IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-4, April 2019 E-ISSN: 2347-2693

Improved Hybrid Genetic Based Rule Mining Algorithm for Software Defect Prediction

S. Maheswari[1*] , R. Ganesan [2] , K. Chitra [3]

[1]Dept. of Computer Science, Bharathiar ,University, Coimbatore, TN, India
[2]Dept. of Computer Science, Govt Arts College, Melur, Madurai Dt,TN, India
[3]Dept. of Mathematics, Govt Arts College, Melur, Madurai Dt,TN, India

[*]*Corresponding Author: rkmahes04@gmail.com Tel: 9965153218*

DOI: https://doi.org/10.26438/ijcse/v7i4.11881195 | Available online at: www.ijcseonline.org

Accepted: 20/Apr/2019, Published: 30/Apr/2019

Abstract - Predicting software defects is an important issue in the software development and maintenance process, which is related to the overall success of the software. This is because predicting software failures in the previous phase can improve software quality, reliability and efficiency, and reduce software costs. However, developing robust defect prediction models is a challenging task and many techniques have been proposed in the literature. This paper proposes a software defect prediction model based on the new improved hybrid genetic rule mining algorithm (IHGBR). The supervised IHGBR algorithm has been used to predict future software failures based on historical data. The evaluation process shows that the IHGBR algorithm can be used effectively with high accuracy. The collected results show that the IHGBR method has better performance.

Keywords: Rule mining, Defect, Genetic, software metrics, Prediction.

I. INTRODUCTION

Software testing can be defined as "the process Software testing can be defined as "the process of analyzing software elements to detect differences between existing and required conditions and evaluating the characteristics of software elements" [1]. The purpose of this test is to "provide information about the quality of test elements in non-functional and functional requirements" [2]. Software quality, on the other hand, can be defined as "the extent to which the software has the required combination of attributes." Human error leads to product defects, which may be involuntary behavior or produce unexpected or incorrect results [3]. The basic principle of testing is to provide information about software quality, failure of use, and discovery of defects before completion.

These tests also help to better understand the system, especially complex systems, making it an integral part of software engineering [4].

This method is used to predict faults in a program module. Association mining is a data mining method used to identify frequently occurring sets of data elements. This is a way to find the correlation between elements in a dataset [5].

The testing has long emphasized the flaws or deficiencies of the system under different conditions. The main problem is that the evidence manager delays the development process and leads to limited final testing before the software is completed. Another problem is the lack of evidence that the test environment and manual testing or testing tools are heavily dependent. Test environments typically do not rely on precise configuration during software development. There is no such problem in the test is the software test team, around the system function, rather than looking for an attitude, what limits the software defects they found [6].

1. OBJECTIVE

The objectives of this research are detailed below:

- To create novel dataset based on metrics extraction from the source code
- To efficiently remove the noise in the novel dataset using latest filtering mechanism.
- To create rule for predicting optimal fault detection.
- To create novel algorithm to predict software defects.
- Using efficient classification algorithm for better prediction of software defects.
- Using efficient metrics and methods to evaluate the result.
- To suggest low-cost software development processes.
- To reduce time for tracking faultiness and effort.

2. SCOPE

- Finding defects to help improve the level of quality.
- Reducing the risk of failures occurring during operation and gain confidence about the level of quality.

- Improve management decisions by providing information for decision making.
- Prevent defects by identifying the processes in the organization that need improvements by gaining insight into the system behavior.
- Implementing proposed techniques in software systems for automatic classification and detection of software defects

In Introduction of this research work highlights the aims, scope and contributions. Software defects and their measurements have also been introduced. Literature Review is an extensive on pre-processing, feature extraction and classifications in the field of software defect prediction. Data mining techniques used in predicting software defects are also highlighted in this section. In this section we discusses in detail the creation of a novel dataset for prediction of software defects and discusses in a novel feature extraction technique that can be applied to dimensionality reduction and Filtering unwanted attributes for enhanced prediction of software defects. Finally we discusses on a new data mining technique using two existing data mining techniques for accurate software defect predictions. The results of the proposed models and techniques have also been discussed in relevance to their respective areas of application in this paper.

