International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Software Fault Prediction Using Canonical Discriminant Quadratic Regressive Milboost Ensemble classifier

[1]Mr. T. Shathish Kumar, [2]Dr. B. Booba

Submitted: 07/02/2024 **Revised**: 15/03/2024 **Accepted**: 21/03/2024

Abstract: The aim of software fault prediction is to sense fault-prone software modules and enhances software quality as well as testing effectiveness through early recognition of faults. It aids to achieve desired software quality through lower cost. Earlier fault prediction classification algorithms to forecast fault-prone software modules. The prediction accuracy of conventional techniques was established to be significantly minimized with better misclassification. The selection of significant metrics from the source code is the fundamental step in the software prediction process. Therefore a novel technique called **CA**nonical **D**iscriminant Qu**a**dratic Regressive **M**ilboost **E**nsemble (CADME) technique is introduced for improving the software prediction accuracy as well as minimizing misclassification rate. Proposed CADME technique comprised metric selection , classification. Initially, number of JAVA packages given as input from the dataset. Then the metric dispersal class extractions are carried by using Jarque–Bera stochastic test. After the class extraction, the important software metrics are chosen for software prediction using generalized canonical correlative normal discriminant analysis. Following the metric selection, the software fault prediction is through by means of Iterative Dichotomize Linear Regressive Quadratic MilBoost. As a result of JAVA classes known with defects or not are predicted in an accurate manner by reducing the loss. This assists to develop the precision and F-score in fault prediction. Simulation results are performed on factors namely accuracy, precision, recall, F-score, and time complexity. The results as well as discussion of various metrics specified that proposed CADME technique improves the accuracy and minimum time complexity of software prediction than the conventional methods.

Keywords: *Software fault prediction, Jarque–Bera stochastic test-based metric dispersal class extractions, generalized canonical correlative normal discriminant analysis based software metrics selection, Iterative Dichotomize Linear Regressive Quadratic MilBoost.*

1. INTRODUCTION

Software fault forecast is employed to enhance software quality, as well as improve the maintenance in the testing. Software fault is a defect that takes occurs when the expected result doesn't equivalent to the actual results. Testing is important part of software improvement process however is usually manual, error-prone, as well as luxurious. Software testing is process of producing reliable, robust, as well as trustworthy software code through detecting faults. But, it still time-utilizing process. In a few testing circumstances, numerous faults are identified over long time. Therefore the detection of faults with minimum time consumption is a demanding issue to improve software quality. Numerous works have designed for software fault prediction.

For identifying the process of detecting software defects , Hybrid Approach to detect Large Class Bad Smell (HA-LCBS) was developed in [1]. But refactoring procedure of huge class imperfection prediction was not performed. For predicting software module as defective or not defective, Defect Prediction based on Convolutional Graph Neural Network (DP-GCNN) was designed in [2] . However time utilization of defect forecast was challenging issue.

The different ensemble ML techniques were presented in [3] for predicting defects of software systems. But it failed to carry out defects prediction with more accuracy and minimum errors. For code smell recognition based on feature selection, A new four-way approach was introduced in [4] .But correlation between the learners was not analyzed to improve the code smell detection accuracy.

A Diverse Ensemble Learning Technique (DELT) was introduced in [5] for software fault classification. But it failed to estimate quality of software scheme . An extended Random Forest (extRF) technique was designed in [6] for software defect forecast and analysis. However it failed to give analytical assessment of ML methods for forecast uses.

For cross-project defect forecast , Kernel Spectral Embedding Transfer Ensemble (KSETE) technique was designed in [7]. But feature learning capability was not improved. Hybrid Swarm Intelligence and DL model was developed [8] for enhancing software quality and efficiency by predicting the defects. However, it failed to

[1]*Research Scholar, Dept.of CSE, VISTAS, VEL's University, Chennai, Tamil Nadu, India. Country*
shathish098@gmail.com
[2]*Professor, Dept.of CSE, VISTAS, VEL's University, Chennai, Tamil Nadu, India. Country*
boobarajashekar@gmail.com

extract features as well as build software defect forecast models as well as compare efficiency of dissimilar learning algorithms. For detecting code smells, Three-hybrid feature selection by ensemble ML algorithms was developed in [9]. But structural metrics were not adequate for detecting code smell to achieve better performance.

A stacking heterogeneous ensemble approach was introduced in [10] for predicting the code smell. But the model complexity in ensemble learning was not minimized to observe trade-offs among building multifaceted method and attaining superior recognition performance.

1.1 contributions of manuscript

To overcome conventional problems, new CADME technique is designed by new contributions,

- CADME technique is developed depend on class extraction, metric selection, as well as classification. This technique enhances software fault prediction accuracy.
- CADME technique uses the Jarque–Bera stochastic test to extract the Java classes from the software packages. A generalized canonical correlative normal discriminant analysis is applied in the CADME technique to identify the relevant and irrelevant software metrics. This process minimizes the software fault prediction.
- Iterative Dichotomize Linear Regressive Quadratic MilBoost is applied for predicting the defective or non-defective java extracted classes by means of constructing the weak learners as Gibbs-Poston indexive ID3 (Iterative Dichotomiser 3) decision tree. The outputs of weak learner results are combined and compute the quadratic loss. The proposed ensemble classifier uses segmented linear regression to select the weak learner to further reduce time complexity. Downhill simplex technique is utilized to find strong classification outcomes by minimum loss. This process improves the accuracy and F-score.
- Lastly, comprehensive experimental evaluation is performed through assortment of performance metrics to demonstrate enhancement of CADME method over existing techniques.

1.2 Outline of manuscript

Outline of manuscript is structured as below. In Section 2, presents literature review of software fault prediction. Section 3 is gives a concise clarification of CADME technique through architecture diagram and various processes. In Section 4, provides experimental settings by dataset explanation. In section 5, performance outcomes of CADME method as well as conventional techniques are discussed through dissimilar parameters. Finally, Section 6 summarizes manuscript.

2. Literature Review

Deep learning models were performed in [11] to classify the smells for detecting code smells. However, it failed to minimize the code smell detection. In [12], five ensemble ML and two DL algorithms were developed to distinguish code smells from the source code. But learning algorithms were not explored to discover greatest methods for code smell recognition.

Gated Recurrent Unit (GRU) through LSTM was developed [13] for software defect prediction. But, it failed to extract significant features for software defect forecast. Temporal Convolutional Network (TCN) was developed in [14] to forecast presence of design code smells. But approach was not effective to improve prediction performance results.

For predicting software defect patterns, Graph neural network was developed in [15] . However, it failed to discover improved technique of faulty code smell. A Gated hierarchical LSTM network (GH-LSTM) was designed [16] to extract both semantic features for defecting the prediction of code smell. But it failed to investigate performance of designed method in cross-project defect forecast.

For enhancing ranking-oriented defect forecast , cost-sensitive rankingSVM (CSRankSVM) method was introduced in [17]. However, data imbalance learning was not performed. Multiple machines learning approach-based code smell detection was developed in [18] for recognition of code smells. But it failed to develop a systematic scheme for code smell recognition. Multi-label classification method was introduced in [19] for predicting the test smells by using a deep representation of the source code. However, it was not efficient to perform other types of test smells. A decision tree algorithm was designed in [20] for identifying software defects. However, it did not focus on eliminating the defects and correction approaches.

