection are major issues. To improve the performance of software quality prediction for accurate detection, we propose PRILEC technique. Significant features or metrics are selected from the dataset using congruence correlative coefficient piecewise regression. After, statistical indexive levenberg extreme learning classifier is employed for software fault prediction with a minimum error rate. The experimental is carried out with respect to a number of instances. The proposed PRILEC technique is compared with the two existing methods (i.e. DP-GCNN, and DNN-DP). The results of the proposed technique with two conventional algorithms using the software fault prediction data analysis dataset in Java. The statistical results confirmed that the proposed PRILEC outperforms for achieving a higher software fault prediction accuracy, precision, recall, f-measure, and specificity with minimum time than the other existing methods by 4%, 2%, 2%, 2%, 29%, and 21% respectively.

In future work, Advanced technique will be developed to identify the fault and quality of the software. For future research, the proposed technique is extended to provide more quality software applications. However, the application of the software fault prediction is still need to be enhanced and more number of methods should be carried out in order to obtain well-formed and generalizable results.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest.

DATA AVAILABLE STATEMENT

Not Applicable.

REFERENCES

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

[2] M. Gupta, K. Rajnish, and V. Bhattacharjee, "Cognitive complexity and graph convolutional approach over control flow graph for software defect prediction," *IEEE Access*, vol. 10, pp. 108870–108894, 2022, doi: 10.1109/ACCESS.2022.3213844.

[3] W. Wen, R. Zhang, C. Wang, C. Shen, M. Yu, S. Zhang, and X. Gao, "A cross-project defect prediction model based on deep learning with self-attention," *IEEE Access*, vol. 10, pp. 110385–110401, 2022, doi: 10.1109/ACCESS.2022.3214536.

[4] H. Wang, W. Zhuang, and X. Zhang, "Software defect prediction based on gated hierarchical LSTMs," *IEEE Trans. Rel.*, vol. 70, no. 2, pp. 711–727, Jun. 2021, doi: 10.1109/TR.2020.3047396.

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

[6] M. S. Alkhasawneh, "Software defect prediction through neural network and feature selections," *Appl. Comput. Intell. Soft Comput.*, vol. 2022, pp. 1–16, Sep. 2022, doi: 10.1155/2022/2581832.

[7] K. Tameswar, G. Suddul, and K. Dookhitram, "A hybrid deep learning approach with genetic and coral reefs metaheuristics for enhanced defect detection in software," *Int. J. Inf. Manage. Data Insights*, vol. 2, no. 2, Nov. 2022, Art. no. 100105, doi: 10.1016/j.jjimei.2022.100105.

[8] S. Zhang, S. Jiang, and Y. Yan, "A software defect prediction approach based on BiGAN anomaly detection," *Sci. Program.*, vol. 2022, pp. 1–13, Apr. 2022, doi: 10.1155/2022/5024399.

[9] 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, Mar. 2021, Art. no. e0247444, doi: 10.1371/journal.pone.0247444.

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

[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] J. Xu, F. Wang, and J. Ai, "Defect prediction with semantics and context features of codes based on graph representation learning," *IEEE Trans. Rel.*, vol. 70, no. 2, pp. 613–625, Jun. 2021, doi: 10.1109/TR.2020.3040191.

[13] X. Xiaolong, C. Wen, and W. Xinheng, "RFC: A feature selection algorithm for software defect prediction," *J. Syst. Eng. Electron.*, vol. 32, no. 2, pp. 389–398, Apr. 2021, doi: 10.23919/JSEE.2021.000032.

[14] J. Zheng, X. Wang, D. Wei, B. Chen, and Y. Shao, "A novel imbalanced ensemble learning in software defect predication," *IEEE Access*, vol. 9, pp. 86855–86868, 2021, doi: 10.1109/ACCESS.2021.3072682.

[15] 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.

[16] A. Wang, Y. Zhang, and Y. Yan, "Heterogeneous defect prediction based on federated transfer learning via knowledge distillation," *IEEE Access*, vol. 9, pp. 29530–29540, 2021, doi: 10.1109/ACCESS.2021.3058886.

[17] N. S. Harzevili and S. H. Alizadeh, "Analysis and modeling conditional mutual dependency of metrics in software defect prediction using latent variables," *Neurocomputing*, vol. 460, pp. 309–330, Oct. 2021, doi: 10.1016/j.neucom.2021.05.043.

[18] 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," *J. Syst. Softw.*, vol. 180, Oct. 2021, Art. no. 111026, doi: 10.1016/j.jss.2021.111026.

[19] Z. Sun, J. Li, H. Sun, and L. He, "CFPS: Collaborative filtering based source projects selection for cross-project defect prediction," *Appl. Soft Comput.*, vol. 99, Feb. 2021, Art. no. 106940, doi: 10.1016/j.asoc.2020.106940.

[20] J. Zhao, S. Guo, and D. Mu, "DouBiGRU-A: Software defect detection algorithm based on attention mechanism and double BiGRU," *Comput. Secur.*, vol. 111, Dec. 2021, Art. no. 102459, doi: 10.1016/j.cose.2021.102459.

[21] S. C. Rathi, S. Misra, R. Colomo-Palacios, R. Adarsh, L. B. M. Neti, and L. Kumar, "Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction," *Expert Syst. Appl.*, vol. 223, Aug. 2023, Art. no. 119806, doi: 10.1016/j.eswa.2023.119806.

