Stochastic Embedded Probit Regressive Reweight Boost Classifier for Software Quality Examination

Noor Ayesha, Yethiraj N G

Abstract*: In software development, Software quality analysis plays a considerable process. Through the software testing, the quality analysis is performed for efficient prediction of defects in the code. Due to the complicated structure of software projects, code examination has become a demanding issue that has to be addressed at the initial stage of testing for achieving the quality improved results. In order to resolve these issues, the Stochastic Gaussian Neighbor Embedding based Probit Regressive Reweight Boost Classification (SGNE-PRRBC) is introduced for accurate quality prediction system through code examination proficient system. The SGNE-PRRBC technique considers the number of program files as input for software quality analysis through feature selection and classification. Initially, the number of program files is taken from the dataset (DS). After collecting the files, the Gaussian distributive stochastic neighbor embedding technique choose the features (i.e. code metrics) based on the distance similarity. With the assist of Pearson correlative probit regressed reweight boost technique, the classification of program files is performed. The boosting algorithm creates 'm' number of weak classifiers i.e. Pearson correlative probit regression to categorize the input program files as normal or defected through analyze the source codes and chosen metrics. After that, the weak learners results are combined into strong through minimizing the out of sample error with gradient descent function. This enhances the accuracy of quality prediction and lessens the false positive rate (FPR). Experimental analysis is performed with various metrics namely accuracy, FPR and computation time (CT) with number of program files. Experimental results evident that the SGNE-PRRBC technique achieves better performance in terms of accuracy, CT and FPR as compared to the conventional methods.

***Keywords:* Software quality analysis, software testing, software metrics, Gaussian distributive stochastic neighbor embedding technique, feature selection, Pearson correlative probit regressed reweight boosting, classification.**

I. INTRODUCTION

Software quality analysis is a significant process due to its effects on the various aspects of the system such as functionality, reliability, availability, compatibility, and maintainability in software development projects. The most complicated issues in software management are the software quality analysis through the testing. Therefore several models have been designed for software quality testing, each with its own unique merits and demerits, with respect to the environment and the area in which it is to be applied. By applying the machine learning techniques, several limitations found in the software testing using existing algorithms were minimized resulting in the improvement of the software quality.

A fuzzy-filtered neuro-fuzzy framework was developed in [1] for detecting the faults of software projects to enhance the software quality. However, the inter-version and inter-project fault prediction performance was not enhanced with lesser time. In [2], a gradual relational association rule mining and artificial neural networks (HyGRAR) was designed to discover the defective and non-defective software entities. The designed method failed to use an efficient machine learning technique for accurate defective prediction with minimum time. By using the open-source software, several machine learning models were developed in [3] for software quality prediction. But the performance of software quality prediction time remained unaddressed.

In [4], an improved correlation oversampling method was designed to identify the software defect with class imbalance learning. The designed method was not improved the overall accuracy. In [5], Defect Prediction using Attention-based Recurrent Neural Network (DP-ARNN) was designed with significant features. But the performance of defect prediction was not improved while considering the more projects.

A CNN-IndRNN model was developed in [6] to identify the open-source software defects for enhancing the software quality with dynamic features. But, the time was not reduced. For software defect prediction, Stacked denoising autoencoders (SDAEs) and two-stage ensemble learning were introduced in [7] with feature learning. But the error rate of defect prediction was not effectively minimized. A Markov chain and a systematic framework were developed in [8] for software quality analysis by identifying the defects. But the framework failed to perform the feature selection.

The Fuzzy Analytical Hierarchy Process (FAHP) was developed in [9] using multi-objective decision-making method to reliably evaluate the software quality. However, software quality prediction accuracy was not improved. With the neural forest classifier, a Just-In-Time Software Defect Prediction (JIT-SDP) method was presented in [10]. But, it does not perform the other software engineering tasks namely defect prediction and software code review.

Revised Manuscript Received on November 11, 2019.

***** Correspondence Author

NOOR AYESHA, Research Scholar, Dept. of Computer Science, Bharathiar University, Coimbatore, India.

YETHIRAJ N G, Assistant Professor, Dept of Computer Science, Maharani's Science College for Women, Bengaluru, India.

1.1 Contribution of the research work

The major issues of conventional methods are addressed by introducing a novel SGNE-PRRBC technique. The contribution of SGNE-PRRBC technique is summarized as below,

- ❖ To enhance the accuracy of software quality prediction, the SGNE-PRRBC technique is proposed. This contribution is achieved by feature selection and classification. The SGNE-PRRBC technique uses Pearson correlative probit regressed reweight boosting technique for classifying the program files by analyzing the source codes and software metrics through the Pearson correlation. Based on the correlation results, the probit regression function classifies the given application program files as normal or defected. The strong classification results with minimal error are attained by combining the results.
- ❖ To lessen the FPR, the SGNE-PRRBC technique utilizes the gradient descent function to reduce the result of sample error of weak classifiers. The SGNE-PRRBC technique accurately finds the defects in a given application program.
- ❖ To lessen the CT, the Gaussian distributive stochastic neighbor embedding technique is presented. The technique selects suitable source code metrics for analyzing the software programs after feature selection. The feature selection is performed with Gaussian distributive function and identifying the relevant metrics for code examination.

