Journal of King Saud University – Computer and Information Sciences 32 (2020) 419–424

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Software fault prediction based on change metrics using hybrid algorithms: An empirical study

Wasiur Rhmann [a,*], Babita Pandey [a], Gufran Ansari [b], D.K. Pandey [c]

[a] CSIT, Babasaheb Bhimrao Ambedkar University, (A Central University), Lucknow, India
[b] B.S. Abdur Rahman Crescent Institute of Science & Technology, India
[c] Lovely Professional University, India

ARTICLE INFO

Article history:
Received 8 December 2018
Revised 31 January 2019
Accepted 6 March 2019
Available online 7 March 2019

Keywords:
Testing
Bug
Defects
Machine learning
Hybrid algorithms
Software metrics

ABSTRACT

Quality of the developed software depends on its bug free operation. Although bugs can be introduced in any phase of the software development life-cycle but their identification in earlier phase can lead to reduce the allocation cost of testing and maintenance resources. Software defect prediction studies advocates the use of defect prediction models for identification of bugs prior to the release of the software. Use of bug prediction models can help to reduce the cost and efforts required to develop software. Defect prediction models use the historical data obtained from software projects for training the models and test the model on future release of the software. In the present work, software change metrics have been used for defect prediction. Performances of good machine learning and hybrid algorithms are accessed in prediction of defect with the change metrics. Android project has been used for experimental purpose. Git repository has been used to extract the v4–v5, v2–v5 of android for defect prediction. Obtained results showed that GFS-logitboost-c has best defect prediction capability.

1. Introduction

As software testing is usually performed under the pressure to deliver the software, Software engineering community has focused on identification of faulty modules prior to the testing of software. Identification of faulty modules helps tester to pay more attention on testing the fault prone modules. Early detection of bugs helps to save cost and efforts. Different software metrics such as: software change matrices (SCM) and code based matrices (CBM) have been used by various researchers to identify the faulty modules in software (Zhou et al., 2010; Choudhary et al., 2018; Malhotra, 2016; Moser and Pedrycz, 2008). Various CBM have been used by researchers to predict faulty modules (Zhou et al., 2010; Choudhary et al., 2018; Malhotra, 2016; Moser and Pedrycz, 2008). SCM matrices are used to measure the change that occurs between two versions of software whereas the CBM is used to measure the size and complexity of code.

Software maintenance is an essential step of software development cycle due to occurrence of bugs, addition or change of features in the software during its life span. Demand of change in software starts after its development and it is managed and controlled by measuring different software metrics for product, process and resources (Fenton and Bieman, 2015). Product and resources used for development of software projects may change with time. Process may be change when a better process is available and advanced tools will be used if available, for adding additional features to the existing software.

The changes in the different attributes may be controlled by measurement of different software metrics. Different software metrics and subsets of these metrics have been used by various researchers for creation of model with the improved fault prediction accuracy (Gondra, 2008).

The main objective of this study is to build fault prediction model based on software change metrics. The following research questions (RQ) are addressed in this study.

Q1. How Hybrid search based algorithms (HSBA) perform in prediction of faults in android software using change metrics?

* Corresponding author.
E-mail address: wasiurrhmann786@gmail.com (W. Rhmann).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2019.03.006
1319-1578/

Q2. What is the difference between the performances of HSBA and Machine learning techniques (MLT) based prediction models for android software?

Q3. Are the different techniques used in study statistically equivalent?

Followings are the main contributions of this work:

- Exploring the use of HSBA for fault prediction using change metrics
- Comparing the performance of HSBA with top performing Machine learning techniques (MLT).
- Statistical analysis of techniques used for fault prediction.

The paper is organized in 8 sections as follows: Introduction is given in Section 1. Related work is discussed in Section 2. Section 3 describes various SCM and datasets. Performance measures have been discussed in Section 4. Implementation details of different algorithms and obtained results have been discussed in Section 5. Sections 6 and 7 contains the answers to various research questions and threats to the study. Section 8 concludes the work.

2. Related work

Choudhary et al. (2018) deployed a MLT for software fault prediction based on SCM. They compared their models with existing models based on CBM for fault prediction. They have used Random forest, J48 and KNN for fault prediction. It was found that Random forest has highest precision 0.73 while J48 and KNN have highest recall 0.375, 0.415 respectively. In the present work, fault prediction models were developed from existing SCM by applying hybrid search based algorithms and MLT. Malhotra (2016) has used MLT for software defect prediction based on Object oriented software metrics (OOSM). They performed the experiments on android software obtained from git repository. They deployed Multilayer perceptron (MLP), Logistic regression (LR) and Support Vector Machines. MLP and LR have shown superior performances as compared to other techniques, whereas SVM showed worst performances.

