I.J. Modern Education and Computer Science, **2018, 4, 51-62**
Published Online April 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2018.04.07

Application of Hybrid Search Based Algorithms for Software Defect Prediction

Wasiur Rhmann
Department of Computer Science and Information Technology, Babasaheb Bhimrao Ambedkar University, (A Central University), Satellite Campus, Amethi, U.P., India
Email:wasiurrhmann786@gmail.com

Received: 20 November 2017; Accepted: 15 January 2018; Published: 08 April 2018

***Abstract*—**In software engineering software defect class prediction can help to take decision for proper allocation of resources in software testing phase. Identification of highly defect prone classes will get more attention from tester as well as security experts. In recent years various artificial techniques are used by researchers in different phases of SDLC. Main objective of the study is to compare the performances of Hybrid Search Based Algorithms in prediction of defect proneness of a class in software. Statistical test are used to compare the performances of developed prediction models, Validation of the models is performed with the different releases of datasets.

***Index Terms*—**Defect, Static metrics, Cyclomatic complexity, Halstead metrics

I. Introduction

With the increase in size and complexity of the software, software engineering community is moving on the use of artificial intelligence techniques in the software development. Software engineers are shifting their intention to early detection of the defects in the software during software development. Earlier identification of defects in the software can help in proper allocation of resources for testing and maintenance. As late detection of defects can cause more rework to take remedial actions. Fixing a bug after it comes in operation is more costly in comparison to fixing it during development [1]. Due to pressure of difficult deadlines to deliver the software, software testing is usually performed in hasty. Detection of higher defect prone classes may prove effective to allocate proper resources. Due to limited manpower and financial constraints there is need of cost effective approaches in detection and repairing of defective software components.

There are different types of defect detection techniques used by researchers namely manual inspection, machine learning [2]. Various machine learning techniques are used for defect prediction namely: classification, artificial neural network, regression tree etc. Although good performances are reported by these techniques but setting of various parameters is essential to obtain better results. Static code measures are used to identify software components that may contain defects. Researches on various software attributes perdition like effort estimation [3], maintainability [4], defect and quality prediction [5] have gained attention of both academia and software industry with the tremendous impact of software applications on human's life. Malhotra [2] presented a framework based on Machine Learning techniques to detect defects in the software. They applied different techniques on the open source software android and compared the performances using statistical techniques, and they find out that support vector machine and voted perceptron method don't have ability to predict defect correctly. Panichella et al. [11] used genetic algorithm for training the prediction model. They trained two models namely: Regression tree, generalized linear model and defects were predicted on the multiple release of six open source projects. Models trained with the genetic algorithm outperformed the model with GA in terms of detection of defect and cost of defect detection. Aljohani and Qureshi[12] proposed a method to detect defect earlier in the development with cheaper method. They used code scanner and code review to detect defects and to improve the quality of software.

Gyimothy et al.[13] used open source software for empirical validation of fault proneness based on the OO metrics. They used linear and logistic regression and machine learning techniques namely: Neural network and decision tree for creation of fault prediction model. NOC was not found efficient for prediction of fault. Olague et al. [14] used open source agile software for validation of OO metrics. Software metrics named as WMC, CBO, RFC and LCOM were found to be effective while NOC metric was insignificant and less significant for two versions. Yuming and Hareton et al. [15] used fault prediction dataset for prediction of faults in two categories: high and low. Pai[16] use Bayesian model to predict the fault proneness models. Different researchers explored the impact of object oriented metrics on the fault prediction using various statistical techniques. In this study we have performed empirical study to assess the performances of various hybrid SBAs. Empirical software engineering deals with the analysis of data obtained from software repositories using statistical and machine learning techniques. In our work [17] we have used UML models to demonstrate the research process in

empirical software engineering. Different software quality attributes may be predicted with the help of software metrics. Maggo and Gupta[18]used machine learning techniques for creation of software reusability prediction model for object oriented software systems. Severity of faults is based on its impact. Defect proneness is dependent variable in our study and it is defined as the probability of occurrence of defect in a class [19].

In this study, following research questions are addressed:

RQ1Are hybrid search based algorithms are feasible to use for defect the prediction of defects in open source software?

RQ3 Which hybrid SBA is best and which is worst in the prediction of defects in the open source software?

RQ4 Are the performances of 10-cross validation are consistent for different data sets?

RQ5 Are the performances of SBAs are statistically different?

Although different techniques are used by researchers to predict defects in software yet up to author's knowledge no work has been reported on the application of hybrid search based algorithms in defect prediction and statistically compare the performance of different hybrid SBAs. The main contributions of our work are as follows:

(1) Comparison of various hybrid SBAs in the defect prediction
(2) Comparison of hybrid SBAs in defect prediction using Statistical techniques
(3) To obtain generalized and unbiased results validation of model is performed
(4) Performance evaluation of the models with the 10 cross validation of the model

In the study we have done the following tasks to complete our study.

- Genetic based feature selection technique is used to select a subset from the set of independent variables
- Application of appropriate data analysis technique
- Selection of suitable performance prediction metric to properly represent the imbalanced data
- Evaluation of true capability of prediction model using efficient validation technique
- Selection of Best hybrid SBA in the prediction of defect using statistical technique

This paper is organized in six sections. Section 2 describes the dataset used in the study and gives the statistical description of the dataset. Section 3 describes the independent and dependent variables used in the study and the software metrics which are used as independent and dependent variables and the correlations among different variables used in the study and different hybrid search based algorithms used in the study. Section 4 describes the performance evaluation metrics used in the defect prediction and Section 5 presents the statistical test to compare the performances of the hybrid SBAs. Section 6 and section 7 describe the applications of presented work and validation of the presented study. Finally, brief conclusion is given in section 8.

II. Description of Datasets Used in the Study

In this study, we have used the dataset obtained from promise data repository [20]. KC1, KC2 and CM1 datasets are used for defect prediction.

Table 1. Description of dataset

Dataset	No. Of instances
KC1	2109
KC2	522
CM1	498

A. Descriptive Statistics about datasets

To better understand the datasets, different data sets are analyzed with various statistical measurements. Various statistical measures are calculated for independent variables of the datasets. Descriptive statistics of datasets are calculated for various metrics of independent variables helps to analyze and understand common features of the different datasets. Different statistical measures namely mean, median, mode, standard deviation, skewness, kurtosis, Minimum, maximum. Table 2 to Table 7 shows the descriptive statistics about datasets kc1, kc2 and cm1.