1.2 Structure of the paper

The article is ordered into five different sections. Section 2 reviews the related works. Section 3 briefly describes the proposed methodology of SGNE-PRRBC for improving the accuracy of software quality. Section 4 provides information on the experimental evaluation along with parameter settings. In section 5, the experimental outcomes and comparative analysis are presented using various performance metrics. Finally, the last section ends the work by the conclusion.

II. RELATED WORKS

A Learning Deep Feature Representation (LDFR) was developed in [11] for identifying the software defects. The model selects the top-level feature representation but failed to minimize the error of defect prediction. Atomic Class-Association Rule Mining (ACAR) technique was developed in [12] for predicting the software defects by analyzing the interaction between features. However, the software prediction time was higher.

Depending on the software metrics, a Layered Recurrent Neural Network (L-RNN) was designed in [13]. But the performance of different parameters remained unaddressed. Software Fault Detection and Recovery (SFDR) were developed in [14] for identifying the fault that occurs in the software. The designed method failed to use data mining techniques for efficient fault recovery.

An extreme learning machine with different kernel approaches was developed in [15] for fault prediction with software code metrics. The designed methods were not achieving higher accuracy in fault prediction. Hybrid Search Based Algorithms (HSBA) were designed in [16] for predicting the software faults using metrics. But the designed algorithm has higher time complexity. A novel method was developed in [17] to improve the quality assurance and software ecosystems using quality attributes. However, code review analyzes were not performed.

In [18], a Cross-project semisupervised defect prediction (CSDP) was carried out for software quality prediction. The method does not enhance the robustness for various kinds of heterogeneous cross-project data. The machine learning techniques were introduced in [19] for automatically identifying the feasible source code defects. The designed technique failed to automatically detect more defect patterns. For software defect prediction, a Weighted Least Squares Twin Support Vector Machine (WLSTSVM) was designed in [20]. The method does not choose the parameters to enhance the software defect prediction performance.

The issues of conventional methods are resolved by proposing the efficient technique, called SGNE-PRRBC. The brief description of SGNE-PRRBC technique is presented in the following sections.

III. METHODOLOGY

In software engineering, software testing is the process to discover the failures in source code lines and enhance the software quality. Software quality analysis is a significant concern in software testing with the code metrics. The different methods have been developed to predict software defects for improving software quality. An efficient technique called the SGNE-PRRBC technique is developed by using software metrics to enhance the quality of a software solution during the development process.

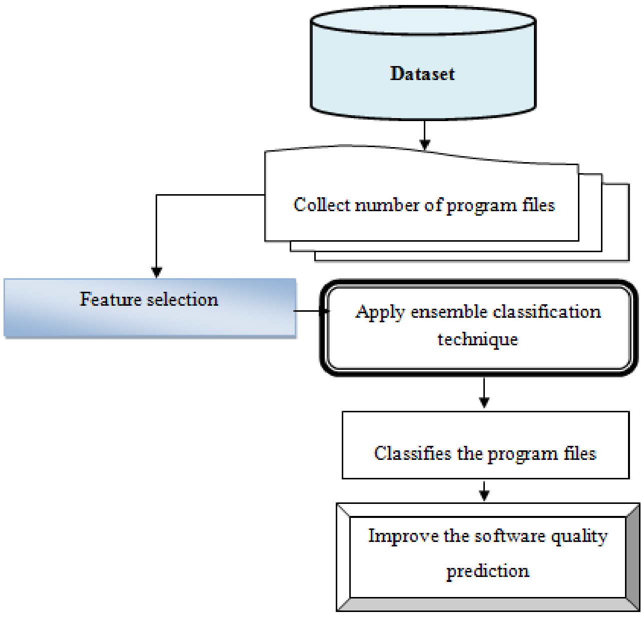
Dataset
Collect number of program files
Feature selection
Apply ensemble classification technique
Classifies the program files
Improve the software quality prediction

Figure 1architecture diagram of proposed SGNE-PRRBC technique

The architecture of SGNE-PRRBC technique is depicted in figure 1 that includes of two processes namely feature selection and classification. Initially, the number of program files $p_{f1}, p_{f2}, p_{f3}, \ldots. pf_n$ are collected from the DS and each program file comprises software codes $Sc_1, Sc_2, Sc_3, \ldots. Sc_n$ for performing the software quality analysis. After that, the features are selected for classifying the given program files as normal or defected. Here the features are represented as a software metrics. With the help of selected software metrics, the program files are classified using ensemble classification techniques. As a result, proposed the SGNE-PRRBC technique improves the accuracy of review or testing. The process of SGNE-PRRBC technique is explained in the next sections.