II. LITERATURE REVIEW

There are many studies on using machine learning techniques to predict software errors. For example, the study in [2] proposed a linear Auto-Regression (AR) method to predict defective modules. The study predicts future software failures based on historical data on accumulated software failures. The study also evaluated and compared the AR model and the Known power model (POWM) using Root Mean Square Error (RMSE) measurements. In addition, the study used three sets of data for evaluation and the results were promising and studied the applicability of several ML methods in predicting faults.

Sharma and Chandra [3] added the most important prior research on each LD technique to their research and the current trends in using machine learning to predict software errors. This study can be used as a basis or step in preparing for future work to predict software errors.

R. Malhotra in [5] used Machine Learning (ML) to perform a good systematic evaluation of software error prediction techniques. This document includes a review of all studies from 1991 to 2013, analyzes the ML techniques of software error prediction models and evaluates their performance. Different ML techniques summarize the advantages and disadvantages of ML techniques compared to statistical and ML techniques.

In [6], this document provides a reference point to allow for a common and useful comparison between different error prediction methods. The study proposes a complete comparison between known error prediction methods, and introduces a new method that evaluates its performance by using a good comparison with other methods.

III. METHODOLOGY

The proposed method predicts software defects using a predetermined pattern and analyzes by establishing a novel database for software metrics and ends with a software defect. This chapter details the methods, the data sets used, and the techniques used to identify software defects.

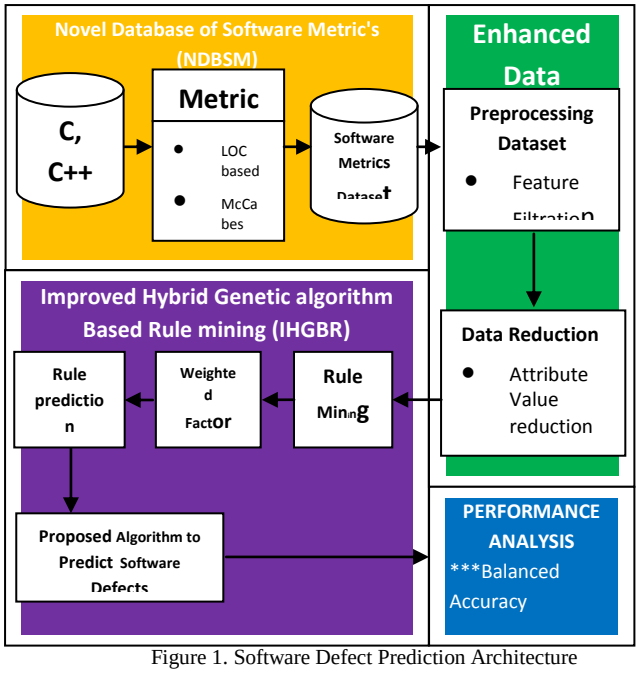

Figure 1. Software Defect Prediction Architecture

Table 1. Software metrics and its definition

Attribute name	Description
Loc	McCabe's line count of code
v(g)	McCabe "cyclomatic complexity"
ev(g)	McCabe "essential complexity"
iv(g)	McCabe "design complexity"
n	Halstead total operators + operands
v	Halstead "volume"
l	Halstead "program length"
D	Halstead "difficulty
i	Halstead "intelligence"
e	Halstead "effort"
b	Halstead
T	Halstead's time estimator
lOCode	Halstead's line count

lOComment	Halstead's count of lines of comments
lOBlank	Halstead's count of blank lines
lOCodeAnd Comment	Numeric
uniq_Op	unique operators
uniq_Opnd	unique operands
total_Op	total operators
total_Opnd	total operands
Branch_count	Total flow graphs

1. Noise Reduction
The first step followed in preprocessing is preliminary filtration. This step removes a portion of the existing noise in the iteration to reduce its impact on subsequent steps. More specifically, an example of noise identified with high confidence is eliminated in this step. This filtering is followed by noiseless filtering. The new filter consists of partial clean data from the previous step and is applied to the training example to produce a clean and noisy set. The final step is to eliminate the noise of the noise score [11].

2. Feature Reduction
An attribute with a constant/fixed value in all instances is easily identifiable because it changes to zero. These attributes do not have any information to distinguish between modules, and in the best case, they are a waste of classification resources. This work reduces redundant attributes in the dataset/metric database. Some attributes are repeated and reduced again. Both attributes have the same value for each instance, resulting in an over-representation of a single attribute [12].