III. Model DEvelopment and Interpretation

This section describes different software metrics used in the study for the development of software defect prediction models, correlation between different software metrics used in the study, feature selection algorithm which selects subset of the useful metrics for the development of the software defect prediction models and different hybrid techniques, which are used for the development of prediction models are described in this section

A. Variables used in the study

In this study we have used 21 static code metrics as independent variables named as: McCabe's lines of code(loc), McCabe's cyclomatic complexity(v(g)), McCabe's essential complexity(ev(g)), McCabe's design complexity(iv(g)), Halstead's total operators and operands(n), Halstead's volume(v), Halstead's program length(l), Halstead's program's difficulty(d), Halstead's intelligence(i), Halstead's efforts(e), Halstead(b), Halstead' s time estimator(t), Halstead's line count(IOcode), Halstead' s count of comments

lines(IOComment), Halstead' s count of blank(IOBlank), IOCodeandComment, Unique_operaotors(Uniq_Op), Unique_Operand(Uniq_Opnd), total operators(total_Op), total_operand(total_Opnd), Flow of graph(branch count). These metrics and their definition are given in table 8. MacCabe found the relation between complex program paths and fault proneness [21] while Halstead explained that code which is difficult to read is more fault prone [22]. There are four MacCabe's metrics namely: loc, cyclomatic complexity, essential complexity and design complexity.

1. *Loc* It is lines of codes
2. *Cyclomatic complexity*

It measures the numbers of linearly independent paths in the program. Each independent path contains at least one new node or edge which is not included in any other independent path of the program.

3. *Essential complexity*

It is defined as the cyclomatic complexity of reduced flow graph and reduced flow graph of program's control flow graph is design by replacing all control structural of program like (if else while etc) which have single entry and exit point.

4. *Design complexity*

It is defined as cyclomatic complexity of reduced flow graph of a control flow graph where reduced flow graph is obtained by eliminating nodes that does not influence the interrelationship of those design modules.

Halstead measures have three categories namely: base measure, derived measure and line of code

5. *Unique_operators It is defined as number of unique operators in the code.*
6. *Unique_operands It is defined as number of unique operands in the code.*
7. *Length (l) It is sum of total operators and total operands in the program*
8. *Vocabulary It is sum of unique operators and unique operands in the program code*
9. *Volume $V=N*log_2(mu)$ (the number of mental comparisons needed to write a program of length N)*

V*=Volume on minimal implementation=(2+ mu2')*log2(2 + mu2')
where mu=vocabulary, mu2' = the number of arguments to the module

10. *Difficulty D = difficulty = 1/L*
11. *Intelligence I = intelligence = L'*V'*
12. *Efforts E = effort to write program = V/L*
13. *Time to write program T = time to write program = E/18 seconds*

Dependent variable in our study is binary which is defect prone or not.

Table 8. Software metrics with definitions

S.N.	Metrics	Definition
1	loc	McCabe's line count of code
2	v(g)	McCabe "cyclomatic complexity
3	ev(g)	McCabe "essential complexity
4	iv(g)	McCabe "design complexity
5	n	Halstead total operators + operands
6	v	Halstead "volume
7	l	Halstead "program length
8	d	Halstead "difficulty
9	i	Halstead "intelligence
10	e	Halstead "effort
11	b	Halstead
12	t	Halstead's time estimator
13	lOCode	Halstead's line count
14	lOComent	Halstead's count of lines of comments
15	lOBlank	Halstead's count of blank lines
16	lOCodeAndComment	numeric
17	uniq_Op	unique operators
18	uniq_Opnd	unique operands
19	total_Op	total operators
20	total_Opnd	total operands
21	branchCount	% of the flow graph
22	defects	{no, yes}

B. Correlations among metrics

Performance of a classification technique depends on the set of dependent and independent variables. It is good to select independent variables which are highly correlated with dependent variables while correlation between independent variables is not good for a good classification technique. Correlated independent variables represent redundant information. In this section correlation analysis of the independent variables used in the study is done. The value of correlation lies between -1 to 1. Positive value of correlation value implies that if one variable increases then other variable will also be increased. While negative value implies increase in one variable implies decrease in other variable. Spearman's correlation test used for calculation of correlation between variables. Correlation value above 0.8 is larger, and values between 0.8 to 0.5 is moderate while below .5 is considered as low [23-24]. If a variable is correlated with itself then correlation coefficient will be 1. Correlation among independent variables of the dataset kc1, kc2 and cm1 are given in the table 9, 10 and table 11.

C. GGA-FS (Generalized Genetic Algorithm based Features Selection)

A feature selection method selects subset of features which are used as independent variables in the prediction model. It was found [25] that Features selection methods produce the subset of features which can be used for creation of model without affecting the classification quality. GGA-FS is a feature selection algorithm based on Genetic algorithm [26]. Application of genetic algorithm

helps to effectively optimize multi criteria optimization. Every candidate solution is a subset of features and represents the individual of population. If there are total m features then there will be possible subsets of features will be 2^m. A feature subset is represented by binary vector of length m where 1 will represent the selection of that feature and 0 will represent rejection of that feature. Fitness function of genetic algorithm is constructed on the basis of accuracy of classification and cost of classifying. This data pre-processing technique is implemented in open source tool Keel (http://www.keel.es.