3. Proposal Methodology

In software scheme, test code is dependable for testing source code. These source codes are vulnerable to dissimilar problems such as potential defective modules in software products during development. Software fault is contaminated portion of program, that sometime expires program unforeseenly, or may assist hackers to read the program, that have disturbing effect on quality as well as security of software products. Therefore, Accurate and timely fault prediction has become a fundamental aspect to improve software quality and reliability. Based on this motivation, an accurate and timely software fault forecast CADME method is introduced. motivation for CADME technique is to select the software metrics

through ML methods for the classification process with no degrading model's effectiveness.

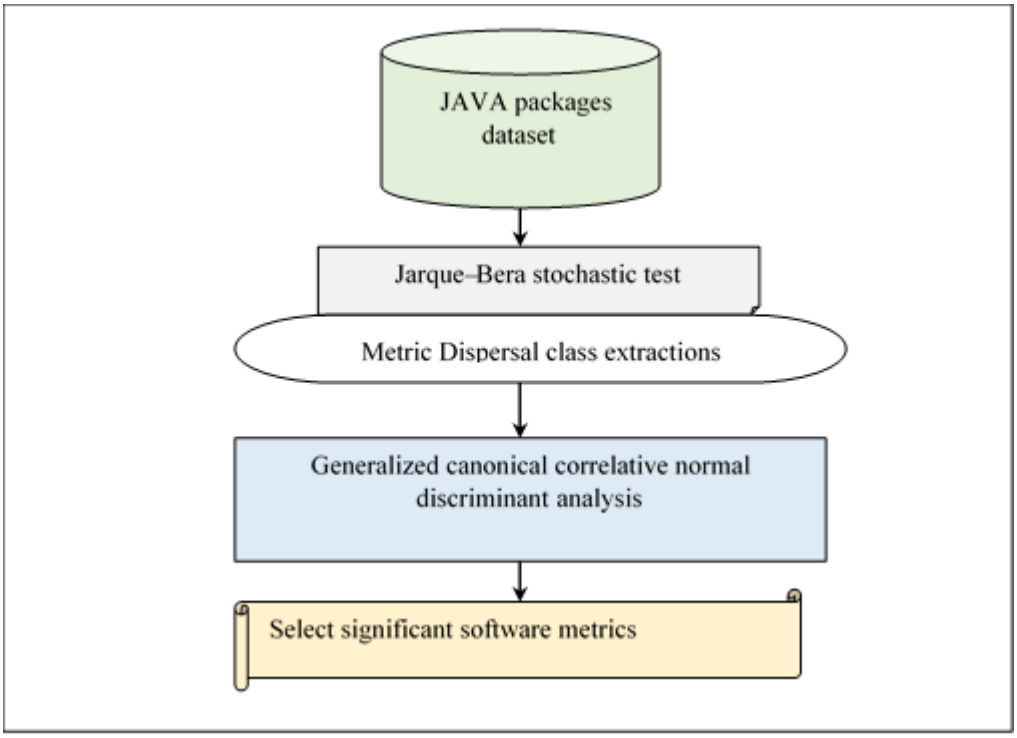

Figure 1 architecture of the proposed CADME technique

Figure 1 given above illustrates architecture of proposed technique to predict software faults through enhanced accuracy as well as minimum time utilization. Initially, JAVA packages Dataset 'DS is considered as input. The number of JAVA packages '$P = \{P_1, P_2, \dots, P_n\}$' are collected from dataset. First, proposed CADME method carry out the normal metric and abnormal metric dispersal class extraction by applying the Jarque–Bera stochastic test. Jarque–Bera stochastic test is statistical test employed to identify whether sample data (i.e. class) matches a normal distribution. After the class extraction, generalized canonical correlative normal discriminant analysis is applied to select the significant software metrics. Normal discriminant analysis is ML technique to recognize pertinent and irrelevant metrics through assist of generalized canonical correlation. Generalized canonical correlation analysis is a statistical method used for finding the cross-correlation among the sets of variables (i.e. metrics).

After extracting the classes and metrics, the CADME technique performs the software fault prediction using Iterative Dichotomize Linear Regressive Quadratic MilBoost. The Quadratic MilBoost is an ensemble technique used for predicting the defective or non-defective java extracted java classes by constructing the set of weak learners as a Gibbs-Poston indexive ID3 (Iterative Dichotomiser 3) (GIID3) decision tree. Output of weak learner outcomes is merged and measures quadratic loss. Segmented linear regression is ML to select weak learner results through minimum loss by setting the threshold for reducing time complexity of the fault forecast. The downhill simplex method is a numerical method to find strong classification results with minimum loss.

3.1 Jarque–Bera stochastic generalized canonical normal discriminant analysis

CADME method is to perform class extraction and metric selection. While constructing a ML model for software fault forecast, not all metrics are important every time. Adding unnecessary software metrics directs to decrease in general accuracy of method as well as increases its complexity. Hence, class extraction and metric selection process are the important fundamental steps as building ML model to enhance overall accuracy of software fault forecast and minimize the complexity.

Therefore, the proposed CADME technique first performs the metric dispersal class extractions and significant metric selection. The metric dispersal class extractions process is done by applying a Jarque–Bera stochastic test. The proposed stochastic test is a statistical measure of the probability distribution of a random variable such as the measured empirical JAVA package function 'and the cumulative JAVA package function. The stochastic test refers to the random probability distribution or patterns that are analyzed statistically.

After that**,** generalized canonical correlative normal discriminant analysis is used to select significant metrics for software fault forecast. It is ML method used for determining a linear combination of metrics that separates into two or more sets namely relevant as well as irrelevant metrics. Resulting relevant features used for dimensionality reduction before later classification. Dimensionality reduction is procedure of minimizing number of irrelevant metrics through attaining set of relevant metrics.

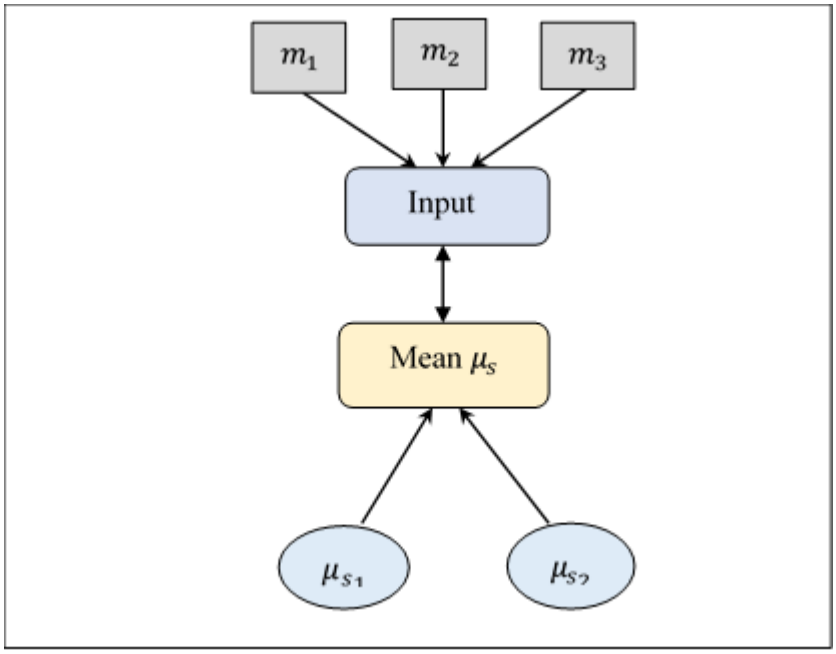

Figure 2 flow process of Jarque–Bera stochastic generalized canonical normal discriminant analysis