3. Missing Value Reduction
The increase in the amount of data and the emergence of data, the problem of missing data is still ubiquitous in statistical problems, and specific methods are needed. In view of our approach to reducing this large amount of data, this paper proposes the application of the Random Forest algorithm [13], which is an interpolation algorithm for the missing data of mixed data sets. The purpose of the algorithm is to accurately predict individual loss values rather than randomly extracting distributions so that the estimates can result in bias parameters estimated in the statistical model.

4. Redundant Reduction
In this method, the numbers of studies that establish the use of feature sub-selection techniques and which feature sub-selection techniques are widely used are determined. It is important that sub-selection of features is made in the input data before the input data is provided to the learning algorithm, as the data may contain redundant and unrelated features. Of the 22 features selected for this system mapping, 16 studied using the feature sub-selection method, ie exactly 50% of the studies used the feature sub-selection method [14].

5. Rule Mining
Rule mining is a classification method designed for accurate defect measurement and prediction. Before creating a failure prediction model, determine the learning scenario to build the model. The data set is divided into two parts, and the identifier learns in 60% of the data in the data set. Knowledge is implicit in a set of rules. Rule mining consists of two nested loops. The outer loop selects the value of the class, while the inner loop creates rules that apply to the class and returns the best combination for the class [15]. Define simple rules for each metric based on the recommended time interval. These rules are activated if the module metrics are not within the specified time interval (which means that the module was manually verified). It shows 12 basic rules and corresponding indicators, as well as 2 derived rules. The first derivative rule, rule 13, defines the separation of the 12 basic rules. If you shoot some basic rules, it is the trigger for rule 13.

6. Clustering Techniques
The clustering technique divides the training data into groups so that the similarity within the group is greater than the similarity in all groups. The clustering technique uses distance and similarity measures to find similarities between two objects to group them. In this work, he studied K-means technology and fuzzy c means clustering. K-means divides the data into k clusters and iteratively randomly selects the centroid. The value of k affects the performance of the technology [16]. We tried four different k values (ie 2, 3, 4 and 5) and found that k = 2 tends to perform better. We also studied the technique of fuzzy C-means [17] (FCM), which automatically divides the data set into (approximate) optimal number of groups [18].

IV. RESULTS AND DISCUSSION

1. Novel Database of Software Metrics (NDBSM)
The proposed software indicator NDBSM data set considers several software indicators in real time in its collection. The collected metrics are then subjected to a series of steps in which the LOC, McCabes, and Halstead techniques are applied to the creation of the database. The metrics considered are based on completed software projects to support the benchmarking business.

NASA IV&V Metrics Data Program - The Software Data Set provided by the Metric Data Repository (MDP) is being used for most experiments in software engineering and related fields. The data repository contains software metrics as attributes in the dataset and also indicates whether a particular dataset is defective or defect free. All data contained in the repository is collected and verified by the

metrics data program. All software indicators are listed in the table 1 above.

NDBSM extracted a total of 22 attributes because it contains 5 different lines of code, 3 metrics for McCabe, 4 basic Halstead metrics, 8 derived Halstead metrics, 1 branch count and 1 output field.