Next, we discuss various hybridized SBAs used in this study. In hybridized algorithms, the advantages of SBAs as well as non-SBAs are combined into one algorithm. Following are the hybridized algorithms used in this study:

1. Fuzzy Adaboost (GFS-AB)
2. Fuzzy Logitboost (GFS-LB)
3. Logitboost with Single Winner Inference (GFS-MaxLB)
4. Hierarchical decision rules (HIDER)
5. Particle Swarm Optimization - Linear Discriminant Analysis (PSOLDA)

GFS-AB, GFS-LB, GFS-MaxLB

Boosting algorithms combine multiple weak classifiers to form a strong classifier and give higher performance than individual classifiers. With the addition of new classifier training dataset is reweighted and a voting strength is assigned to the classifier. By combining weak classifiers a compound classifier is generated on the basis of weak classifiers. Both AB and LB come under the category of boosting algorithms, and they learn fuzzy rules where fuzzy rule base are weighted combination of weak classifiers. GFS-AB and GFS-LB build the fuzzy rules base in incremental manner. Boosting fuzzy rules imply fitting a single rule to a set of weighted training example. This process continues as many times as there are rules in the base [27]. Thus, GFS-AB and GFSLB help to build the rule base in an incremental manner by down-weighting the training instances which are correctly classified by a rule instead of removing them which avoids conflicting rules in the rule base [28, 29]. The authors Sanchez and Otero [30] have analyzed the disadvantages of using boosting algorithms to learn fuzzy classifiers. The high interaction between the rules leading to poor quality rule base is the main drawback. The underlying reason of this drawback is the use of 'sum of votes' scheme to generate the output. Thus, the authors propose an interface scheme known as "single winner" which does not combine the rules with the arithmetic sum [30]. Using this interface scheme, the authors showed that the resultant fuzzy rule base is of high quality as well as good quality.

HIDER

It is a supervised learning algorithm. In this algorithm hierarchical set of rules are produced which can be applied for continuous as well as discrete domains [31]. When a training example is classified correctly by a rule then all other previous rules will not match the condition.

PSOLDA

It is used for classification of the data. As Linear Discriminant Analysis (LDA) does not use feature selection technique so its performance degrades [32] so it uses particle swarm optimization (PSO) technique for feature selection technique and improve LDA in terms of the classification accuracy.
In our experiments the parameters set for Genetic algorithm based features selection are given in the table 12.

Table 12. Control parameters setting for Evolutionary feature selection algorithm

S.N.	Technique	Parameters	
1	GGA	Cross Probability	0.7
		Mutation Probability	0.01
		Population Size	50
		Number of Evaluations	500
		Beta Equilibrate Factor	0.99
		Number of Neighbors	1
		Use Elitism	Yes

Table 13 presents the parameters set for hybrid search based algorithms.

Table 13. Control parameters setting for hybrid search based algorithms

S.N.	Technique	Parameters
1	GFS-adaboost-c	Numlabels 3 Numrules 8
2	GFS-logitboost	numlabels 3 numrules 25
3	GFS-MaxlogitBoost	Numlabels 3 Numrules 8
4	HIDER-C	populationSize 100 nGenerations 100 mutationProbability 0.5 crossPercent 80 extremeMutationProbability 0.05 pruneExamplesFactor 0.05 penaltyFactor 1 errorCoeficient 0
5	PSOLDA-C	

Results of 10 cross validation of different models for the dataset kc1, kc2 and cm1 are presented in the table 14, 15 and table 16.

Table 14. Performance of the hybridized SBAs models for given kc1 dataset

S. N.	Technique	Accuracy	g-mean
1	GFS-adaboost-c	0.979	0.893
2	GFS-logitboost	0.949	0.929
3	GFS-MaxlogitBoost	0.922	0.891
4	PSOLDA-C	0.8909	0.791
5	HIDER-C	0.881	0.537

Table 15. Performance of the hybridized SBAs models for given kc2 dataset

S. N.	Technique	Accuracy	g-mean
1	GFS-adaboost-c	0.969	0.9691
2	GFS-logitboost	0.9789	0.98302
3	GFS-MaxlogitBoost	0.9731	0.975
4	PSOLDA-C	0.929	0.8797
5	HIDER-C	0.9693	0.961

Table 16. Performance of the hybridized SBAs models for given cm1 dataset

S. N.	Technique	Accuracy	g-mean
1	GFS-adaboost-c	0.979	0.967
2	GFS-logitboost	0.989	0.9754
3	GFS-MaxlogitBoost	0.9678	0.9132
4	PSOLDA-C	0.9738	0.962
5	HIDER-C	0.9738	0.962

Performances of different techniques are presented graphically in fig. 1 and fig. 2. Fig. 1 compares the performances of different techniques which are used in the study for the creation of the software defect prediction models on the basis of accuracy. Fig. 2 presents the performances of different techniques used for the creation of software defect prediction models on the basis of g-mean of the techniques.

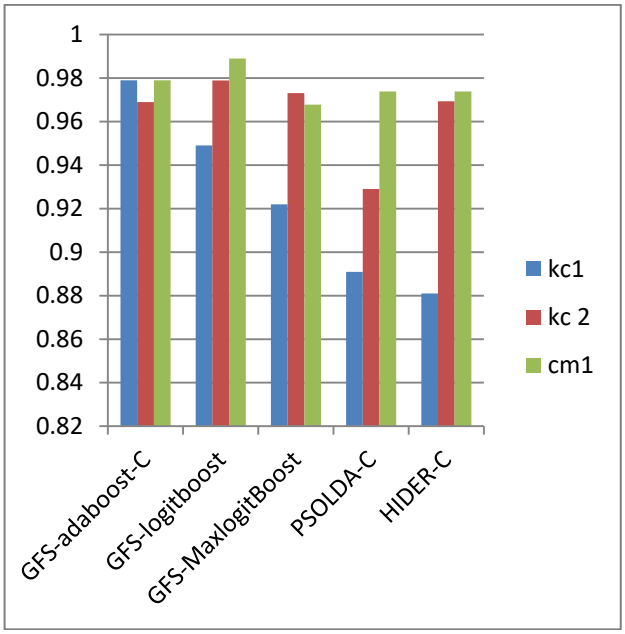

Fig.1. Performance comparison of different techniques with accuracy

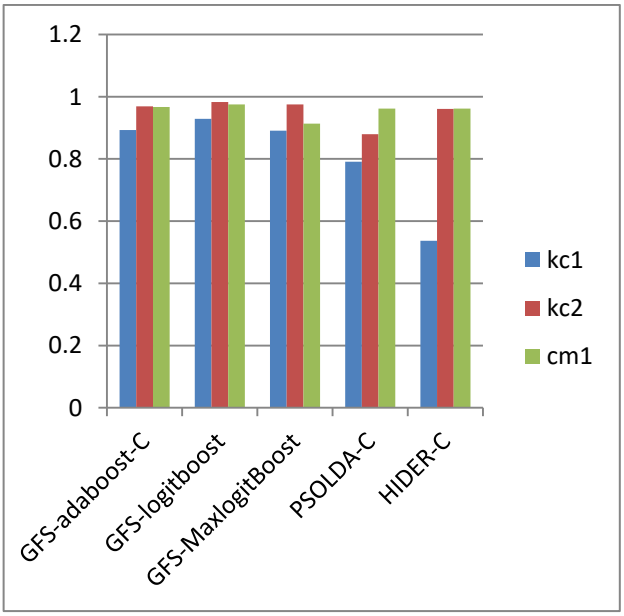

Fig.2. Performance comparison of different techniques with g-mean

IV. Performance Evaluation Metrics

To evaluate the performances of the hybrid search based algorithms two metrics namely g-mean and accuracy are used [33].