Figure 2 shows the flow process of class extraction and software metric selection by using Jarque–Bera stochastic generalized canonical normal discriminant analysis. First, the JAVA packages were collected from the dataset and provided as input. By applying the Jarque–Bera stochastic test, normal metric dispersal classes and abnormal metric dispersal classes are estimated. The Jarque–Bera stochastic test analysis process is given below,

$$S_t = \frac{m}{6}\left[{\varphi_s}^2 + 0.25(\varphi_k - 3)^2\right] \quad (1)$$

Where, m denotes the number of Java Packages, φ_s denotes a sample skewness, φ_k is the sample kurtosis. The skewness and kurtosis describe a particular aspect of a probability distribution. The sample skewness is measure of asymmetry of probability distribution of random variables namely measured empirical JAVA package function 'and cumulative JAVA package function.

$$\varphi_s = \frac{\frac{1}{m}\sum\left(EF_n(p) - CF_n(p)\right)^3}{\frac{1}{m}\sum\left(\left(EF_n(p) - CF_n(p)\right)^2\right)^{3/2}} \quad (2)$$

Where, φ_s denotes a skewness, empirical JAVA package function '$EF_n(p)$' and cumulative JAVA package function '$CF_n(p)$' , m denotes a number of Java Packages.

$$\varphi_k = \frac{\frac{1}{m}\sum\left(EF_n(p) - CF_n(p)\right)^4}{\frac{1}{m}\sum\left(\left(EF_n(p) - CF_n(p)\right)^2\right)^2} \quad (3)$$

Where, φ_k denotes a kurtosis, $EF_n(p)$ denotes an empirical JAVA package function and '$CF_n(p)$' denotes a cumulative JAVA package function, m denotes a number of Java Packages. The output of the Jarque–Bera stochastic test 'S_t' is always nonnegative. If it is distant from zero, the metric dispersal classes is identified the abnormal metric dispersal classes. The output of 'S_t is closer to the zero is said to be a normal metric dispersal classes. In this way, metric dispersal classes are extracted.

With the extracted classes, significant software metrics are selected for accurate software fault prediction to minimize the complexity. The proposed technique uses the generalized canonical correlative normal discriminant analysis**.** It is ML method for identifying relevant and irrelevant metrics through generalized canonical correlation. It is method of creating sense of cross-correlation among sets of variables (i.e. metrics).

Let us consider the number of metrics

$$M_i = \{m_1, m_2, m_3, \ldots m_n\} \in DS \quad (4)$$

Where, DS denotes a dataset, $m_1, m_2, m_3, \ldots m_n$ denotes the number of software metrics. Here two sets such as relevant and irrelevant sets are initialized. Mean value of set is computed as below,

$$\mu_s = \frac{1}{n}\sum_{i=1}^{n} m_i \quad (5)$$

Where, μ_s denotes a mean of sets and n indicates a number of software metrics. Therefore, the correlation between the mean and software metrics is estimated as given below,

$$CC = c_{var}\,[\mu_s, m_i] \quad (6)$$

Where CC correlation, c_{var} denotes a covariance measured between the mean value 'μ_s' and the metrics 'm_i'.

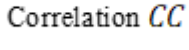

Figure 3 correlations between mean and software metrics

Figure 3 depicts the correlation between the mean and software metrics for finding the significant metrics. The correlation is mathematically computed as given below,

$$c_{var}\,[\mu_s, m_i] = \frac{\sum(m_i - \mu_{si})(m_j - \mu_{sj})}{n} \quad (7)$$

Where, 'μ_{si}' indicates mean of set 'i' and 'μ_{sj}' denotes mean of set 'j'. Therefore, the minimum covariance provides higher correlation results. Based on correlation measures, the metrics are identified as relevant as well as irrelevant. Only significant relevant software metrics are chosen for fault prediction. Algorithm of Jarque–Bera stochastic generalized canonical normal discriminant analysis is given below,

// Algorithm 1: Jarque–Bera stochastic generalized canonical normal discriminant analysis

Input: Dataset 'DS', Java Packages '$P = \{P_1, P_2, \dots, P_n\}$'

Output: Significant software metric selection

Begin

Step 1: For each Java Package 'P' in the Dataset 'DS'

Step 2: Measure Jarque–Bera stochastic test 'S_t'

Step 3: Estimate the difference between the empirical JAVA package function and the cumulative JAVA package function using (2) (3)

Step 4: If '$S_t\ closer\ to\ 0$'

Step 5: Strong correlation established between '$EF_n(p)$' and '$CF_n(p)$'

Step 6: Extract normal metric dispersal java class

Step 7: **else**

Step 8: Extract abnormal metric dispersal java class

Step 9: end if

Step 10: end for

Step 11: For extracted java class 'C'

Step 12: Apply generalized canonical correlative normal discriminant analysis

Step 13: Define number of sets si and sj

Step 14: Define mean value 'μ_{si}' and μ_{sj}

Step 15: Measure the generalized canonical correlation between the mean and software metrics '$c_{var}\,[\mu_s, m_i]$'

Step 16: Categorizes the software metrics into a particular set

Step 17: Return significant software metrics for different java classes

End

Algorithm 1 shows a process of the java class extraction and significant metric selection. First, Jarque–Bera stochastic test is applied for identifying the normal and abnormal dispersal java class. After the class extraction, significant software metrics are selected by applying a generalized canonical correlative normal discriminant

analysis. As a result, significant software metrics are identified and therefore it minimizing time utilization of the software faults forecast.

3.2 Iterative Dichotomize Linear Regressive Quadratic MilBoost-based software fault forecast

Subsequent to software metric chosen process, the software fault prediction is performed through Iterative Dichotomize Linear Regressive Quadratic MilBoost (Multiple Instance Learning) Ensemble learning. The proposed technique is an ensemble meta-algorithm which converts performance of weak classifier results to strong ones. Weak classifier is base classifier which not efficient to give precise software prediction results. On the converse, the proposed ensemble meta-algorithm is a strong classifier which gives accurately higher classification outcomes through integrating set of weak classification results. Main advantage of MilBoost boost technique is to learn the multiple instances (i.e. input samples) as well as gives precise classification outcomes and minimizes the error rate.