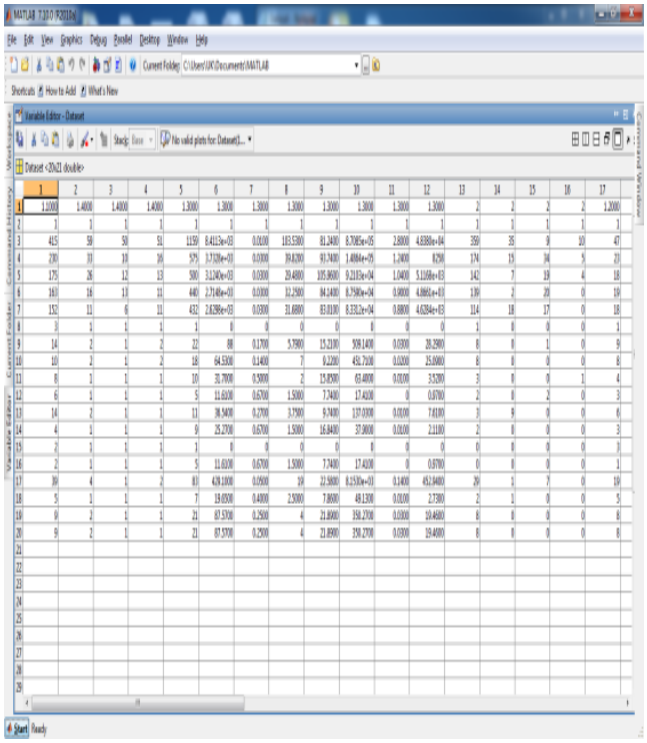

Figure 2. Dataset metric extraction

2. Enhanced Data Preprocessing Technique (EDPT)

The NDBSM database is used as input to the EDPT. EDP T deletes all files that are not included in the metric extraction, namely the readme file, test scripts, and help files. In addition, 0.2% of the "confirmation ID - filename" record associated with the source code file (9 of the 4623 unique tuples) is also eliminated. These records are outliers and, in extreme cases, the source files are moved or deleted. More specifically, the version control system identifies directory change/refactoring as a complete deletion of the file by default. An unusual number of rows are added or deleted each time a file moves one or more levels up or down in the directory structure. In some cases, including large files, more than 10,000 rows have been added or removed in one promise. The above cleaning leads to more accurate model creation. Figure 4 shows the reduced EDPT record for the data set.

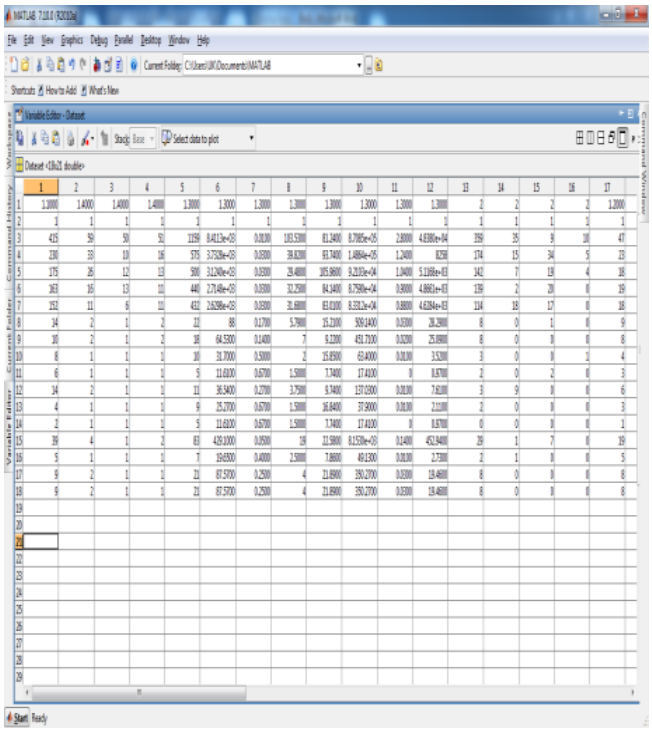

Figure 3. Record Reduction

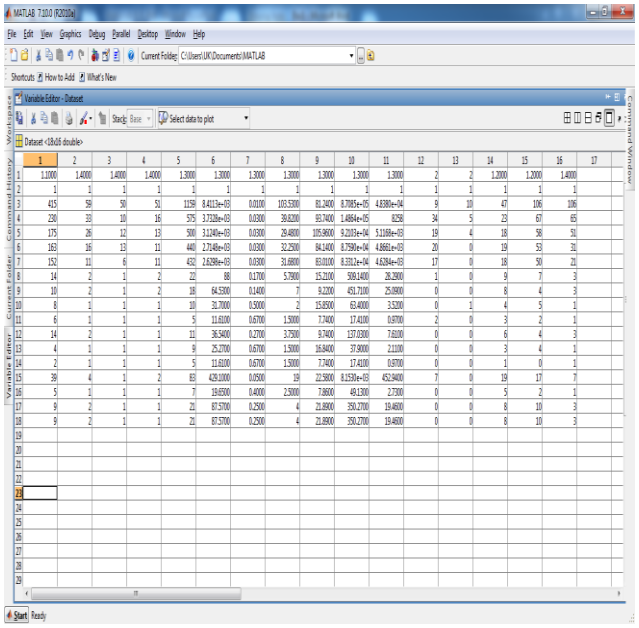

Figure 4. Attribute Reduction

3. Improved Hybrid Genetic Based Rule mining (IHGBR)

In the current work, software metric 21 is a metric for McCabe and Halstead and is measured using a target metric. Using the Matlab tool, the data set is applied to the naive Bayes classifier and the proposed algorithm. This data set is based on a combination of structure and object orientation. Most of the source code is written in C and C++. The study