1.1 Gaussian distributive stochastic neighbor embedding based feature selection

In SGNE-PRRBC technique, the first process is the feature selection (i.e. metric selection) for classifying the program files in order to enhance the software quality. By applying the Gaussian distributive stochastic neighbor embedding technique, the feature selection is performed. It is a machine learning technique used for obtaining similar metrics based on the Gaussian distribution function. A software metric is a measure of software features used to identify the quality of the softwares. In general, there are several metrics are available for testing the software program files. The testing is performed with the number of metrics that causes a higher complexity of quality prediction. Therefore, the proposed technique selects more similar metrics for quality analysis. Let us consider the number of metrics $M_i = \{m_1, m_2, m_3, \ldots. m_n\}$. Then the Gaussian distributive function is applied for finding similar features as follows,

$$y = \frac{\exp -\frac{1}{2\sigma^2}\left(\|M_i - b_j\|^2\right)}{\sum \exp -\frac{1}{2\sigma^2}\left(\|M_i - b_j\|^2\right)} \qquad (1)$$

From (1), y indicates a output, M_i indicates number of metrics, b_j indicates a objective i.e. software quality analysis , σ indicates a deviation. Gaussian distributive function calculates the distance similarity among M_i and b_j with Euclidean distance $\|M_i - b_j\|$. Based on the distance measure, the features which are closer to the objectives is selected and distant features are removed. At last, the relevant metrics are chosen for classification which enhance the accuracy and minimize the CT of quality prediction.

1.2 Pearson correlative probit regressed reweight boosting technique

The proposed SGNE-PRRBC technique uses the Pearson correlative probit regressed reweight boosting technique to perform the program file classification with the selected software metrics. The Boosting is an ensemble classification algorithm that converts the results of weak learner results into strong ones. Weak learner is a base classifier that does not provide accurate classification for software quality prediction. By combining the results of weak classifier, the strong learner is also a classifier which enhances the base learners performance. Therefore, the SGNE-PRRBC technique utilizes ensemble classification algorithm for software code quality prediction with the various metrics. Figure 2 depicts the structure of ensemble learning algorithm.

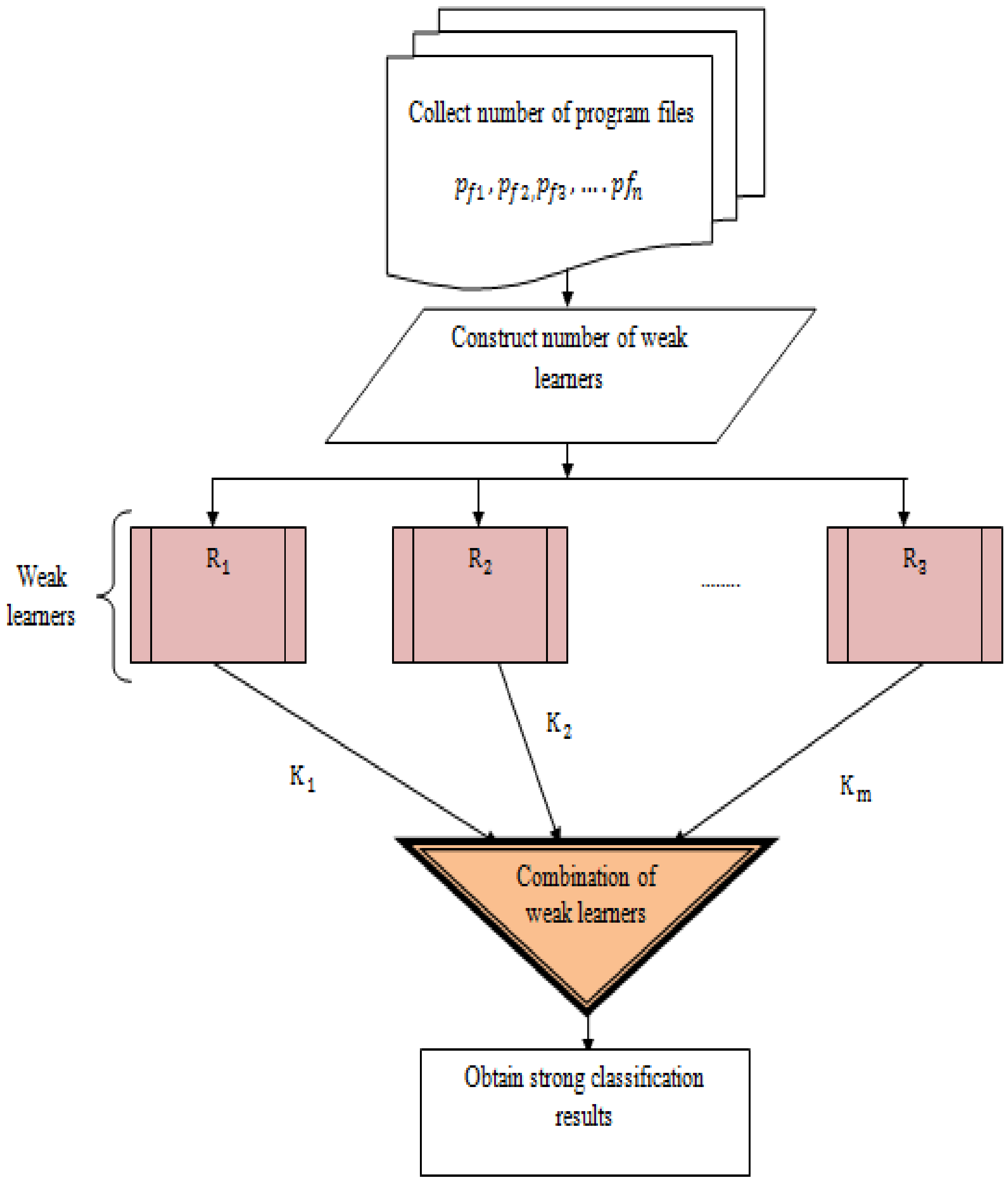
Collect number of program files
$p_{f1}, p_{f2}, p_{f3}, \dots, p_{fn}$
Construct number of weak learners
Weak learners
R_1
R_2
....
R_3
K_1
K_2
K_m
Combination of weak learners
Obtain strong classification results