A. g-mean

g-mean calculation is based on two accuracies: accuracy of positives (a+) and accuracy of negatives (a_). It is usually used in case of imbalanced data set. The formulae for a+ and a- is given as:

$$a+=\frac{TP}{TP+FP}$$
$$a-=\frac{TN}{TN+FN}$$
$$g=\sqrt{(a+)\times(a-)}$$

B. Accuracy

Accuracy or correctness is defined as the ratio of the number of classes correctly predicted to the total number of classes.

Accuracy is calculated using the formula given below,

$$\text{Accuracy}=\frac{TP+TN}{TP+FP+FN+TN}x100$$

C. Confusion matrix

Confusion matrix is used to predict the performance of the prediction models. In confusion matrix rows refers to actual outputs and column refers to predicted outputs. Confusion matrix used for vulnerability prediction is given in the table 17:

The meaning of different terms used in the confusion matrix is given below:

Table 17. Confusion matrix of two class outcome variables

		Predicted	
		0(Not defective)	1(Defective)
Actual	0(Not defective)	TND	FD
	1(Defective)	FND	TD

TND (True Not Vulnerable): Denotes the number of correctly predicted non defective classes;

False Not Defective (FND): It means the number of incorrectly predicted non-defective classes.

False Defective (FD): It means the number of incorrectly predicted defective classes.

True Defective (TD): It means the number of correctly predicted defective classes.

Table 18. Validation results of different datasets with accuracy measure

Technique	Kc1	Kc2	Cm1
GFS-adaboost-c	0.979	0.969	0.979
GFS-logitboost	0.949	0.9789	0.989
GFS-MaxlogitBoost	0.922	0.9731	0.9678
PSOLDA-C	0.8909	0.929	0.9738
HIDER-C	0.881	0.9693	0.9738

Validation method

In order to evaluate the performance of presented model unseen data is used for testing the model. Testing and training datasets are different as using same dataset for testing and training the model may cause over fitting of the model. In our study we have used 10 cross validation in which dataset is divided in 10 equal parts and one part is used as testing data while other 9 are used as training the models. This process is repeated for 10 times each time with different part as test data from data partitions and average is taken from results of 10 times repetition.

Table 19. Validation results of different datasets with performance measure g-mean

Technique	Kc1	Kc2	Cm1
GFS-adaboost-c	0.893	0.9691	0.967
GFS-logitboost	0.929	0.98302	0.9754
GFS-MaxlogitBoost	0.891	0.975	0.9132
PSOLDA-C	0.791	0.8797	0.962
HIDER-C	0.537	0.961	0.962

V. Statistical Comparison

In this section performances of the different techniques are statistically evaluated and for the evaluation of the performances of different techniques Friedman test is used. Friedman test is a non-parametric test and described below:

A. Friedman test

Friedman test is a non-parametric test that works on multiple dataset. It is based on chi square distribution with N-1 degrees of freedom. Here N is the number of techniques used in the study. This study has 4 degree of freedom as there are 5 techniques used to create prediction models. Hypothesis is checked at $\alpha = 0.05$. Performances of the presented techniques on 3 data sets are evaluated with Friedman test.

Testing hypothesis for accuracy

Null Hypothesis: There is no statistical difference between the performances of various defect prediction techniques

Alternate Hypothesis: There is statistical difference between the performances of prediction techniques

Testing hypothesis for g-mean

Null Hypothesis: There is no statistical difference between the performances of various defect prediction techniques

Alternate Hypothesis: There is statistical difference between the performances of prediction techniques

Table 20. Test Statistics Friedman Test

N	3
Chi-Square	7.379
df	4
Asymp. Sig.	.117
Exact Sig.	.102
Point Probability	.008

Table.21. Mean ranks in Friedman test using accuracy measure

S. N.	Technique	Mean rank
1	GFS-adaboost-c	3.83
2	GFS-logitboost	4.67
3	GFS-MaxlogitBoost	2.67
4	PSOLDA-C	1.83
5	HIDER-C	2.50

Since p-value>0.05, so Null hypothesis is accepted. Hence, there is no statistical difference between the performances of different hybrid SBAs on the basis of accuracy.

Table 22. Test Statistics Friedman Test

N	3
Chi-Square	8.881
df	4
Asymp. Sig.	.064
Exact Sig.	.032
Point Probability	.002

Table.23. Mean ranks in Friedman test using g-mean

S. N.	Technique	Mean rank
1	GFS-adaboost-c	3.67
2	GFS-logitboost	5.00
3	GFS-MaxlogitBoost	2.67
4	PSOLDA-C	1.83
5	HIDER-C	1.83

Since p-value>0.05, so Null hypothesis is accepted. Hence, there is no statistical difference between the performances of different hybrid SBAs on the basis of g-mean for the prediction of vulnerability.

VI. Applications of Work

This section describes the applications of the study. Once the defect prone classes of the software which is being developed are identified, it will help the organization in the following ways:

1. The designers can redesign such classes so that less number of classes suffers from defects during the later phases of software development as well as less number of defects detected during software testing phase. Software tester can effectively use resources to test the software by performing rigorous testing of highly defect prone classes. Tester can assign higher priority to higher defect prone classes. Software developer can focus on critical classes with more attention.

2. Static code metrics which are more related with the defect proneness of class can be used by software practitioners to set a quality benchmark for secure software and this permissible range for secure software should be used across different organizations to develop secure software. Any deviation from secure range should force software organization to take preventive measures.

3. This study explores the possibility of the use of hybrid search based algorithms to predict the defect prone classes and helps the security experts to take decision regarding which technique is best among hybrid search based algorithms to defect prediction.

VII. Threats to the Validity

A. *Internal validity*

Any threat which may be introduced in the research design and can alter the conclusion is called internal threat. Internal threat is possible if besides independent variables there is some other factor which may influence the both independent and dependent variable. In our case there may be size of software which may affect the metrics used in the study and the defect proneness of the software. So this threat of confounding variable lies in our study.