compared the mean accuracy (values from 0 to 1 in the table), true positive rate, false positive rate, sensitivity and specificity. Accuracy is calculated based on the number of instances correctly classified. Based on the results of these analyses, the method is applicable to size data sets. The table below uses different classifiers to accurately classify and classify instances using the total number of instances in the dataset. It is also highlighted based on sensitivity and specificity values to provide the best classifier. Table 2 lists the weight analysis of the IHGBR in the extracted features, while Table 3 lists the weighting factors table for the IHGBR.

Table 2. Weighted factor prediction

Rule ID	Rule	Weight
1	If LOC>150	4
	Else if LOC >101 && LOC <=150	3
	Else if LOC >51 && LOC <=100	2
	Else if LOC >25 && LOC <=50	1
	Else if LOC <=25	0
2	If V(G) >10	4
	Else if V(G) >7 && V(G) <=10	3
	Else if V(G) >5 && V(G) <=7	2
	Else if V(G) >2 && V(G) <=5	1
	Else if V(G) <=2	0
3	If ev(G) >5	2
	Else if ev(G) >2 && ev(G) <=5	1
	Else if ev(G) <=2	0
4	If iv(G) >10	4
	Else if iv(G) >7 && iv(G) <=10	3
	Else if iv(G) >5 && iv(G) <=7	2
	Else if iv(G) >2 && iv(G) <=5	1
	Else if iv(G) <=2	0
5	If V >350	2
	Else if V >100 && V <=350	1
	Else if V <=100	0
6	If l > 0.1	1
	Else	0
7	If d >10	2
	Else if d >5 && d <=10	1
	Else if d <=5	0
8	If i >50	2
	Else if i >20 && i <=50	1
	Else if i <=20	0
9	If e >5000	4
	Else if e >3000 && e <=5000	3
	Else if e >1500 && e <=3000	2
	Else if e >500 && e <=1500	1
	Else if e <=500	0
10	If t >500	4
	Else if t >300 && t <=500	3
	Else if t >150 && t <=300	2
	Else if t >50 && t <=150	1
	Else if t <=50	0
11	If IOBlank >50	1
	Else	0
12	If Uniq_Opr > 15	1
	Else	0
13	If Uniq_Oprnd > 35	1
	Else	0
14	If Branch_Count > 35	4
	Else if Branch_Count >25 && Branch_Count <=35	3
	Else if Branch_Count >15 && Branch_Count <=25	2
	Else if Branch_Count > 8 && Branch_Count <=15	1
	Else if Branch_Count <= 8	0

Table 3. Weighted Factor Table

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
R1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R3	4	4	2	4	4	0	2	2	4	4	0	1	1	1	4
R4	4	4	2	4	4	0	2	2	4	4	1	0	1	1	4
R5	4	4	2	4	4	0	2	2	4	4	1	0	1	1	4
R6	4	4	2	4	4	0	2	2	4	4	1	0	1	1	4
R7	4	4	2	4	4	0	2	2	4	4	1	0	1	1	3
R8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R9	0	1	0	1	1	1	1	0	1	2	0	0	0	0	0
R10	0	1	0	1	1	1	1	0	1	2	0	0	0	0	0
R11	0	0	0	0	0	1	2	0	1	1	0	0	0	0	0
R12	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
R13	0	1	0	0	1	1	1	0	1	1	0	0	0	0	0
R14	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
R15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R16	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R17	1	2	0	1	4	0	2	1	0	3	0	0	1	1	1
R18	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

4. RULE PREDICTION

Any attribute with a weight > 2.5 receives a prediction factor of 1, otherwise IHGBR is zero. Table 4 lists the predictions for the IHGBR rules, while Table 5 lists the difference tables.

Table 4. Rule Prediction

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
R1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R3	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R4	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R5	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R6	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R7	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R17	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
R18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5. Rule Prediction Differentiation Table

	Mean Difference (Sum(Attr_PrWeight)/15)
R1	0
R2	0
R3	0.466667
R4	0.466667
R5	0.466667
R6	0.466667
R7	0.466667
R8	0
R9	0
R10	0
R11	0
R12	0
R13	0
R14	0
R15	0
R16	0
R17	0.133333
R18	0

According to the established survey and analysis of NASA's MDP data, the difference in mean values is > 0.407, which is described as "software defect" in IHGBR. Table 6 lists the prediction table IHGBR.