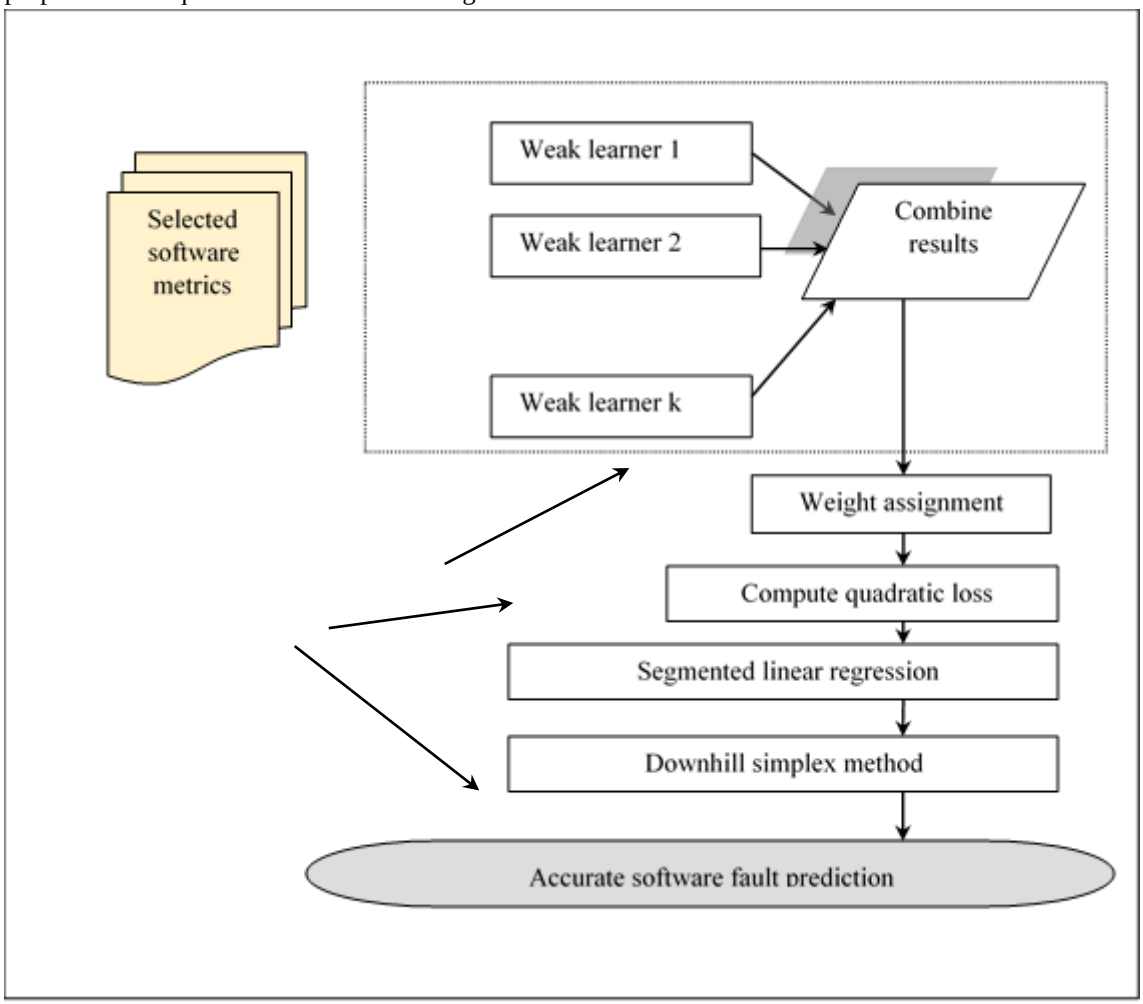

figure 4 block diagram of Iterative Dichotomize Linear Regressive Quadratic MilBoost

Figure 4 depicts the structure of the MIL Boosting technique for fault prediction. The ensemble boosting technique includes a training set $\{c_i, y\}$ where c_i represents an input (i.e. number of classes), 'y' denotes strong classification results. The proposed MIL Boosting technique uses a 'k' number of weak learners as Gibbs-Poston indexive ID3 (Iterative Dichotomiser 3) (GIID3) decision tree algorithm for classifying the java package class with defects and on defects based on the selected software metrics. The advantage of the ID3 (Iterative Dichotomiser 3) decision tree is to handle both numerical and categorical data. It also handles multi-output problems and the output classification outcomes are integrated to attain strong results.

Let us consider the extracted software metrics '$m_1, m_2, m_3, \ldots m_n$ ' and java package classes $c_1, c_2, \ldots c_m$. The ID3 (Iterative Dichotomiser 3) decision tree is applied to identify the software faults in the java package classes. The decision tree is build by root node, internal node, as well as leaf nodes. In tree, each root node carry outs "test" on input every branch indicates result of test, as well as every leaf node provides class label .

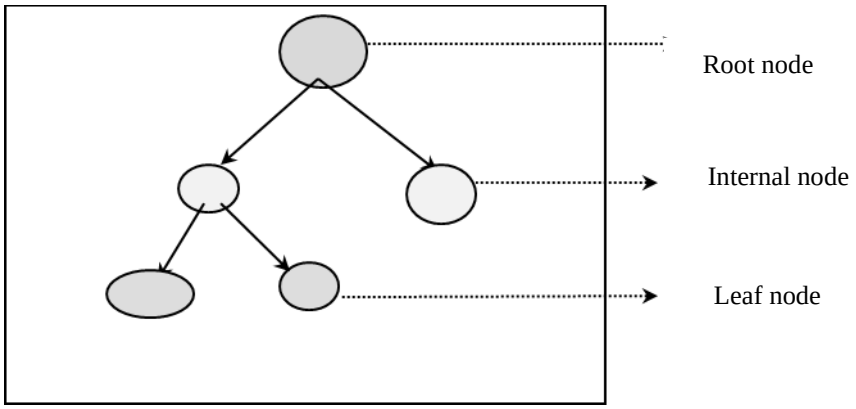

Figure 5 structure of ID3 decision tree

Figure 5 illustrates the structure of the ID3 decision tree with root, internal, as well as leaf nodes. For every iteration of algorithm, root node calculates similarity between the java package classes $c_1, c_2, \dots c_m$ and man of the particular class by using the Gibbs-Poston index. The Gibbs-Poston index is a statistical method for measuring the similarity between two variables. The index estimation is given below,

$$W \to I = Q\left[1 - \frac{\sum_{i=1}^{n}|c_i - m_{cj}|}{2n}\right] \quad (8)$$

Where, Q denotes the number of categories, c_i denotes an input java package class, m_{cj} denotes a mean of the particular categories. Here, two categories are initialized as defective and non-defective, 'n' indicates a number of java package class data samples, W indicates an output of weak learner results. The higher similarity value indicates that the java package classes are categorized into either defective or non-defective categories. Finally, the results are obtained at the leaf node of the decision tree.

The weak learner did not efficiently provide accurate classification outcomes. So, proposed boosting method combines weak learner results to enhance precision of software fault prediction. To attain strong classification outcomes, output of weak learners is summed as follows,

$$y = \sum_{i=1}^{k} W_i \quad (9)$$

Where 'y' indicates the ensemble classification output, W_i indicates weak classification results. For each weak classification result, the weight is initialized randomly.

$$y = \sum_{i=1}^{k} W_i * \delta \quad (10)$$

Where 'δ' indicates weight of weak learner results. After weight assignment, the error is measured among actual and predicted outcomes. Quadratic loss is used to validate the accuracy of the predictive model. It works by taking the difference among actual and forecasted outcomes as given below,

$$QL = \frac{1}{2} * [W_{act} - W_{pre}]^2 \quad (11)$$

Where QL denotes a quadratic loss calculated as variation among actual outcomes 'W_{act}' and predicted outcomes 'W_{pre}'. Depend on quadratic loss value, initial weight gets updated. Segmented linear regression is applied to select weak learner results through lesser error by setting threshold for minimizing the complexity of the fault prediction.

$$Z = \begin{cases} QL < T \ ; select\ weak\ learner \\ QL > T \ ; remove\ weak\ learner \end{cases} \quad (12)$$

Where Z denotes an output of segmented linear regression, QL denotes a quadratic loss, T indicates a threshold. As outcome , weak learner through lesser quadratic loss is selected for next process and removes the others.