Figure 2 structure of Pearson correlative probit regressed reweight boosting

Figure 2 depicts the structure of the Pearson correlative probit regressed reweight boosting ensemble classifier. Let us assume the training sets $\{X_i, Z_i\}$ where X_i indicates a input i.e. $p_{f1}, p_{f2}, p_{f3}, \dots, p_{fn}$ comprises a source code $Sc_1, Sc_2, Sc_3, \dots Sc_n$ and Z_i indicates the final ensemble classification results. Ensemble classifier creates an empty set of 'm' weak learners $R_1, R_2, \dots, R_n$ with number of $p_{f1}, p_{f2}, p_{f3}, \dots, p_{fn}$ and selected features. The proposed ensemble technique uses the weak learner as a probit regression to classify the program file as a correct or defected code.

The regression is the machine learning technique that performs the statistical processes for measuring the relationships between two variables (i.e. source code and software metrics). The regression function uses the probit model i.e. **prob**ability + un**it** where the output falls into a specific one of the categories '0' or '1'. Hence the regression function is called a binary classification model. The relationships between two variables are discovered by using the regression function with the Pearson correlation.

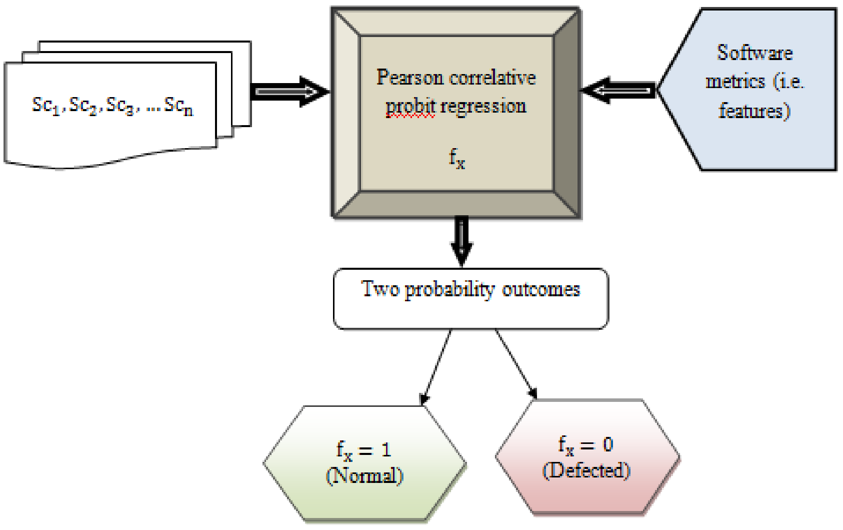
$Sc_1, Sc_2, Sc_3, \ldots Sc_n$
Pearson correlative probit regression
f_x
Software metrics (i.e. features)
Two probability outcomes
$f_x = 1$ (Normal)
$f_x = 0$ (Defected)

Figure 3 Pearson correlative probit regression-based classification

Figure 3 illustrates a Pearson correlative probit regression for classifying the program file is normal or defected based on the set of metrics. The regression function receives the input of source codes $S_{ci} = Sc_1, Sc_2, Sc_3, \ldots Sc_n$ and performs the analysis with the set of metrics $M_i = \{m_1, m_2, m_3, \ldots . m_n\}$ with the help of correlation measure. The correlation is measured using below mathematical expression,

$$\rho_f = \frac{(\sum S_{ci} M_i) - (\sum S_{ci})(\sum M_i)}{\sqrt{[\sum S_{ci}{}^2 - (\sum S_{ci})^2][\sum M_i{}^2 - (\sum M_i)^2]}} \quad (2)$$

In (2), ρ_f indicates a Pearson correlation coefficient, S_{ci} symbolizes source code, M_i symbolizes a metrics, $\sum S_{ci} M_i$ indicates a sum of product of paired score, $\sum S_{ci}$ is the sum of S_{ci} score, $\sum S_{ci}{}^2$ is the sum of squared S_{ci} score, $\sum M_i{}^2$ is the sum of squared score of $\sum M_i$. The Pearson correlation coefficient offers the output value between -1 and +1 where '-1 *to* 0' denotes the negative correlation and 0 *to* $+1$' denotes the positive correlation. The regression function returns '1' when the positive correlation is appeared. Otherwise, it returns '0'.