Table 6. Prediction Table

	Mean Difference (Sum(Attr_PrWeight)/ 15)	Prediction Result	Actual Result
R1	0	No Defect	No Defect
R2	0	No Defect	No Defect
R3	0.466667	Defect	Defect
R4	0.466667	Defect	Defect
R5	0.466667	Defect	Defect
R6	0.466667	Defect	Defect
R7	0.466667	Defect	Defect
R8	0	No Defect	No Defect
R9	0	No Defect	No Defect
R10	0	No Defect	No Defect
R11	0	No Defect	No Defect
R12	0	No Defect	No Defect
R13	0	No Defect	No Defect
R14	0	No Defect	No Defect
R15	0	No Defect	No Defect
R16	0	No Defect	No Defect
R17	0.133333	No Defect	Defect
R18	0	No Defect	No Defect

5. PERFORMANCE MEASURES

Performance measures of IHGBR are detailed below along with IHGBR classification results in Table 7 and Figure 5 and 6.

True Positive = a =4

False Negative = b =0

False Positive = c =13

True Negative = d = 1

Accuracy = acc = (a+d)/(a+b+c+d) = (4+1)/(4+0+13+1) = 5/18 = 99.72

probability of detection = pd = recall = d/(b+d) = 1 / (0+1) = 1

probability of false alarm = pf = c/(a+c) = 13/17 = 0.765

precision = prec = d/(c+d) = 1/14 = 0.0714

effort = amount of code selected by detector
= (c.LOC + d.LOC)/(Total LOC)= 1174 /1262.1 = 0.9302

Table 7. IHGBR Classification Results

Methods	Sensitivity	Specificity	Accuracy
CM1	0.483	0.986	89.13
JM1	0.198	0.956	83.04
KC1	0.450	0.983	87.91
KC3	0.412	0.922	84.8
MC1	0.693	1	99.34
MC2	0.591	1	69.23
MW1	0.429	0.978	89.14
PC1	0.51	0.999	89.62
PC2	0	1	99.37
PC3	0.986	0.966	84.02
PC4	0.577	0.928	92.27
PC5	0.491	0.990	97.28

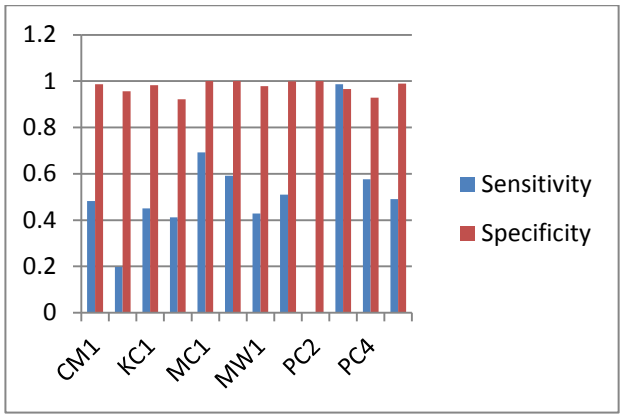

Figure 5. IHGBR Sensitivity and Specificity

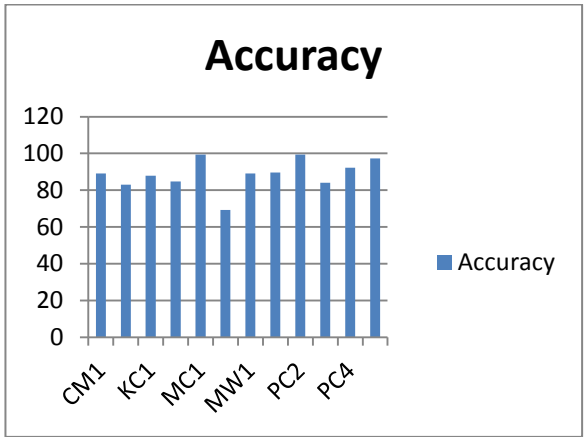

Figure 6. IHGBR Accuracy

V CONCLUSION AND FUTURE WORK

Predicting software defects is a technique in which a predictive model is created to predict future software failures based on historical data. Several methods have been proposed that use different data sets, different software metrics, and different performance metrics. This paper evaluates the use of algorithms proposed in software defect prediction problems. Three machine learning techniques have been used, namely NDBSM, EDPT and IHGBR. The evaluation process is implemented using a real test/debug data set. The experimental results are collected based on accuracy, sensitivity, and specificity. The results show that IHGBR technology is an effective method to predict future software defects. In addition, the experimental results show that using the IHGBR method provides better performance for predictive models than other methods. In future the model can be adapted to Java based open source projects and E-Commerce networking projects using the same metrics.