After that, weight of weak learner updated. If weak learner performs accurate classification, after that initial weight minimized. Otherwise, weight enhanced. Depend on updated outcomes, ensemble technique gives final classification results with minimum loss with aid of downhill simplex technique. Downhill simplex technique is a numerical method that helps to discover lesser of objective function (i.e. quadratic loss). This aids to reduce wrong fault perdition resulting in it enhancing the precision and F-score in the classification process.

$$F = \arg\min\ QL\,(W) \quad (13)$$

From (13), F denotes an output of downhill simplex, $\arg min$ indicates argument of minimum function, QL represents Downhill simplex method, W and denotes a weak learner. Like this, every java package classes are properly classified through enhanced accuracy, precision, and lesser loss. Algorithmic process of classification is given below,

// **Algorithm 2: Iterative Dichotomize Linear Regressive Quadratic MilBoost based software fault prediction**
Input: extracted software metrics '$m_1, m_2, m_3, \dots m_n$ ' , java package classes $c_1, c_2, \dots c_m$
Output: Increase the software fault prediction accuracy
Begin **Step 1**: **For** each extracted metrics 'm_i' and Java package classes $c_1, c_2, \dots c_m$ **Step 2:** Construct a 'k' weak learners **Step 3:** Initialize the two output categories' defective and non defective (R_1, R_2) **Step 4:** **for each** category **Step 5:** Initialize the mean ' m_{cj}' **Step 6:** **End for** **Step 7:** Measure similarity coefficient 'I' **Step 8:** Classify the data into a particular category **Step 9**: **End for** **Step 10:** Combine a set of weak learners' results '$y = \sum_{i=1}^{k} W_i$ ' **Step 11:** **For** each W_i **Step 12:** Initialize the weight 'δ' **Step 13**: Calculate quadratic loss 'QL ' **Step 14**: Apply segmented regression **Step 15**: **if** $(QL < T)$ **then** **Step 16**: Select weak learner results **Step 17**: **else** **Step 18**: Remove the weak learners **Step 19: end if** **Step 20:** Apply downhill simplex method **Step 21:** Find weak learner with minimum error **Step 22: Obtain strong classification results** **End**

Algorithm 2 provides process of Iterative dichotomized linear regressive Quadratic MilBoost to enhance software fault prediction accuracy as well as precision. Initial, ensemble technique constructs 'k' set of Gibbs-Poston indexive ID3 (Iterative Dichotomiser 3) decision tree classifier. Initialize number of classes with mean value. After that Gibbs-Poston index similarity is measured among Java package classes and the mean. Depend on similarity value, Java package classes are categorized into a specific class. Then weak classification results are integrated and assigned weights. After that, quadratic loss is calculated for every weak learner's outcomes. Segmented regression selects the results with a minimum loss than the threshold and it is used for further process. Then the downhill simplex method is applied for finding weak learner results by lesser loss. This aids to enhance software fault prediction accuracy , precision.

4. Experimental Evaluation

Experimental setup of CADME and conventional methods HA-LCBS [1] DP-GCNN [2] are implemented in Java language and the dataset extracted from https://drive.google.com/drive/folders/101QbQ-TtQpyZa-APCFo4hCGAc_c-g6-Y . Different numbers of class data samples are collected ranging between 1000 and 10000 for software fault prediction. First, the number of classes is extracted from the java packages using Jarque–Bera stochastic test. After that, the significant software metrics are selected for minimizing the complexity of software prediction by using generalized canonical correlative normal discriminant analysis. Finally, the software fault prediction is performed by means of Iterative Dichotomize Linear Regressive Quadratic MilBoost for identifying defects or non-defects

4.1 Performance Results

Performance analysis of CADME and conventional methods HA-LCBS [1] DP-GCNN [2] are explained based on software fault prediction accuracy, precision, recall, F-measure, time complexity with respect to class data samples..

Software fault prediction accuracy: It calculated as ratio of a number of (i.e., class data samples) predicted positively or accurately divided by total number of class data samples. It is formulated as below.

$$SFPA = \sum_{i=1}^{n} \frac{C_{Pacc}}{C_i} * 100 \quad (14)$$

Where, $SFPA$' denotes a software fault prediction accuracy, C_i denotes a class samples . 'C_{Pacc}' denotes a class data samples predicted positively or accurately. It is measured in percentage (%).

Precision: Estimates the number of properly positive class data samples against total number of positive instances. It is mathematically stated as given below.

$$Precision = \frac{tr_p}{tr_p + fl_p} \quad (15)$$

Where, tr_p denotes a true positive,fl_p indicates false positive rate. It is measured in percentage (%).

Recall: Evaluates the number of correctly positive class data samples as well as negative class data samples. Precision is mathematically stated as given below.

$$Recall = \frac{tr_p}{tr_p + fl_n} \quad (16)$$

Where, tr_p denotes a true positive , fl_n indicates false negative. It is measured in percentage (%).

F-score: it is calculated as average of precisions and recall. It is estimated as below,

$$F - score = \left[2 * \frac{precision * recall}{precision + recall}\right] * 100 \quad \textbf{(17)}$$

F-score is measured in percentage (%).

Software fault prediction time: It is referred as amount of time utilized through algorithm for software fault prediction. It is calculated as given below,

$$SFPT = \sum_{i=1}^{n} C_i * [Time\ (SM)] \quad (18)$$

Where, $SFPT$ denotes a Software fault prediction time , C_i indicates number of class data samples, $Time\ (SM)$ represent time for detecting software fault prediction. It is measured milliseconds (ms).

Table 1 Software fault prediction accuracy

Number of class data samples	Software fault prediction accuracy (%)		
	CADME	HA-LCBS	DP-GCNN
1000	99.2	97	98.2
2000	98.25	96	97.25
3000	97	92.05	95
4000	95.75	89.35	93.25
5000	95	87.55	91.4
6000	94.16	85.25	91.66
7000	93.57	83	91.42
8000	93.12	79.15	91.5
9000	92.777	75.35	91.11
10000	92.5	73	90.7

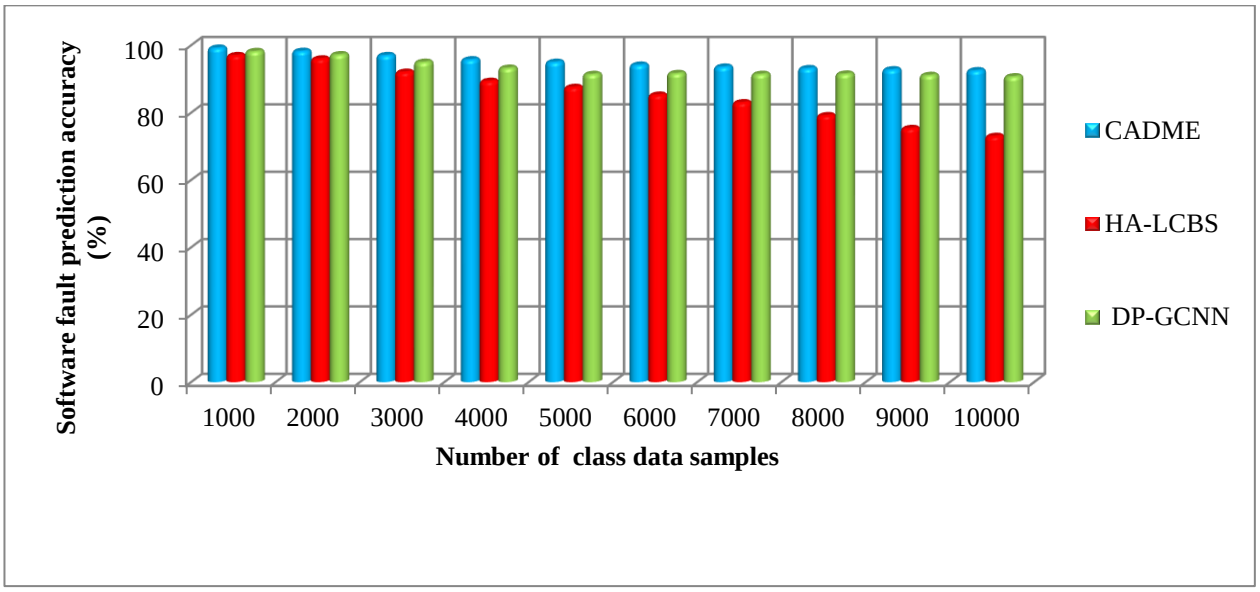

Figure 6 Graphical representation of Software fault prediction accuracy

Table 1 , figure 6 depicts performance results of SFPA with number of class data samples . Among three methods, proposed CADME technique gives enhanced accuracy results when compared to conventional methods. Let us assume, the number of data samples is considered as 1000 for estimating the accuracy. By using CADME, the accuracy was found to be 99.2% and HA-LCBS [1] ,DP-GCNN [2] was 97% and 98.2%. Likewise, various performance outcomes are examined with number of data samples. Finally, observed results of proposed CADME are compared to conventional methods. average of ten comparisons denotes that performance of the accuracy with CADME technique is enhanced by 12% and 2% than the [1] ,[2] respectively. This improvement is achieved through the Iterative dichotomized linear regressive Quadratic MilBoost. The ensemble classification technique constructs the decision tree classifier and measures the similarity among class data samples as well as mean of the specific class. Then weak learner outcomes are integrated to create a strong output by minimizing quadratic loss. This helps to improve SFPA.