$$f_x = \begin{cases} 1 ; & \text{program file is normal} \\ 0; & \text{program file is defected} \end{cases} \quad (3)$$

From (3), f_x symbolizes the probit regression function. Positive correlation among the source code and metrics denotes '1' that the software program is normal and it provides expected results. Here, '0' denotes a software program not matched with the expected results. Thus, the regression function significantly finds the defect occurrence on the source code line of the input software program. In this way, all the program files are classified as normal or defected. However, the weak learner output includes some training error that affects the classification performance. To enhance the classification accuracy, the boosting classifier combines the outcomes of weak learners.

$$Z = \sum_{i=1}^{m} K_i (x) \quad (4)$$

From (4), Z indicates a output of ensemble classifier, $K_i (x)$ indicates the output of weak classifier results. After that, the ensemble classifier assigns similar weight to each weak learner. The weighted sums of weak learners are formalized as below,

$$Z = \sum_{i=1}^{m} \mu_t * K_i (x) \quad (5)$$

Where, μ_t represents the weight assigned to weak classification results. The weight is random numerical values. After assigning similar weight to each weak classifier, the out of sample error is calculated depends on the difference among expected and empirical error (i.e. observed error). The out of sample error is mathematically calculated using the given formula.

$$O_E = \{E_{ex} - E_{em}\} \quad (6)$$

Where, O_E denotes an out of sample error of weak classifiers, E_{ex} represents the expected error, E_{em} is the empirical error. Depends on the error rate, the weight of each weak learner is readjusted known as re-weighting. Hence the name of boosting is called as reweight boosting classifier. The misclassified input obtains higher weight and has the lesser weight if the weak learner correctly classified the program files. The reweighted sum of weak learner are obtained as follows,

$$Z = \sum_{i=1}^{m} \mu_t{}^r * K_i (x) \quad (7)$$

From (7), Z is an outcome of strong classification results, $\mu_t{}^r$ indicates an updated weight of weak learner $K_i (x)$. By applying the gradient descent function, the ensemble classifier minimizes the out of sample error for achieving higher classification accuracy.

$$g(x) = \arg\min O_E \ (K_i (x)) \quad (8)$$

From (8), $g(x)$ indicates a gradient descent function, $arg\ min$ is the argument of minimum, O_E is an out of sample error of weak learners $K_i\ (x)$. The final output of ensemble classification results lessen the error rate and enhance the classification results. By using an ensemble classification algorithm, the proposed technique addresses the classification issues in code examination proficient system. It offers efficient results for classify the program files as correct and defected. Therefore, the defected codes are rewritten and enhance the quality of the software program by efficient prediction of defects in the codes.

```
Input: Dataset, software programs $p_{f1}, p_{f2}, p_{f3}, \dots p_{fn}$, source codes $S_{ci} = Sc_1, Sc_2, Sc_3, \dots Sc_n$
Output: Improve the accuracy of software quality prediction
Begin
  1.  for each metric $M_i$
  2.     Measure similarity '$y$' with Gaussian distributive function
  3.     Select more relevant metrics
  4.     Collect a number of software programs $p_{f1}, p_{f2}, p_{f3}, \dots p_{fn}$
  5.      Construct 'm' set of weak learners $R_1, R_2, R_3, \dots R_m$
  6.      for each source code $S_{ci}$
  7.        for each selected metrics $M_i$
  8.          Measure the correlation $\rho_f$
  9.          if ($\rho_f = 0$ to $+1$) then
  10.           $f_x$ returns '1'
  11.           source code is classified as normal
  12.         else
  13.           $f_x$ returns '0'
  14.           source code is classified as defected
  15.       end if
  16.     end for
  17. end for
  18.    Combine all the weak learner's results $Z = \sum_{i=1}^{m} K_i(x)$
  19.    For all $K_i(x)$
  20.      Assigns the similar weight '$\mu_t$'
  21.    end for
  22.      for each result of $K_i(x)$
  23.        Calculate out of sample error $O_E$
  24.        Adjust the weight $Z = \sum_{i=1}^{m} \mu_t{}^{r} * K_i(x)$ based on the error
  25.     End for
  26.    Find weak learner with minimum error $argmin\, O_E\ (K_i(x))$
  27.    Obtain strong classification results
End
```

Algorithm 1 Stochastic Gaussian Neighbor Embedding based Probit Regressive Reweight Boost Classification

Algorithm 1 illustrates an algorithmic process of the proposed method for examining the software code to improve the software quality analysis through the ensemble classification algorithm. Initially, the software programs are collected from the DS and the codes are used for quality analysis. Then the relevant metrics are selected for analyzing the software codes. The reweight boost ensemble technique uses the set of weak classifiers to analysis the source code with the features (i.e. software metrics). The analysis is done with the help of Pearson correlative probit regression. After analyzing, the files are classified as normal or defected. Through performing the weight updates, the output of weak classifier results is combined to create a strong one. For each weak classifier, weights are assigned subsequently the out of sample error is computed. The proposed ensemble classifier utilizes gradient descent function to lessen the error rate of classification. Based on the out of sample error, then the weight of every weak learner gets updated. At last, the ensemble technique discovers the best classifier with lesser error. Therefore, the ensemble technique enhances the accuracy of classification and lessens the FPR.