[22] L. Kumar, S. K. Sripada, A. Sureka, and S. K. Rath, "Effective fault prediction model developed using least square support vector machine (LSSVM)," *J. Syst. Softw.*, vol. 137, pp. 686–712, Mar. 2018, doi: 10.1016/j.jss.2017.04.016.

[23] L. Kumar, A. Tirkey, and S.-K. Rath, "An effective fault prediction model developed using an extreme learning machine with various kernel methods," *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 7, pp. 864–888, Jul. 2018, doi: 10.1631/fitee.1601501.
[24] J. M. Sunil, L. Kumar, and L. B. M. Neti, "Bayesian logistic regression for software defect prediction (S)," in *Proc. Int. Conferences Softw. Eng. Knowl. Eng.*, Jul. 2018, pp. 1–6, doi: 10.18293/seke2018-181.
[25] L. Kumar, S. Misra, and S. K. Rath, "An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes," *Comput. Standards Interfaces*, vol. 53, pp. 1–32, Aug. 2017, doi: 10.1016/j.csi.2017.02.003.
[26] L. Kumar and S. K. Rath, "Empirical validation for effectiveness of fault prediction technique based on cost analysis framework," *Int. J. Syst. Assurance Eng. Manage.*, vol. 8, no. 2, pp. 1055–1068, Nov. 2017, doi: 10.1007/s13198-016-0566-4.
[27] L. Kumar, S. Rath, and A. Sureka, "An empirical analysis on effective fault prediction model developed using ensemble methods," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2017, pp. 244–249, doi: 10.1109/COMPSAC.2017.53.
[28] J. Goyal and B. Kishan, "Progress on machine learning techniques for software fault prediction," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 8, no. 2, pp. 305–311, Apr. 2019, doi: 10.30534/ijatcse/2019/33822019.
[29] I. Mehmood, S. Shahid, H. Hussain, I. Khan, S. Ahmad, S. Rahman, N. Ullah, and S. Huda, "A novel approach to improve software defect prediction accuracy using machine learning," *IEEE Access*, vol. 11, pp. 63579–63597, 2023, doi: 10.1109/ACCESS.2023.3287326.
[30] H. Li, X. Zhang, and Y. Yang, "Congruence correlative piecewise regression-based feature selection for software fault prediction," *J. Softw., Evol. Process*, vol. 36, no. 3, pp. 487–501, 2024.
[31] Q. Wang, Z. Liu, and J. Sun, "Piecewise regression model for nonlinear software fault prediction," *IEEE Trans. Softw. Eng.*, vol. 50, no. 1, pp. 78–92, Jan. 2024.
[32] L. Zhao, P. Wang, and M. Li, "Enhanced software fault prediction using statistical indexive Levenberg extreme learning machines," *Int. J. Mach. Learn. Cybern.*, vol. 15, no. 4, pp. 625–638, 2024.
[33] Y. Chen and G. B. Huang, "Optimizing software fault prediction models using Levenberg–Marquardt algorithm," *Appl. Soft Comput.*, vol. 124, no. 1, 2024, Art. no. 109631.
[34] U. Lorenzo-Seva and J. M. F. Ten Berge, "Tucker's congruence coefficient as a meaningful index of factor similarity," *Methodology*, vol. 2, no. 2, pp. 57–64, Jan. 2006, doi: 10.1027/1614-2241.2.2.57.
[35] P. Goudarzian and S. Y. Erfanifard, "The efficiency of indices of richness, evenness and biodiversity in the investigation of species diversity changes (case study: Migratory water birds of Parishan international wetland, Fars Province, Iran)," *Biodiversity Int. J.*, vol. 1, no. 2, pp. 41–45, Aug. 2017, doi: 10.15406/bij.2017.01.00007.
[36] *Hardlimit Activation Function*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.cs.montana.edu/courses/spring2005/530/help/nnhelp/hardlim.html
[37] M. K. Transtrum and J. P. Sethna, "Improvements to the Levenberg–Marquardt algorithm for nonlinear least-squares minimization," 2012, *arXiv:1201.5885*.
[38] *Software Defect Prediction Data Analysis Dataset*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.kaggle.com/code/semustafacevik/software-defect-prediction-data-analysis/data

SUREKA SIVAVELU received the M.Sc. degree (Hons.) in CS and the M.Tech. degree in CSE from VIT. She has been an Assistant Professor (Sr.) with the School of Computer Science Engineering and Information Systems, Vellore Institute of Technology, Vellore, since 2007. She used to hold some administrative posts with the School of Information Technology and Engineering. She has supervised more than 20 UG and PG projects. Her research interest includes software testing.

VENKATESH PALANISAMY received the M.S. degree in research and the Ph.D. degree in computer science domain from Anna University, Chennai, India, in 2008 and 2013, respectively. He is currently a Professor with the School of Computer Science Engineering and Information Systems, Vellore Institute of Technology, India. He has more than 21 years of teaching experience and guiding two Ph.D. students. He has published articles in reputed peer-reviewed national and international journals. He holds the post of Assistant Director of the Software Development Cell, involved in software product design. His research interests include machine learning, data analytics, the Internet of Things, and healthcare analytics.