Moser and Pedrycz (2008) have deployed the MLT for fault prediction and analyzed the predictive capability of some sets of metrics such as: process metrics and code metrics for software defect prediction. Results obtained showed correctly classified classes were above 75% with a recall value more than 80%. It was found that defect prediction models based on process metrics give better performance as compared to code metrics.

Yang et al. (2015) have developed a learning-to-rank model to predict software defects and compared it with other learning based algorithms. Sharma and Chandra (2018) have used soft computing techniques for fault prediction. In our previous work, hybrid algorithms have been used for defect prediction using static code metrics for cross project and with-in defect prediction (Rhmann, 2018a,b).

Kaur and Kaur (2018) have used six machine learning techniques for creation of fault prediction models. They used different software metrics such as C-K, Henderson-Sellers, QMOOD, McCabe's, Martin's metrics etc for fault prediction. Experiments were performed on five open source projects and obtained results showed that Random forest and Bagging techniques have good predication capability while the performance of Naïve Bayes is not so good. Kaur and Kaur (2014) have applied six machine learning techniques to predict faulty classes. They have used five open source software for experimental purpose and used object-oriented metrics as independent variables to predict faulty classes. Obtained results demonstrated that Bagging and J48 classifier have better performance in comparison to other techniques used for fault prediction. Manjula and Florence (2018) have proposed a hybrid approach based on Machine Learning and Genetic algorithm for software defect prediction. They have used Genetic Algorithm (GA) for selection of better features from datasets. Then Decision Tree(DT) model is applied for software defect prediction. Obtained results showed better classification accuracy. Erturk and Sezer (2015) have applied Adaptive Neuro Fuzzy Inference System (ANFIS) to predict faults in software using McCabe software metrics. They also build prediction models with Support Vector Machine (SVM) and Artificial Neural Network (ANN). PROMISE repository has been used to obtain datasets for experimental purpose. ROC-AUC achieved for SVM, ANN and ANFIS techniques were found to be 0.7795, 0.8685, and 0.8573 respectively. The present work deals with the creation of defect prediction model based on software change metrics as independent metric using hybrid search based algorithms and MLT. The performances of these techniques are statistically compared.

3. Defect prediction models

In this section, SCM, datasets and different techniques which are used for creation of defect prediction models are discussed.

3.1. Software change metric (SCM) used for defect prediction

Any change made during the software development life cycle is measured by change or process metrics (Fenton and Bieman, 2015). These software metrics are collected during the whole life-cycle of software project with multiple releases of software. Some process metrics are code churn measures, change bursts, and code deltas. Codes churn metric measures the rate of code evolving. Change burst metric measures the changes which occurred consecutively. Code delta metric measures the difference between two software builds in terms of software metric like LOC. In the work, the 9 SCM as shown below were used as independent variables and given as input to MLT and HBSA. Defective /Non defective class was treated as output.

1. *LOC-ADDED* is the number of lines of program code added to the file
2. *LOC-DELETED* is the number of lines of program code deleted from the file
3. *LOC-CHANGED* is the number of lines of program code changed from the file
4. *MAX-LOC-ADDED* is the maximum number of lines of program code deleted for all commits
5. *MAX_LOC-CHANGED* is the maximum number of lines of program code added to the file
6. *MAX_LOC_DELETED* is the maximum number of lines of program code deleted to the file
7. *CODE CHURN* is defined as sum total of (difference between added lines of program code and deleted lines of program code) for a file considering all of its revisions in repository
8. *MAX CODE CHURN* is defined as maximum of (difference between added program lines and deleted program lines)) for a file in software considering all of its revisions in repository
9. *AVERAGE CODE CHURN* is defined as average of (difference between added program lines and deleted program lines)) for a file in software considering all of its revisions in repository

3.2. Techniques used for defect prediction model creation

In this study, top three machine learning techniques (Choudhary et al., 2018), and top two hybrid algorithms (Rhmann, 2018) have been used for creation of defect prediction models. These techniques are described in the Table 1.

Table 1
Different techniques used for defect prediction.