IV. EXPERIMENTAL SETUP AND PARAMETER SETTINGS

Experimental analysis of SGNE-PRRBC technique and existing methods Fuzzy-filtered neuro-fuzzy framework [1] and HyGRAR [2] is performed in JAVA programming. The data set using open source projects collected from https://sourceforge.net/. The DS comprises the number of open source projects. For the experimental consideration, SchoolMate project files are taken for testing the quality of softwares. The SchoolMate project comprises the 66 program files. Each and every source code of the program is monitored and finds the normal or defected files to enhance the software quality. Experimental analysis is performed with various metrics namely

- Accuracy
- False-positive rate
- Computation time

V. RESULTS AND DISCUSSION

The performance results of SGNE-PRRBC technique and existing methods [1] and [2] are discussed in this section with various metrics. With the assist of table and graphical representation, the performance is evaluated according to the following metrics.

1.3 Impact of Accuracy

Accuracy is measured as number of program files are correctly classified as normal or defected from the DS. The accuracy is mathematically calculated using the given formula,

$$A = \left(\frac{Number\ of\ \ p_f\ are\ correcly\ classified}{n}\right) * 100 \quad (9)$$

Where A represents the accuracy, n denotes a number of files. Accuracy is calculated in percentage (%).

The comparative result analysis of accuracy is presented in below Table 1.

Table 1 Accuracy versus the number of program files

Number of program files	Accuracy (%)		
	SGNE-PRRBC	Fuzzy-filtered neuro-fuzzy framework	HyGRAR
5	80	60	40
10	90	80	60
15	87	73	67
20	90	85	80
25	92	88	84
30	93	87	83
35	94	89	86
40	90	85	83
45	93	89	87
50	92	88	84

Table 1 describes the experimental result of accuracy for three methods with number of program files in the range from 5 to 50. These programs are taken from the school mate project files. The results show that the accuracy of SGNE-PRRBC technique is minimized as compared to the existing Fuzzy-filtered neuro-fuzzy framework [1] and HyGRAR[2] respectively.

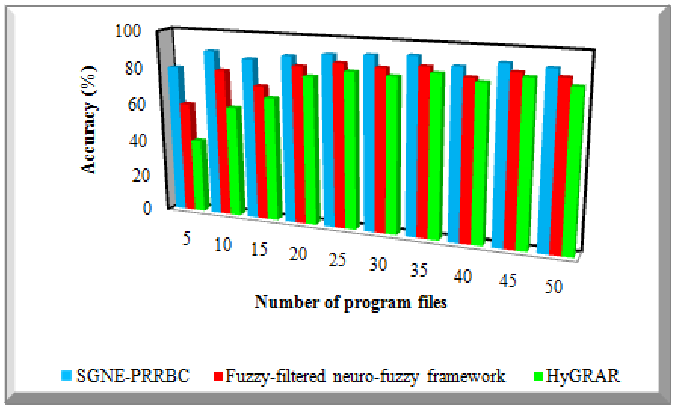

Figure 4 comparative analysis of accuracy

Figure 4 given above illustrates the accuracy of file classification. In the above graph, x axis represents the number of program files taken from the schoolmate projects and y axis denotes the accuracy of file quality prediction through the classification. As inferred in the figure, the accuracy is found to be improved using the proposed SGNE-PRRBC technique than the other two existing methods.

This is due to the SGNE-PRRBC technique uses the ensemble classification algorithm. Initially, the Pearson correlative probit regression function is used to analyze the source codes of the program files with the selected software metrics. Then the regression function classifies the input file as normal or defected based on the probabilistic results. These weak learners' results are summed and obtain the strong classification results as an output. From the experimental, it is identified that for '5' program files are taken in the first iteration, the proposed SGNE-PRRBC technique correctly classified 4 program files and the accuracy is 80% whereas other two existing methods Fuzzy-filtered neuro-fuzzy framework [1] and HyGRAR[2] correctly classified 3 and 2 files and their accuracy are 60% and 40% respectively. The accuracy of SGNE-PRRBC technique is increased by 10% and 25% as compared to existing [1] and [2].

1.4 Impact of False positive rate

FPR is measured as a number of program files which are incorrectly classified as normal or defected from the software programs. FPR is formalized as below,

$$FPR = \left(\frac{\text{Number of } p_f \text{ are incorrecly classified}}{n}\right) * 100 \qquad (10)$$

From (10), FPR denotes the false positive rate, n indicates a number of program files. FPR is measured in percentage (%).

Table 2 False positive rate versus the number of program files

Number of program files	False-positive rate (%)		
	SGNE-PRRBC	Fuzzy-filtered neuro-fuzzy framework	HyGRAR
5	20	40	60
10	10	20	40
15	13	27	33
20	10	15	20
25	8	12	16
30	7	13	17
35	6	11	14
40	10	15	18
45	7	11	13
50	8	12	16

Table 2 describes the experimental results of FPR for three methods with number of program files is varied from 5 to 50. The graphical analysis of false-positive rates of three methods is shown in figure 5.