B. *External validity*

External validity of the study deals with whether results of the study can be generalized. In our study we have considered three data sets for experimentation purpose. Data sets of the metrics are calculated on the C, C++ software. So there is a threat of applicability of proposed approach for other language like java etc. However, we have described the setting of experiments the results can be repeated and replicated.

C. *Construct validity*

Construct validity threats refer if the variables used in the study are correctly referred to the same thing as that is supposed to measure by them. Static metrics used in the study are well-established and are effectively represents the concepts they are supposed to refer. Selected metrics accurately and properly refers the concept of static variables thus they effectively reduce the threats of construct validity. Since metrics and faults data are collected from publicly available NASA datasets KC1, KC2 and CM1, we have no information how they are calculated and their accuracy cannot be assured. This is possible threat of construct validity.

VIII. Conclusions

It is common practice in software engineering to predict defective software classes for next release of the software to effectively allocate the software testing resources. Identification of defect prone classes may help to prioritize the software classes for testing. This may lead to saving of resources and cost of the system. In this study we investigated the performances of various hybrid search based algorithms for defective software classes' prediction in the early phases of the software development. We have used various static software metrics for prediction of defective software classes and applied it on different data sets kc1, kc2 and cm1. The performance of difference model is evaluated using g-mean and accuracy. The major findings of the work are: GFS-logitboost hybrid SBA is better in terms of g-mean and accuracy while PSOLDA-C hybrid search algorithm is found to be least effective in terms of g-mean and accuracy. To generalize our results we plan to perform it on various large dataset. There is a need to investigate the confounding effect of relationship between defect proneness and static software metrics.

References

[1] H. Lim, A. L. Goel, *Software Effort Prediction*, Wiley Encyclopedia of Computer Science and Engineering, 2008.

[2] Ruchika Malhotra, "An empirical framework for defect prediction using machine learning techniques with Android software", *Applied Soft Computing*, 2016, pp. 1-17.

[3] M. Jorgensen, "Experience with the accuracy of software maintenance task effort prediction models", *IEEE Trans. Softw. Eng.*, vol. 21, pp. 674-681, 1995.

[4] M. Riaz, E. Mendes, E. Tempero, "A systematic review of software maintainability prediction and metrics", *in: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 367–377.

[5] Y. Zhou, H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines", *J. Syst. Softw*., vol. 80, pp. 1349–1361, 2007.

[6] Y. Ma, G. Luo, X. Zeng, A. Chen, "Transfer learning for cross-company software defect prediction", *Inform. Softw. Technol.*, vol. 54, pp. 248–256, 2012.

[7] A. Tosun, A. Bener, B. Turhan, T. Menzies, "Practical considerations in deploying statistical methods for defect prediction: a case study within the Turkish telecommunications industry", *Inform. Softw. Technol*., vol. 52, pp. 1242-1257, 2010.

[8] X. Yuan, T.M. Khoshgoftaar, E.B. Allen, K. Ganesan, "An application of fuzzy clustering to software quality prediction", *in: Proceedings, 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, 2000, pp. 85–90.

[9] N.E. Fenton, M. Neil, "A critique of software defect prediction models", *IEEE Trans. Softw. Eng*., vol. 25, pp. 675-689, 1999.

[10] Parag C. Pendharkar, "Exhaustive and heuristic search approaches for learning a software defect prediction model", *Engineering Applications of Artificial Intelligence*, vol. 23, pp. 34–40, 2010.

[11] Annibale Panichella, Carol V. Alexandru, Sebastiano Panichella, Alberto Bacchelli, Harald C. Gall, "A Search Based Training Algorithm for Cost-aware Defect Prediction", *ACM*, Colorado, USA, 2016, pp. 1077-1084.

[12] Ohood A. Aljohani, Rizwan J. Qureshi, "Proposal to Decrease Code Defects to Improve Software Quality", *I. J. Information Engineering and Electronic Business*, vol. 5, pp. 44-51, 2016.

[13] Gyimothy T, Ference R, Siket I, "Empirical validation of object-oriented metrics on open source software for fault prediction", IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 897-910, 2005.

[14] Olague H, Etzkorn L, Ghoston S, Quattlebaum S, "Empirical validation of three software metrics suites to predict fault proneness of object-oriented classes developed using highly iterative or agile software development process", *IEEE Trans. Software. Eng.*, vol. 33, no. 8, pp. 402-419, 2007.

[15] Yuming Z, Hareton L, "Empirical analysis of object oriented design metrics for predicting high severity faults", *IEEE Transaction Softw. Engineering*, vol. 32, no. 10, pp. 771-784, 2006.

[16] Pai G., "Empirical analysis of software fault content and fault proneness using Bayesian methods", *IEEE Transaction Software Engineering*, vol. 33, no. 10, pp. 675-686, 2007.

[17] Wasiur Rhmann, "UML Models of Research Process in Empirical Software Engineering", *International Journal of Computer Sciences and Engineering*, vol. 5, no. 10, pp. 176-180, 2017.

[18] Surbhi Maggo, Chetna Gupta, "A Machine Learning based Efficient Software Reusability Prediction Model for Java Based Object Oriented Software", *I. J. Information Technology and Computer Science*, vol. 2, pp. 1-13, 2014.

[19] Ruchika Malhotra, ArviderKaur, |Yogesh Sigh, "Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines", *Int J. Syst. Assur. Eng. Manag*., vol. 1, no. 3, pp. 269-281, 2010.

[20] Sayyad Shirabad, J. and Menzies T. J., The PROMISE Repository of Software Engineering Databases, School of Information Technology and Engineering, University of Ottawa,Canada,http://promise.site.uottoawa.ca/SERepository

[21] T.J. McCabe, "A Complexity Measure", *IEEE Transaction on Software Engineering*, vol. 2, no. 4, pp. 308-320, 1976.

[22] M.H. Halstead, Elements of Software Science, Elsevier, 1977.

[23] Shatnawi R, Li W., "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process", *J. Syst. Softw*., vol. 81, no. 11, pp. 1868–1882, 2008.