Technique	Technique name	Description
MLT	Random Forest	A number of decision tree are used to train in Random forest. Random subset of features is used to train various decision trees. The average of the prediction is used to predict.
	Multilayer Perceptron	It is based on backpropagation to train classifier. It contains a number of hidden layers with sigmoid function between input and output layer. Each hidden layer except output layer contains biased neuron. The output is non linear function of the input.
	J48	This technique is C4.5 algorithm implemented in java in Weka and default setting is used for performing experiments.
HSBA	Fuzzy-AdaBost (GFS-Adaboost-c)	This algorithm learns from fuzzy rules. Multiple weak classifiers are combined to make a higher performance classifier. A voting strength is assigned to each weak classifier Jesus et al. (2004).
	Logitboost (GFS-logitboost-c)	This algorithm also uses fuzzy rules to trains classifier. Fuzzy rule base is constructed in incremental manner by down weighting correctly classified training instances Otero and Sanchez (2006)

3.3. Data collection

For experimentation purpose historical data from software projects were used to predict defective classes in the software. The complete process of data collection is shown in Fig. 1. The data was collected from GIT repository. android-4.0.1_r1, android-5.0.0_r1, android-2.0.1_r1, android-5.0.0_r1 versions were used to investigate the performance of different MLT. Change metrics where extracted for android-4.0.1_r1 to android-5.0.0_r1 and for android-2.0.1_r1 to android-5.0.0_r1. For simplicity datasets obtained from android-4.0.1_r1 to android-5.0.0_r1 is written as v4–v5 and dataset for android-2.0.1_r1 – android-5.0.0_r1 is written as v2–v5. Following steps were used to extract the SCM from GIT repository for android project.

1. Git log command git log-numstat-oneline android-4.0.1_r1 android-5.0.0_r1 is used to extract numbers of lines added and deleted in different files between two versions.
2. Select filename, sum(inserted_lines), sum(deleted_lines), sum (changed_lines) from files group by filename; is used for calculation of inserted, deleted and changed lines in LOC for each file.
3. The different change metrics are computed using MySQL query and file which contains bugs are obtained by log information and they are combined with different metrics.

The detail description of dataset including numbers of instances, numbers of defective and non defective classes are given in Table 2.

4. Performance evaluation metrics

Data sets used in the study contains unequal numbers of defective and non defective i.e. datasets are skewed, there is need of careful selection of performance measures. In this study, positive instances are defective classes. Four different terms are used to describe the performance of models

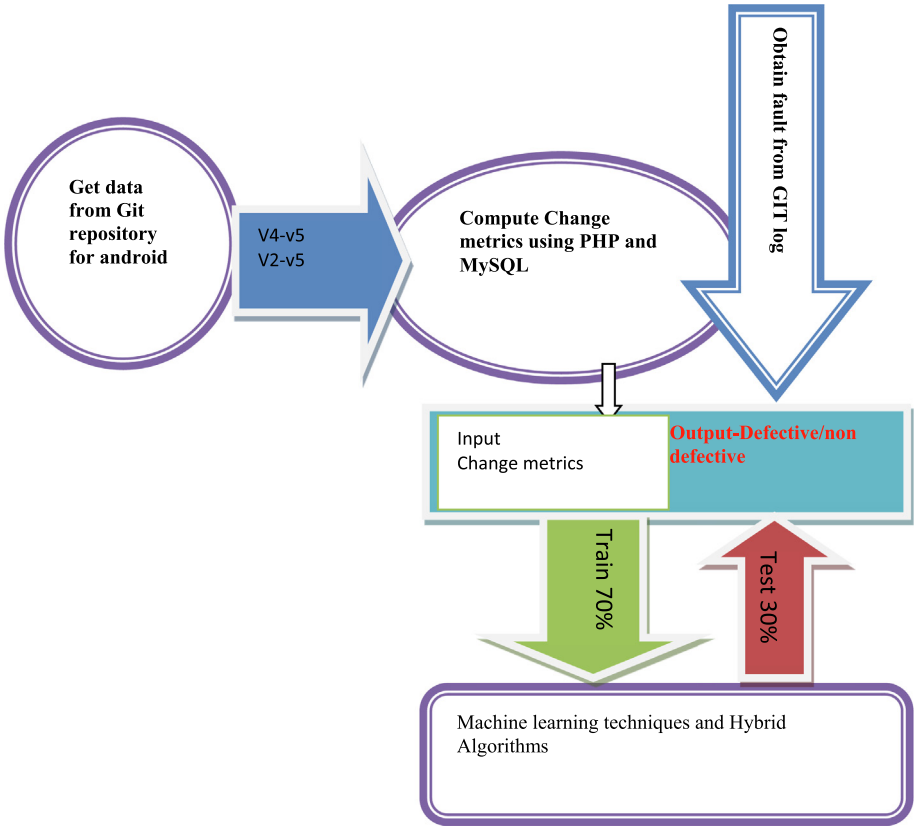

Fig. 1. Software defect prediction model using git change log.