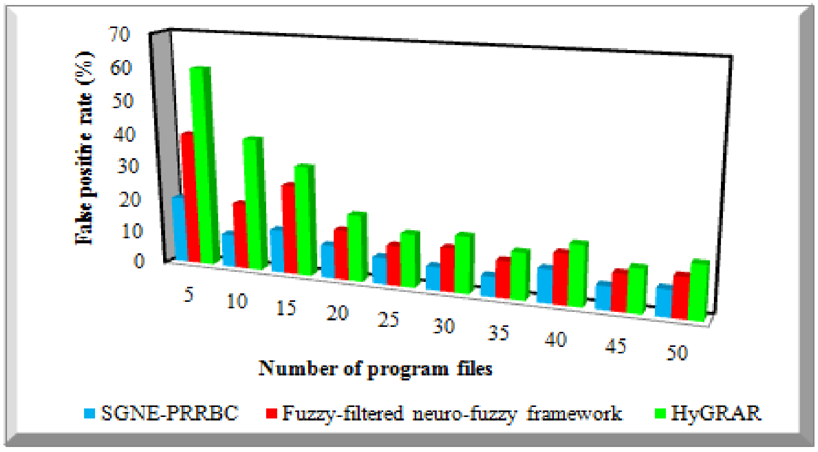

Figure 5 comparative analysis of false-positive rate

Figure 5 given above shows the performance graphical representation of FPR. The numbers of program files are given to the x axis and y axis denotes the FPR. From figure 5, it is evident that, the SGNE-PRRBC technique provides minimal FPR than the other two methods. This is because of the fact that the SGNE-PRRBC technique uses the ensemble classification minimizes the out of sample error using gradient descent function. The initial weight is updated depending on the error value. This helps to minimize the incorrect program files classification. The FPR of SGNE-PRRBC technique is reduced by 41% and 56% as compared to existing [1] and [2].

1.5 Impact of CT

CT is measured as the amount of time consumed for classifying the given program files as normal or defected. CT is formalized as below,

$$CT = n_{p_f} * t\ (classifying\ one\ p_f) \quad (11)$$

Where CT represents the computation time, n_{p_f} represents the number of program files, t indicates a time for classifying one program files. CT is measured in milliseconds (ms).

Table 3 Computation time versus the number of program files

Number of program files	Computation time (ms)		
	SGNE-PRRBC	Fuzzy-filtered neuro-fuzzy framework	HyGRAR
5	12	13	14
10	14	16	17
15	18	20	21
20	20	22	24
25	23	25	28
30	24	27	30
35	25	28	32
40	26	29	33
45	27	30	34
50	30	32	35

Table 3 describes the experimental results of CT for three methods with number of program files. From table 3, the result of CT evident that, the SGNE-PRRBC technique is provides the minimal CT than the conventional methods.

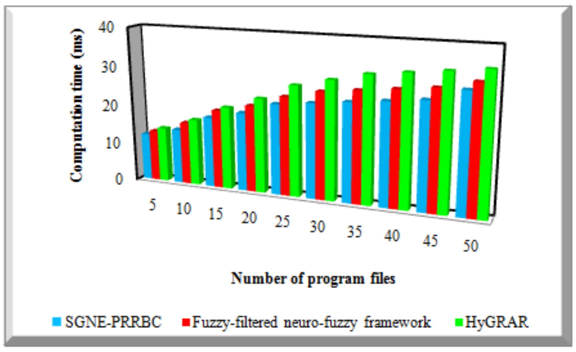

Figure 6 comparative analysis of computation time

As shown in figure 6, the comparative analysis of CT with three different methods of SGNE-PRRBC technique and existing Fuzzy-filtered neuro-fuzzy framework [1] and HyGRAR [2]. In the above graphical result, it is inferred that the CT using SGNE-PRRBC technique is lesser than the conventional methods. This is due to the utilization of Gaussian distributive stochastic neighbor embedding technique. The designed technique performs feature selection for software quality analysis. There are different metrics are selected for analyzing the source code lines of programs for identifying the defect or normal. The stochastic neighbor embedding technique discovers the metrics that is closer to software quality prediction are chosen based on the Gaussian's distributive function. Finally, the relevant metrics are selected for classification. This results in improves the accuracy and lessen the CT of quality prediction. The result of SGNE-PRRBC technique reduces the CT by 10% and 18% as compared to existing [1] and [2].

The above-discussed result evident that SGNE-PRRBC technique is enhances the code examination proficient system in software engineering with greater accuracy and lesser time.

VI. CONCLUSION

In this paper, the SGNE-PRRBC technique is developed by combining feature selection and classification. The SGNE-PRRBC technique provides the contributions of improving the software code quality analysis using an ensemble classifier based code review expert system. First, a Gaussian distributive stochastic neighbor embedding technique is applied to find the software metrics in the feature selection. With the selected features, the classification is performed by applying a Pearson correlative probit reweight boosting technique. The code examination is carried out through the regression analysis. At last, the ensemble classifier model is designed to enhance the accuracy of quality prediction with lesser time. Experimental analysis is performed with number of program files. The evaluation results demonstrated that SGNE-PRRBC technique is feasible for software development and provides better performance in terms of accuracy and CT as well as FPR than the conventional methods.