Table 2 Precision

Number of class data samples	**Precision** (%)		
	CADME	**HA-LCBS**	**DP-GCNN**
1000	99.49	97.87	98.96
2000	98.97	97.32	98.44
3000	97.94	94.89	96.42
4000	96.8	93.22	94.60
5000	96.79	90.58	95.13
6000	96.39	90.19	95.02
7000	96.12	89.38	94.36
8000	95.91	84.67	95.18
9000	95.75	81.81	94.98
10000	95.62	79.72	94.51

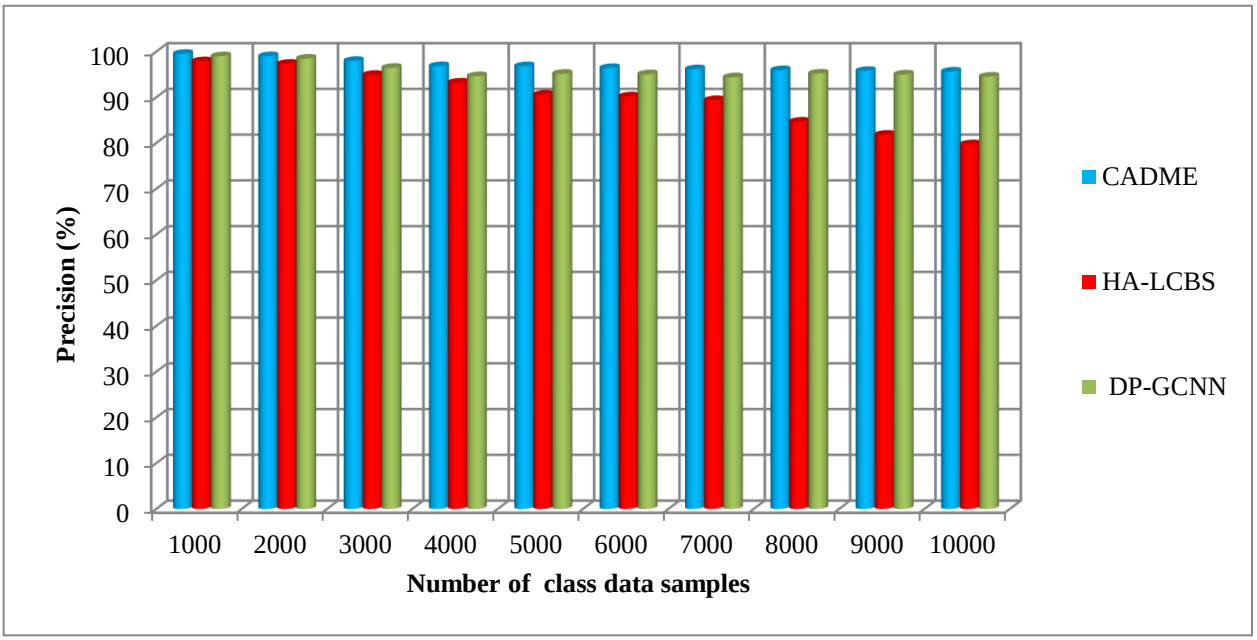

Figure 7 Graphical representation of precision

Table 2 , figure 7 depicts experimental evaluation results of precision using CADME technique, HA-LCBS [1] DP-GCNN [2]. Examined outcomes denote that the overall performance results of precision using the proposed CADME technique were found to be increased when compared to conventional methods. Let us assume 1000 numbers of class data samples in initial iteration and the observed precision outcomes with CADME technique were found to be 99.49 % whereas the performance of precision results of [1] and [2] was found to be 97.87 % and 98.96% respectively. Likewise, dissimilar performance outcomes are examined with different number of class data samples. Accordingly, average performance outcomes denote performance of precision using the CADME technique by 8% and 1% than the [1] ,[2]. This is due to application of an improved Iterative dichotomized linear regressive Quadratic MilBoost ensemble classifier. Ensemble classifier precisely classifies data samples to defective or non-defective and obtains strong classification results by applying the downhill simplex method. The downhill simplex method is used to discover weak learner results by lesser quadratic loss. This aids to enhance true positives and minimize false positives.

Table 3 Recall

Number of class data samples	Recall (%)		
	CADME	HA-LCBS	DP-GCNN
1000	99.69	98.92	99.16
2000	99.23	98.37	98.69
3000	98.96	96.29	98.18
4000	98.64	94.28	98.04
5000	97.84	93.90	95.34
6000	97.27	92.92	95.73
7000	96.87	89.38	96.17
8000	96.57	88.23	95.44
9000	96.34	84.37	95.22
10000	96.15	83.09	95.24

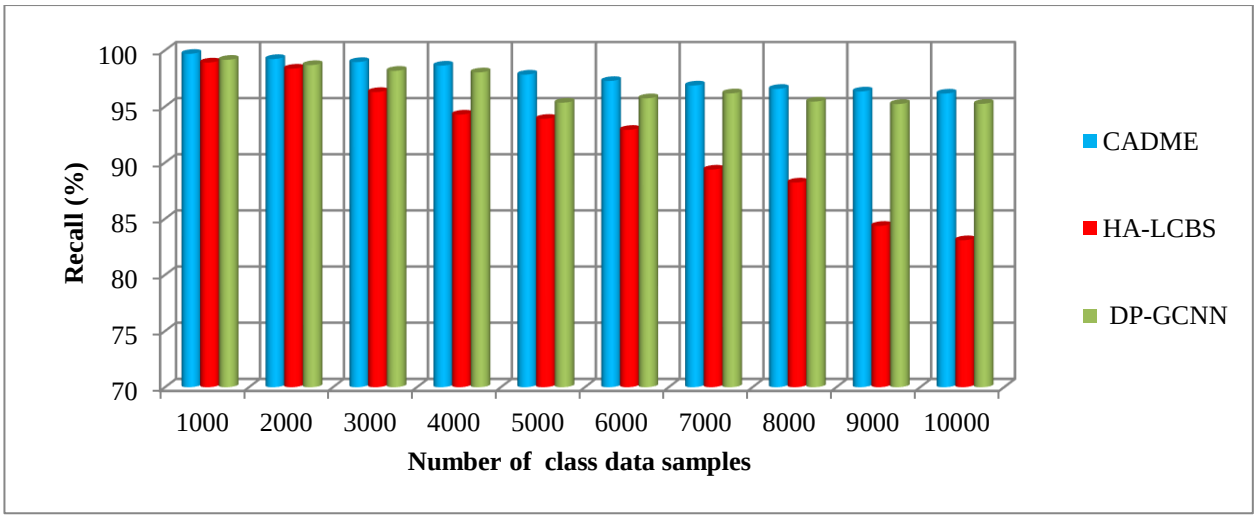

Figure 8 Graphical representation of recall

Table 3 , figure 8 illustrate overall performance of recall using number of numbers of class data samples. For every method, ten various performance results are examined and the outcomes shown in figure 8. Performance of recall by

CADME method is higher when compared to [1] [2]. This is because of CADME technique increases true positive rate as well as minimizes the false negative in software fault prediction with the help of an Iterative dichotomized linear regressive Quadratic MilBoost ensemble classifier.