Table 2
Description of datasets.

Dataset	Numbers. of instances	Numbers of defective classes	Number of Non-defective classes
android-4.0.1_r1 – android-5.0.0_r1	170	62	108
android-2.0.1_r1 – android-5.0.0_r1	324	62	262

TP(True Positive) – It is the number of correctly classified defective classes.

TN(True Negative) – It is the number of correctly classified non defective classes.

FP(False Positive) – Number of non defective classes wrongly classified defective classes.

FN (False Negative): Number of defective classes wrongly classified non defective classes.

Precision and recall values are used by various researchers to evaluate the performance of defect prediction models (Kim et al., 2008; Nam et al., 2013).

Precision (Positive Predictive value).

Precision is obtained by dividing numbers of classes correctly classified as defective to total number of defective classes.

Precision = TP/(TP + FP).

Recall (Sensitivity) – It is obtained by dividing the numbers of modules correctly classified as defective to total defective modules.

Recall = TP/(TP + FN).

Values of recall and precision lies in between 0 and 1. Best model will have values of recall and precision equal to 1.

5. Implementation and results

In the study, open source software Weka (http://www.cs.waikato.ac.nz/ml/weka) is used to deploy MLT and open source software keel (http://www.keel.es) is used to develop model based on hybrid algorithms. HSBAs are implemented in open source tool Keel. In the study 10-fold cross validation is used. To perform 10-fold cross validation, dataset is divided into 10 parts, 9 parts are used in training the model and remaining 1 part is used for validation of the model. This process is repeated 10 times and combined result is used as final result for the performance of classifier. Defaults setting of MLT have been used in weka software for performing experiments. Table 3 describes the performances of various MLT used for defect prediction. Due to space limitations we have used top three MLT (Choudhary et al., 2018; Malhotra, 2016) and top two HSBA (Rhmann, 2018) for software fault prediction. MLT used in the study are Random Forest, Multilayer Perceptron and J48. Random forest is Ensemble learning technique, Multilayer Perceptron is Neural network based and J48 is Decision tree based. HSBA used in the study are GFS-Adaboost-c and GFS-LoogitBoost-c.

Table 4 describes the value of control parameters set for implementing HSBA. Table 5 describes the results obtained by HSBA.

Fig. 2 shows a clear picture of comparison of performance of various MLT and HBSA in v4–v5 and v2–v5 datasets. From Fig. 2, it is clear that high recall values are shown by HSBA Algorithm for both databases as comparison to MLT. Whereas, the precision value of RF, J48, GFS-LoogitBoost-c and GFS-AdaBoost-c are much higher for v2–v5 database as compared to v4–v5. Overall Precision and Recall values are lower for v4–v5 as compared to v2–v5 datasets except for HBSA.

Table 3
Performance of MLT.

Dataset	Technique	Precision (P)	Recall (R)
v4–v5	Random Forest (RF)	0.616	0.618
v4–v5	Multilayer Perceptron (MLP)	0.413	0.624
v4–v5	J48	0.524	0.624
v2–v5	Random Forest (RF)	0.820	0.836
v2–v5	Multilayer Perceptron (MLP)	0.616	0.618
v2–v5	J48	0.8 22	0.840

Table 4
Control Parameters for HSBA.

S. No.	Name of SBA	Control parameters
1	GGA-FS	Cross Probability 0.7, Mutation Probability 0.01 Population size 50, Number of evaluation 10,000 Beta Equibilirate factor 0.99, Number of Neighbour 1 Use Elitism Yes
2	GFS-Adaboost-c	labels = 3, rules = 8
3	GFS-LoogitBoost-c	labels = 3, rules = 25

Table 5
Results for HSBA.