[24] Succi G, Pedrycz W, Djokic S, Zuliani P, Russo B. "An empirical exploration of the distributions of the Chidamber and Kemerer object-oriented metrics suite". *Empir. Softw. Eng* ., vol. 10, pp. 81–103, 2005.

[25] Maysam Toghraee, Hamid Parvin and Farhad Rad, "The Impact of Feature Selection on Meta-Heuristic Algorithms to Data Mining Methods", *I. J. Modern Education and Computer Science*, vol. 10, pp. 33-39, 2016.

[26] Jihoon Yang, Vasant Honavar, "Feature Subset Selection Using a Genetic Algorithm", *IEEE Intelligent System*, Vol. , pp. 44-49, 1998.

[27] Freund Y, Schapire RE, Short A. "Introduction to Boosting*". J. Jpn Soc. Artif. Intell*, vol. 14, pp. 771–780, 1999.

[28] Jesus MJ, Hoffmann F, Navascués LJ, Sánchez L. "Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms". *IEEE Trans Fuzzy Syst*., vol. 12, pp. 296–308, 2004.
[29] Otero J, Sanchez L. "Induction of descriptive fuzzy classifiers with the Logitboost algorithm". *Soft. Comput*., vol. 10, pp. 825–835, 2006.
[30] Sanchez L, Otero J. "Boosting fuzzy rules in classification problems under single-winner inference". *Int. J. Intell. Syst.*, vol. 22, pp. 1021–1034, 2007.
[31] Aguilar-Ruiz JS, Riquelme JC, Toro M. "Evolutionary learning of hierarchical decision rules". *IEEE Trans. Syst*., vol. 33, pp. 324–331, 2003.
[32] Lin SW, Chen SC., "PSOLDA: a particle swarm optimization approach for enhancing classification accuracy rate of linear discriminant analysis". *Appl. Soft. Comput*., vol. 9, pp. 1008–1015, 2009.
[33] Ruchika Malhotra, *Empirical Research in Software Engineering*, CRC Press, Taylor and Francis group, 2016.

Authors' Profiles

Wasiur Rhmann received the B.Sc.(Hons.) in Physics and M.C.A. from Aligarh Muslim University Aligarh India in 2010 and 2013 respectively. He has submitted his Ph.D. thesis in Babasaheb Bhimrao Ambedkar University (A Central University) Lucknow, India. His research interest is Software engineering, Software testing and UML modeling. He has published papers in several Internationals and National journals of repute. Presently he is working as Assistant Professor in Babasaheb Bhimrao Ambedkar University (A Central University) Satellite campus Amethi, India.

Table 2. Descriptive statistics about kc1

	Loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b
Mean	20.37226	2.838028	1.674443	2.54642	49.82945	258.6967	0.319583	6.771242	21.24007	5242.386	0.086738
Median	9	1	1	1	16	57.06	0.2	3.5	14.4	213.97	0.02
Mode	2	1	1	1	4	8	0.67	1.5	5.33	12	0
Standard Deviation	29.75444	3.900763	2.200659	3.375859	83.59987	516.3176	0.317029	7.863646	21.50037	17444.98	0.175507
Sample Variance	885.3268	15.21595	4.8429	11.39643	6988.939	266583.9	0.100507	61.83692	462.2658	3.04E+08	0.030803
Kurtosis	16.38214	19.05927	27.35304	24.35597	22.40961	36.03354	5.613137	5.597749	7.322908	89.19928	34.44
Skewness	3.352322	3.737101	4.632832	4.018701	3.697793	4.581499	1.793769	2.139364	2.131496	7.699113	4.529341
Range	287	44	25	44	1106	7918.82	2	53.75	193.06	324803.5	2.64
Minimum	1	1	1	1	0	0	0	0	0	0	0
Maximum	288	45	26	45	1106	7918.82	2	53.75	193.06	324803.5	2.64

Table 3. Descriptive statistics about kc1

	lOCode	lOComment	lOBlank	lOCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount
Mean	291.245	14.52537	0.945946	1.759602	0.132764	7.631674	9.537316	31.04372	4.665908
Median	11.89	5	0	0	0	6	5	10	1
Mode	0.67	0	0	0	0	3	1	3	1
Standard Deviation	969.1652	24.1883	3.085271	3.85685	0.704023	5.730347	12.19573	51.77606	7.792206
Sample Variance	939281.1	585.074	9.518898	14.87529	0.495648	32.83687	148.7357	2680.76	60.71848
Kurtosis	89.19942	17.84364	60.51908	37.02712	103.079	1.241395	8.187673	22.42278	19.17806
Skewness	7.69912	3.416811	6.56998	4.780011	8.789034	1.140766	2.387301	3.698032	3.755887
Range	18044.64	262	44	58	12	37	120	678	88
Minimum	0	0	0	0	0	0	0	0	1
Maximum	18044.64	262	44	58	12	37	120	678	89

Table 4. Descriptive statistics about kc2

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t
Mean	36.89	4.89	2.44	3.65	94.63	555.47	0.27	9.73	28.32	18542.99	0.188	1094.81
Median	13	2	1	2	27	109.20	0.14	6	20.56	613.59	0.04	34.09
Mode	4	1	1	1	4	8	0.67	1.5	5.33	12	0	0.67
Standard Deviation	77.94	10.97	6.66	8.05	233.23	1817.46	0.27	11.08	32.22	113271.2	0.608	7556.52
Sample Variance	6075.27	120.48	44.42	64.88	54396.31	3303191	0.075	122.95	1038.48	1.28E+10	0.370	57101018
Kurtosis	130.26	136.64	226.42	180.33	156.65	222.89	2.570	14.43	42.88	253.4991	218.094	321.25
Skewness	9.42	9.97	13.30	11.59	10.54	13.15	1.30	2.80	4.60	14.74497	12.958	16.78
Range	1274	179	124	142	3981	33814.56	2	103.53	415.06	2147484	11.27	153047
Minimum	1	1	1	1	1	0	0	0	0	0	0	0
Maximum	1275	180	125	143	3982	33814.56	2	103.53	415.06	2147484	11.27	153047