REFERENCES

[1] Y. Tohman, K. Tokunaga, S. Nagase, and M. Y. "Structural approach to the estimation of the number of residual software faults based on the hyper-geometric districution model," IEEE Trans. on Software Engineering, pp. 345–355, 1989.

[2] A. Sheta and D. Rine, "Modeling Incremental Faults of Software Testing Process Using AR Models ", the Proceeding of 4th International Multi-Conferences on Computer Science and Information Technology (CSIT 2006), Amman, Jordan. Vol. 3. 2006.

[3] D. Sharma and P. Chandra, "Software Fault Prediction Using MachineLearning Techniques," Smart Computing and Informatics. Springer, Singapore, 2018. 541-549.

[4] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," Applied Soft Computing 21, (2014): 286-297

[5] Malhotra, Ruchika. "A systematic review of machine learning techniques for software fault prediction." Applied Soft Computing 27 (2015): 504-518.

[6] D'Ambros, Marco, Michele Lanza, and Romain Robbes. "An extensive comparison of bug prediction approaches." Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 2010.

[7] Gupta, Dharmendra Lal, and Kavita Saxena. "Software bug prediction using object-oriented metrics." Sādhanā (2017): 1-15.

[8] M. M. Rosli, N. H. I. Teo, N. S. M. Yusop and N. S. Moham, "The Design of a Software Fault Prone Application Using Evolutionary Algorithm," IEEE Conference on Open Systems, 2011.

[9] T. Gyimothy, R. Ferenc and I. Siket, "Empirical Validation of ObjectOriented Metrics on Open Source Software for Fault Prediction," IEEE Transactions On Software Engineering, 2005.

[10] Singh, Praman Deep, and Anuradha Chug. "Software defect prediction analysis using machine learning algorithms." 7th International Conference on Cloud Computing, Data Science & EngineeringConfluence, IEEE, 2017.

[11] M. C. Prasad, L. Florence and A. Arya, "A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques," International Journal of Database Theory and Application, pp. 179-190, 2015.

[12] Okutan, Ahmet, and Olcay Taner Yıldız. "Software defect prediction using Bayesian networks." Empirical Software Engineering 19.1 (2014): 154-181.

[13] Bavisi, Shrey, Jash Mehta, and Lynette Lopes. "A Comparative Study of Different Data Mining Algorithms." International Journal of Current Engineering and Technology 4.5 (2014).

[14] Y. Singh, A. Kaur and R. Malhotra, "Empirical validation of objectoriented metrics for predicting fault proneness models," Software Qual J, p. 3–35, 2010.

[15] Malhotra, Ruchika, and Yogesh Singh. "On the applicability of machine learning techniques for object oriented software fault prediction." Software Engineering: An International Journal 1.1 (2011): 24-37.

[16] A. TosunMisirli, A. se Ba¸ S.Bener,"A Mapping Study on Bayesian Networks for Software Quality Prediction", Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, (2014).

[17] T. Angel Thankachan1, K. Raimond2, "A Survey on Classification and Rule Extraction Techniques for Data mining",IOSR Journal of Computer Engineering ,vol. 8, no. 5,(2013), pp. 75-78.

[18] T. Minohara and Y. Tohma, "Parameter estimation of hyper-geometric distribution software reliability growth model by genetic algorithms", in Proceedings of the 6th International Symposium on Software Reliability Engineering, pp. 324–329, 1995.

[19] Olsen, David L. and Delen, " Advanced Data Mining Techniques ", Springer, 1st edition, page 138, ISBN 3-540-76016-1, Feb 2008.

[20] L. H. Crow, "Reliability for complex repairable systems," Reliability and Biometry, SIAM, pp. 379–410, 1974