Examined performance outcomes of CADME technique are compared to results of conventional methods. Overall comparison outcomes denote that performance of recall using CADME method is considerably enhanced by 7% and 1% than the [1],[2].

Table 4 F-score

Number of class data samples	F-score (%)		
	CADME	HA-LCBS	DP-GCNN
1000	99.58	98.39	99.05
2000	99.09	97.84	98.56
3000	98.44	95.58	97.29
4000	97.71	93.74	96.28
5000	97.31	92.21	95.23
6000	96.82	91.53	95.37
7000	96.49	89.38	95.25
8000	96.23	86.41	95.30
9000	96.04	83.07	95.09
10000	95.88	81.37	94.87

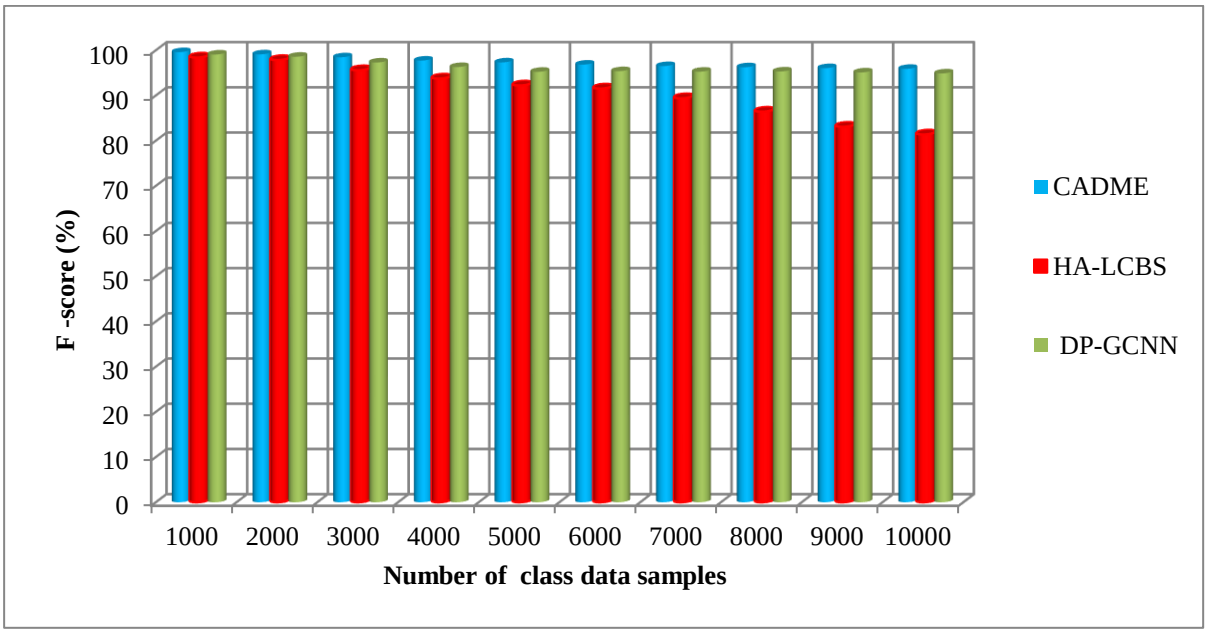

Figure 9 Performance results of F-score

Table 4 , figure 9 demonstrate overall performance outcome of F-score against number of data samples. F-score in the fault prediction is estimated depend on precision and recall. Overall examined result indicates which performance results of F-score with CADME technique are improved when compared to conventional methods. This is due to application of the proposed ensemble classifier increasing performance of precision and recall in software fault prediction. Lastly, overall performances of CADME technique are compared to outcomes of conventional methods. Average of ten comparisons represents which entire performance of F-score is considerably enhanced by 7% and 1% than the [1], [2] respectively.

Table 5 Software fault prediction time

Number of class data samples	Software fault prediction time (ms)		
	CADME	HA-LCBS	DP-GCNN
1000	13	19	16
2000	16.4	28.15	22
3000	21	35.45	28.5
4000	26	42.55	34
5000	31	59.35	42.5
6000	42.6	75.35	52.8
7000	47.6	85.95	63
8000	56.8	105.25	88
9000	67.5	125.25	108
10000	76	140.35	120

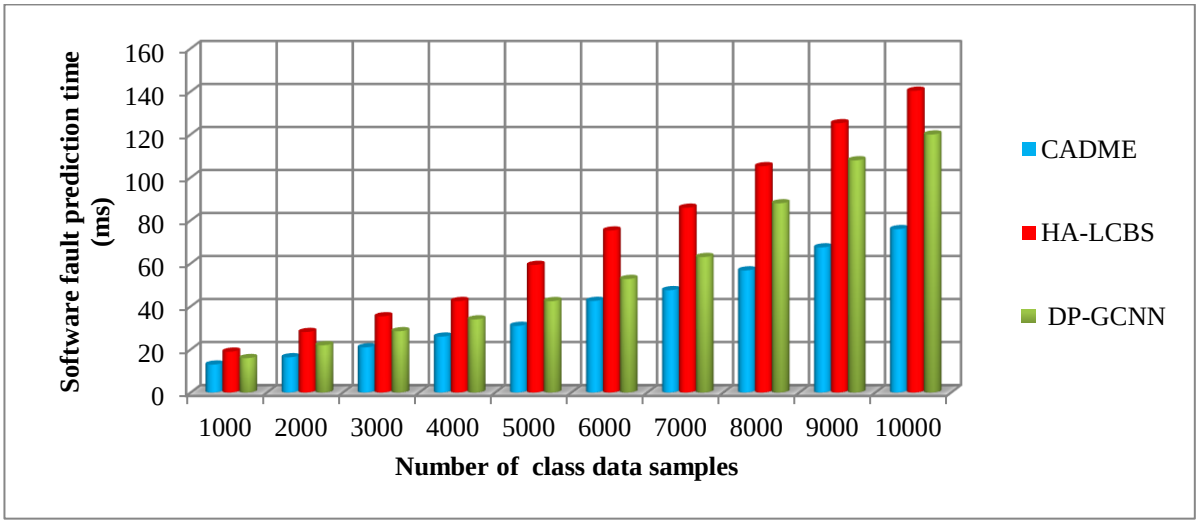

Figure 10 Performance results of Software fault prediction time

Performance analysis of $SFPT$ versus the number of class data samples is shown in table 5 and figure 10. From the figure, a software fault prediction time gets enhanced as improving the number of class data samples. Examined outcomes represent that overall performance of the proposed CADME technique decreases the $SFPT$ when compared to existing methods. Let us assume 1000 class data samples for performing experiments. Entire performance of $SFPT$ by CADME technique was found to be $13ms$. In addition, the software fault prediction time using HA-LCBS [1], and DP-GCNN [2] was found to be $19ms$ and $16ms$. For every method, a variety of performance outcomes is examined with dissimilar counts of input data samples. Overall performance outcomes designate that the $SFPT$ is significantly minimized by 43% and 27% than the existing [1] [2] respectively. This is because of the metric dispersal class extraction and significant software metric selection. First, the Jarque–Bera stochastic test is used for extracting java classes from the packages. Followed by, significant software metrics selected by applying the generalized canonical correlation. Depend on correlation measure, relevant metrics are chosen. Selected metrics are given to the proposed ensemble classifier for fault prediction. Ensemble classifier eliminates weak learner outcomes which have superior quadratic loss than the threshold by applying segmented linear regression. This process reduces time complexity of fault prediction.