Table 5. Descriptive statistics about kc2

	lOCode	lOComment	lOBlank	lOCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount
Mean	27.77203	2	4.33908	0.281609	9.197701	14.4659	57.61149	37.02337	8.765134
Median	8	0	1	0	8	7	16.5	11	3
Mode	2	0	0	0	3	1	3	1	1
Standard Deviation	64.43149	5.582052	9.214753	1.038236	6.36018	22.08666	142.9907	90.39862	21.94278
Sample Variance	4151.416	31.15931	84.91168	1.077934	40.45189	487.8206	20446.35	8171.91	481.4855
Kurtosis	157.262	27.67945	67.17269	45.08061	1.991855	78.62646	163.3166	145.6426	139.5504
Skewness	10.4646	4.935493	6.684254	6.016919	1.04648	6.635769	10.79764	10.13254	10.0833
Range	1107	44	121	11	46	325	2468	1513	360
Minimum	0	0	0	0	1	0	1	0	1
Maximum	1107	44	121	11	47	325	2469	1513	361

Table 6. Descriptive statistics about cm1

	loc	v(g)	ev(g)	iv(g)	n	V	l	d	i	e	b
Mean	29.64478	5.382329	2.490763	3.528916	143.9564	900.1758	0.146325	15.82938	38.45536	34884.93	0.304699
Median	17	3	1	2	67.5	329.82	0.09	11.64	27.4	3677.62	0.11
Mode	6	1	1	1	25	11.61	0.04	2	7.74	17.41	0.01
Standard Deviation	42.75357	8.347359	3.658847	5.464398	221.0499	1690.814	0.159337	15.33096	36.9963	134164.7	0.565998
Sample Variance	1827.868	69.6784	13.38716	29.85965	48863.05	2858853	0.025388	235.0383	1368.726	1.8E+10	0.320354
Kurtosis	34.74216	44.04822	20.20538	41.60005	22.73178	33.85464	8.346532	9.430589	10.61536	154.4767	33.16868
Skewness	4.908239	5.463422	3.948111	5.368056	3.91489	4.867111	2.430377	2.476472	2.769378	11.13782	4.802044
Range	422	95	29	62	2074	17124.28	1.3	125.77	293.68	2153691	5.71
Minimum	1	1	1	1	1	0	0	0	0	0	0
Maximum	423	96	30	63	2075	17124.28	1.3	125.77	293.68	2153691	5.71

Table 7. Descriptive statistics about cm1

	t	lOCode	lOComment	lOBlank	lOCodeAnd Comment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount
Mean	1938.056	3.787149	12.28313	11.53414	0.006024	15.1992	25.45221	88.38996	55.57068	9.348193
Median	204.31	1	4	5	0	14	15	42	26	5
Mode	0.97	0	0	0	0	9	8	14	2	1
Standard Deviation	7453.592	8.508658	25.82861	19.98148	0.10012	9.617815	33.92582	134.9175	86.96953	15.07222
Sample Variance	55556027	72.39726	667.1169	399.2594	0.010024	92.50237	1150.961	18202.74	7563.699	227.1718
Kurtosis	154.4768	27.25425	61.25413	22.31643	337.1594	4.799356	19.03281	22.80975	22.08943	37.16227
Skewness	11.13782	4.64599	6.342694	4.11859	17.93496	1.588675	3.612816	3.919459	3.885485	4.970882
Range	119649.5	80	339	164	2	71	314	1260	814	161
Minimum	0	0	0	0	0	1	0	1	0	1
Maximum	119649.5	80	339	164	2	72	314	1261	814	162

Table 9. Correlation table for Kc1

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount
loc	1																				
v(g)	0.902619	1																			
ev(g)	0.718833	0.819965	1																		
iv(g)	0.895167	0.965689	0.77574	1																	
n	0.94851	0.922511	0.746253	0.893768	1																
v	0.938516	0.915556	0.753569	0.88492	0.994956	1															
l	-0.43218	-0.37987	-0.26071	-0.3693	-0.41203	-0.37579	1														
d	0.846844	0.868341	0.698391	0.815358	0.891586	0.860161	-0.4986	1													
i	0.802832	0.673304	0.511383	0.674493	0.826783	0.799857	-0.402	0.679523	1												
e	0.821338	0.862413	0.738864	0.807287	0.908211	0.93493	-0.25663	0.778999	0.592246	1											
b	0.918359	0.896867	0.738853	0.866989	0.973711	0.978958	-0.35603	0.841017	0.780946	0.915576	1										
t	0.821337	0.862413	0.738864	0.807287	0.908211	0.93493	-0.25662	0.778998	0.592245	1	0.915583	1									
lOCode	0.985491	0.91852	0.723255	0.91692	0.96105	0.949684	-0.42118	0.858635	0.814257	0.833695	0.929966	0.833694	1								
lOComment	0.686718	0.517073	0.411757	0.519622	0.547563	0.545146	-0.21956	0.470282	0.462036	0.444171	0.536657	0.444171	0.61415676	1							
lOBlank	0.824427	0.731436	0.624973	0.689669	0.777579	0.772102	-0.34171	0.732524	0.625273	0.687308	0.758482	0.687308	0.77003915	0.605030981	1						
locCodeAndComment	0.365868	0.328405	0.321791	0.304549	0.34756	0.359956	-0.1284	0.296738	0.254271	0.365634	0.365819	0.365636	0.34261273	0.292028118	0.28447719	1					
uniq_Op	0.78349	0.768621	0.616077	0.729753	0.809496	0.771159	-0.56097	0.932447	0.716386	0.632034	0.751686	0.632033	0.79397732	0.446653307	0.68988805	0.281684979	1				
uniq_Opnd	0.915715	0.827906	0.666726	0.803406	0.94588	0.931094	-0.45336	0.844061	0.925089	0.771366	0.910905	0.771365	0.91925681	0.546855272	0.77059197	0.315476818	0.849227	1			
total_Op	0.946597	0.925176	0.747825	0.901752	0.998146	0.992046	-0.414	0.889622	0.824082	0.903576	0.970887	0.903576	0.96118718	0.544072398	0.76843997	0.345526116	0.809307	0.937207395	1		
total_Opnd	0.944167	0.910989	0.73788	0.873885	0.995162	0.991866	-0.40556	0.887768	0.824655	0.908589	0.970853	0.908589	0.95330449	0.548924894	0.78625553	0.348188646	0.803442	0.952468259	0.987337866	1	
branchCount	0.901108	0.998608	0.819791	0.964733	0.923869	0.917383	-0.37941	0.869659	0.672704	0.866195	0.898445	0.866195	0.91687486	0.516180911	0.72964142	0.328974507	0.769751	0.829343826	0.92642448	0.912511978	1