REFERENCES

[1] Kapil Juneja, "A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation", Applied Soft Computing Journal, Elsevier, Volume 77, 2019, Pages 696–713

[2] Diana-Lucia Miholca, Gabriela Czibula, Istvan Gergely Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks", Information Sciences, Elsevier, Volume 441, May 2018, Pages 152-170

[3] Arvinder Kaur and Inderpreet Kaur, "An empirical evaluation of classification algorithms for fault prediction in open source projects", Journal of King Saud University – Computer and Information Sciences, Elsevier, Volume 30, 2018, Pages 2–17

[4] K. Nitalaksheswara Rao and Ch. Satyananda Reddy, "An Efficient Software Defect Analysis Using Correlation-Based Oversampling", Arabian Journal for Science and Engineering, Springer, Volume 43, Issue 8, 2018, Pages 4391–4411

[5] Guisheng Fan, Xuyang Diao, Huiqun Yu, Kang Yang, and Liqiong Chen, "Software Defect Prediction via Attention-Based Recurrent Neural Network", Scientific Programming, Hindawi, Volume 2019, April 2019, Pages 1-14

[6] Yuancheng Li, Longqiang Ma, Liang Shen, Junfeng Lv, Pan Zhang, "Open source software security vulnerability detection based on dynamic behavior features", PLoS ONE, Volume 14, Issue 8, Pages 1-14

[7] Haonan Tong, Bin Liu, Shihai Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning", Information and Software Technology, Elsevier, Volume 96, 2018, Pages 94-111

[8] Ivan Janicijevic, Maja Krsmanovic, Nedeljko Zivkovic, Sasa Lazarevic, "Software quality improvement: a model based on managing factors impacting software quality", Software Quality Journal, Springer, Volume 24, Issue 2, 2016, Pages 247–270

[9] Hussam Ghunaim and Julius Dichter, "Applying the FAHP to Improve the Performance Evaluation Reliability of Software Defect Classifiers", IEEE Access, Volume 7, 2019, Pages 62794 – 62804

[10] Yu Qiu, Yun Liu, Ao Liu, Jingwen Zhu, Jing Xu, "Automatic Feature Exploration and an Application in Defect Prediction", IEEE Access, Volume 7, 2019, Pages 112097 – 112112

[11] ZhouXu, ShuaiLi, JunXu, JinLiu, XiapuLuo, YifengZhang, TaoZhang, JackyKeung, YutianTang, "LDFR: Learning deep feature representation for software defect prediction", Journal of Systems and Software, Elsevier, Volume 158, 2019, Pages 1-20

[12] Yuanxun Shao, Bin Liu, Shihai Wang, Guoqi Lia, "A novel software defect prediction based on atomic class-association rule mining", Expert Systems with Applications, Elsevier, Volume 114, 2018, Pages 237-254

[13] Hamza Turabieh, Majdi Mafarja, Xiaodong Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction", Expert Systems With Applications, Elsevier, Volume 122, 2019, Pages 27–42

[14] Amjad A. Hudaib, Hussam N. Fakhouri, "An Automated Approach for Software Fault Detection and Recovery", Communications and Network, Volume 8, Pages 158-169, 2016

[15] Lov Kumar, Anand Tirkey, Santanu-Ku. Rath, "An effective fault prediction model developed using an extreme learning machine with various kernel methods", Frontiers of Information Technology & Electronic Engineering, Springer, Volume 19, Issue 7, 2018, Pages 864-888

[16] Wasiur Rhmann, Babita Pandey, Gufran Ansari, D.K.Pandey, "Software fault prediction based on change metrics using hybrid algorithms: An empirical study", Journal of King Saud University - Computer and Information Sciences, Elsevier, 2019, Pages 1-16

[17] Jakob Axelsson, Mats Skoglund, "Quality assurance in software ecosystems: A systematic literature mapping and research agenda", The Journal of Systems & Software, Elsevier, Volume 114, 2016, Pages 69-81

[18] Fei Wu, Xiao-Yuan Jing, Ying Sun, Jing Sun, Lin Huang, Fangyi Cui, Yanfei Sun, "Cross-Project and Within-Project Semisupervised Software Defect Prediction: A Unified Approach", IEEE Transactions on Reliability, Volume 67, Issue 2, 2018, Pages 581 – 597

[19] Yuwei Zhang, Ying Xing, Yunzhan Gong, Dahai Jin, Honghui Li, Feng Liu, "A variable-level automated defect identification model based on machine learning", Soft Computing, Springer, 2019, Pages 1–17

[20] Divya Tomar and Sonali Agarwal, "Prediction of Defective Software Modules UsingClass Imbalance Learning", Applied Computational Intelligence and Soft Computing, Hindawi Publishing Corporation, Volume 2016, January 2016, Pages 1-12