5. Conclusion

Fast growth of larger as well as further multifaceted software scheme needs rapid and precise methods for

identifying possible defects in source code of software. Therefore, a novel CADME technique is designed for detecting software defects accurately by lesser time consumption. In the CADME technique, metric dispersal class extraction is performed by means of the Jarque–Bera stochastic test. Followed by, generalized canonical correlative normal discriminant analysis is performed to identify the significant metrics for fault prediction. These two processes of the CADME technique help to reduce time utilization of software fault prediction. Finally, the ensemble technique called Iterative Dichotomize Linear Regressive Quadratic MilBoost is developed for accurate defective or non-defective software data samples with lesser misclassification rate. Comprehensive experimental measurement is done by different parameters . Overall observed results indicate that the presented CADME method attains enhanced accuracy with lesser time for software defect prediction than the conventional methods.

References

[1] Ayad Tareq Imam, Basma R. Al-Srour, Aysh Alhroob, "The automation of the detection of large class bad smell by using genetic algorithm and deep learning", Journal of King Saud University – Computer and Information Sciences, Elsevier, Volume 34, Issue 6, 2022, Pages 2621-2636. https://doi.org/10.1016/j.jksuci.2022.03.028

[2] Lucija Šikić, Adrian Satja Kurdija, Klemo Vladimir, Marin Šilić, "Graph Neural Network for Source Code Defect Prediction", IEEE Access, Volume 10, 2022, Pages 10402 – 10415. DOI: 10.1109/ACCESS.2022.3144598

[3] Sweta Mehta & K. Sridhar Patnaik, "Improved prediction of software defects using ensemble machine learning techniques", Neural Computing and Applications, Springer, Volume 33, 2021, Pages 10551–10562. https://doi.org/10.1007/s00521-021-05811-3

[4] Inderpreet Kaur And Arvinder Kaur, "A Novel Four-Way Approach Designed With Ensemble Feature Selection for Code Smell Detection", IEEE Access, Volume 9, 2021, Pages 8695 – 8707. DOI: 10.1109/ACCESS.2021.3049823

[5] Umamaheswara Sharma Bhutamapuram, Ravichandra Sadam, "With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique", Journal of King Saud University - Computer and Information Sciences, Elsevier, 2021, Pages 1-17. https://doi.org/10.1016/j.jksuci.2021.09.010

[6] Fahad H. Alshammari, "Software Defect Prediction and Analysis Using Enhanced Random Forest (extRF) Technique: A Business Process Management and Improvement Concept in IOT-Based Application Processing Environment", Mobile Information Systems, Hindawi, Volume 2022, September 2022, Pages 1-11. https://doi.org/10.1155/2022/2522202

[7] Haonan Tong,Bin Liu,and Shihai Wang, "Kernel Spectral Embedding Transfer Ensemble for Heterogeneous Defect Prediction", IEEE Transactions on Software Engineering , Volume 47, Issue 9, 2021, Pages 1886 – 1906. DOI: 10.1109/TSE.2019.2939303

[8] Zhen Li , Tong Li , YuMei Wu , Liu Yang , Hong Miao , and DongSheng Wang, "Software Defect Prediction Based on Hybrid Swarm Intelligence and Deep Learning", Computational Intelligence and Neuroscience, Hindawi, Volume 2021, December 2021, Pages 1-17. https://doi.org/10.1155/2021/4997459

[9] ShivaniJain, AnjuSaha, "Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection", Science of Computer Programming, Elsevier, Volume 212, 2021, Pages 1-34. https://doi.org/10.1016/j.scico.2021.102713

[10] Amal Alazba and, Hamoud Aljamaan, "Code smell detection using feature selection and stacking ensemble: An empirical investigation", Information and Software Technology, Elsevier,

[11] Volume 138, 2021, Pages 1-14. https://doi.org/10.1016/j.infsof.2021.106648

[12] Tushar Sharma, Vasiliki Efstathiou, Panos Louridas, Diomidis Spinellis, "Code smell detection by deep direct-learning and transfer-learning", Journal of Systems and Software, Elsevier, Volume 176, 2021, Pages 1-25. https://doi.org/10.1016/j.jss.2021.110936

[13] Seema Dewangan, Rajwant Singh Rao, Alok Mishra and Manjari Gupta, "Code Smell Detection Using Ensemble Machine Learning Algorithms", Applied Science, Volume 12, 2022, Pages 1-22. https://doi.org/10.3390/app122010321

[14] Hafiz Shahbaz Munir, Shengbing RenID*, Mubashar Mustafa, Chaudry Naeem Siddique, Shazib Qayyum, "Attention based GRU-LSTM for software defect prediction", PLoS ONE, Volume 16, Issue 3, 2021, Pages 1-19. https://doi.org/10.1371/journal.pone.0247444

[15] Pasquale Ardimento, Lerina Aversano, Mario Luca Bernardi, Marta Cimitile, Martina Iammarino, "Temporal convolutional networks for just-in-time design smells prediction using fine-grained software metrics", Neurocomputing, Elsevier, Volume 463,

2021, Pages 454-471. https://doi.org/10.1016/j.neucom.2021.08.010

[16] Jiaxi Xu, Fei Wang, Jun Ai, "Defect Prediction With Semantics and Context Features of Codes Based on Graph Representation Learning", IEEE Transactions on Reliability, Volume 70, Issue 2, 2021, Pages 613 – 625. DOI: 10.1109/TR.2020.3040191

[17] Hao Wang, Weiyuan Zhuang, and Xiaofang Zhang, "Software Defect Prediction Based on Gated Hierarchical LSTMs", IEEE Transactions on Reliability, Volume 70, Issue 2, 2021, Pages 711 – 727. DOI: 10.1109/TR.2020.3047396

[18] Xiao Yu, Jin Liu , Jacky Wai Keung , Qing Li , Kwabena Ebo Bennin, Zhou Xu, Junping Wang , and Xiaohui Cui, "Improving Ranking-Oriented Defect Prediction Using a Cost-Sensitive Ranking SVM", IEEE Transactions on Reliability , Volume 69, Issue 1, 2020, Pages 139 – 153. DOI**:** 10.1109/TR.2019.2931559

[19] Aleksandar Kovaˇcevi´, Jelena Slivka, Dragan Vidakovi , Katarina-Glorija Gruji ,Nikola Luburi, Simona Proki, Goran Sladi, "Automatic detection of Long Method and God Class code smells through neural source code embeddings", Expert Systems with Applications, Elsevier, Volume 204, 2022, Pages 1-18. https://doi.org/10.1016/j.eswa.2022.117607

[20] Mouna Hadj-Kacem and Nadia Bouassida, "A multi-label classification approach for detecting test smells over java projects", Journal of King Saud University - Computer and Information Sciences, Elsevier, 2021, Pages 1-10. https://doi.org/10.1016/j.jksuci.2021.10.008

[21] Mohamed Maddeh, Sarra Ayouni , Sultan Alyahya, And Fahima Hajjej, "Decision tree-based Design Defects Detection", IEEE Access , Volume 9, 2021, Pages 71606 – 71614, DOI: 10.1109/ACCESS.2021.3078724