Table 10. Correlation table for Kc2

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount
loc	1																				
v(g)	0.96405	1																			
ev(g)	0.90286	0.92753	1																		
iv(g)	0.96352	0.9766	0.95231	1																	
n	0.98953	0.96328	0.91284	0.96646	1																
v	0.97536	0.95855	0.93412	0.97045	0.99143	1															
l	-0.3363	-0.2882	-0.1883	-0.2698	-0.3113	-0.2484	1														
d	0.80537	0.74759	0.64136	0.73415	0.78456	0.71086	-0.5618	1													
i	0.89005	0.83454	0.7323	0.81271	0.88538	0.84089	-0.4457	0.78372	1												
e	0.91854	0.91953	0.9397	0.94986	0.93985	0.96917	-0.1461	0.62336	0.71292	1											
b	0.9675	0.95161	0.92817	0.96368	0.9839	0.99303	-0.2251	0.70241	0.83118	0.96321	1										
t	0.90108	0.90248	0.94015	0.93929	0.92553	0.96393	-0.1299	0.57456	0.69655	0.99386	0.95814	1									
lOCode	0.99213	0.96382	0.92112	0.96826	0.99494	0.98592	-0.3232	0.79602	0.88216	0.93493	0.97834	0.92214	1								
lOComment	0.74658	0.70237	0.58792	0.69232	0.70383	0.66548	-0.2819	0.70706	0.63826	0.62237	0.6617	0.5751	0.70664962	1							
lOBlank	0.92641	0.85346	0.73417	0.83555	0.91814	0.88716	-0.3642	0.77465	0.88074	0.78674	0.88101	0.76368	0.90852472	0.69491919	1						
locCodeAndComment	0.61406	0.66533	0.55726	0.62869	0.6107	0.58117	-0.1831	0.60827	0.5204	0.57295	0.58733	0.52434	0.6006057	0.56003601	0.52887559	1					
uniq_Op	0.65652	0.59453	0.48731	0.57721	0.61287	0.53562	-0.6919	0.91127	0.70689	0.42722	0.52463	0.39055	0.63240759	0.57982406	0.6364147	0.465166595	1				
uniq_Opnd	0.96489	0.91987	0.84069	0.90638	0.95505	0.92299	-0.4371	0.84499	0.95598	0.82921	0.9145	0.81104	0.9558571	0.69764201	0.91244054	0.600555025	0.75406	1			
total_Op	0.993	0.96633	0.9185	0.9716	0.99632	0.98971	-0.3074	0.77255	0.88164	0.93976	0.98225	0.92714	0.99138348	0.7009068	0.9124159	0.602462988	0.61487	0.95627522	1		
total_Opnd	0.99235	0.96544	0.90998	0.96436	0.99515	0.98497	-0.3133	0.78508	0.88586	0.9338	0.97753	0.91758	0.98906702	0.70037198	0.91543643	0.621224861	0.62317	0.9618085	0.997098	1	
branchCount	0.95967	0.99589	0.92131	0.97232	0.96668	0.96185	-0.2885	0.74849	0.83626	0.91872	0.95479	0.90266	0.96688512	0.70238894	0.85970814	0.661481348	0.58414	0.91573642	0.963253	0.961711846	1

Table 11. Correlation table for Cm1

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount
loc	1																				
v(g)	0.94	1																			
ev(g)	0.77	0.8	1																		
iv(g)	0.91	0.92	0.71	1																	
n	0.94	0.9	0.77	0.87	1																
v	0.95	0.91	0.77	0.88	0.99	1															
l	-0.35	-0.34	-0.28	-0.3	-0.39	-0.34	1														
d	0.72	0.77	0.67	0.67	0.84	0.79	-0.54	1													
i	0.79	0.66	0.55	0.69	0.81	0.79	-0.4	0.51	1												
e	0.81	0.85	0.68	0.82	0.84	0.87	-0.19	0.71	0.46	1											
b	0.94	0.91	0.76	0.88	0.98	0.99	-0.29	0.78	0.78	0.87	1										
t	0.81	0.85	0.68	0.82	0.84	0.87	-0.19	0.71	0.46	1	0.87	1									
lOCode	0.67	0.73	0.6	0.67	0.72	0.73	-0.26	0.61	0.51	0.66	0.72	0.66499	1								
lOComment	0.86	0.79	0.68	0.73	0.79	0.81	-0.28	0.61	0.64	0.68	0.8	0.68329	0.54945	1							
lOBlank	0.67	0.66	0.56	0.64	0.73	0.72	-0.33	0.64	0.58	0.59	0.71	0.59158	0.69361	0.609803	1						
locCodeAndComment	-0.04	-0.02	-0.02	-0.02	-0.03	-0.03	0.39	-0.05	-0.06	-0.01	0.09	-0.0157	-0.015	-0.02478	-0.02977	1					
uniq_Op	0.8	0.8	0.64	0.75	0.83	0.8	-0.62	0.88	0.64	0.65	0.79	0.65666	0.63274	0.68966	0.68698	-0.08817	1				
uniq_Opnd	0.94	0.86	0.71	0.85	0.94	0.94	-0.4	0.69	0.91	0.71	0.93	0.71529	0.67822	0.816564	0.704932	-0.04322	0.81231	1			
total_Op	0.94	0.91	0.77	0.87	0.99	0.99	-0.39	0.84	0.8	0.84	0.98	0.84216	0.71854	0.795034	0.726415	-0.03899	0.84204	0.94339	1		
total_Opnd	0.92	0.89	0.75	0.85	0.99	0.98	-0.39	0.83	0.8	0.83	0.97	0.83914	0.73358	0.786305	0.732367	-0.03774	0.82118	0.94443	0.98415	1	
branchCount	0.94	0.99	0.82	0.91	0.91	0.92	-0.35	0.77	0.69	0.82	0.91	0.82547	0.73789	0.802225	0.665335	-0.03233	0.79994	0.87464	0.9145	0.901754	1

How to cite this paper: Wasiur Rhmann, " Application of Hybrid Search Based Algorithms for Software Defect Prediction", International Journal of Modern Education and Computer Science(IJMECS), Vol.10, No.4, pp. 51-62, 2018.DOI: 10.5815/ijmecs.2018.04.07