Dataset	Technique	Precision (P)	Recall (R)
v4–v5	GFS-AdaBoost-c (GFS-AB-c)	0.646	0.963
v4–v5	GFS-LoogitBoost-c (GFS-LB-c)	0.646	0.963
v2–v5	GFS-Adaboost-c (GFS-AB-c)	0.815	0.992
v2–v5	GFS-LoogitBoost-c (GFS-LB-c)	0.822	0.992

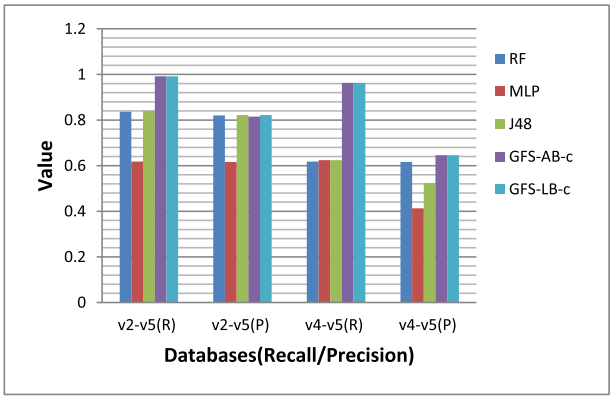

Fig. 2. Precision obtained by deploying various MLT and HBSA in v4–v5 and v2–v5 datasets.

The correlation coefficient among the independent variable of v4–v5 dataset is positive as shown in Table 6, which indicate that the independent variables are redundant. Therefore, preprocessing step for dimensional reduction is required (Raschka and Mirjalili, 2017). This is the reason, the MLT and shows poor performance on v4–v5 dataset.

The correlation coefficient among v4–v5 is negative as shown in Table 7. This indicates that the independent variables are negatively correlated or they are non-redundant. Therefore, preprocessing of dataset is not required and thus MLT and HBSA shows good performance in v_2–v_5 dataset.

5.1. Friedman test

Friedman test is a nonparametric test. It doesn't require sample to drawn from normal population. In this test, ranks have been assigned to different model prediction techniques based on their

Table 6
Correlation among the independent variable for v_4–v_5.

Metric/Metric	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	1.0								
M2	1.0	1.0							
M3	1.0	1.0	1.0						
M4	1.0	1.0	1.0	1.0					
M5	1.0	1.0	1.0	1.0	1.0				
M6	1.0	1.0	1.0	1.0	1.0	1.0			
M7	0.9	0.8	0.9	0.9	0.8	0.9	1.0		
M8	0.9	0.8	0.9	0.9	0.8	0.9	1.0	1.0	
M9	0.9	0.8	0.9	0.9	0.8	0.9	1.0	1.0	1.0

Table 7
Correlation among the independent variable for v_2–v_5.

Metric/Metric	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	1								
M2	−0.02	1							
M3	0.84	−0.03	1						
M4	1	−0.02	0.84	1					
M5	−0.02	1	−0.03	−0.02	1				
M6	0.84	−0.03	1	0.842	−0.03	1			
M7	1	−0.02	0.84	1	−0.02	0.84	1		
M8	1	−0.02	0.84	1	−0.02	0.84	1	1	
M9	1	−0.02	0.84	1	−0.02	0.84	1	1	1

Table 8
Ranks of different techniques obtained by Friedman test based on precision.

Technique	RF	MLP	J48	GFS-AB-c	GFS-LB-c
Mean Rank	3.00	1.00	3.25	3.25	4.50

Table 9
Ranks of different techniques based on recall.

Technique	RF	MLP	J48	GFS-AB-c	GFS-LB-c
Mean Rank	1.50	1.75	2.78	4.50	4.50

defect prediction capability on different datasets. This test is used to check whether there is any statistical difference or not between different techniques.

Ho: Different techniques used in the study are statistically equivalent in terms of their defect prediction capability.

Ha: Different techniques used in the study are statistically different in terms of their defect prediction capability.

Tables 8 and 9 gives the mean rank of different techniques obtained by applying Friedman test.

Test statistics used are N = 2, Chi-Square = 5.368, df (degree of freedom) = 4, p-value = 0.252. Since p value is greater than significance level (0.05). Therefore, null hypothesis is accepted.

Test Statistics used are N = 2, Chi-Square = 7.243, df (degree of freedom) = 4, p-value = 0.124. Since p value is greater than significance level (0.05). Therefore, null hypothesis is accepted. Hence the performance of different techniques used for defect predictions are statistically same.

6. Answer to various research questions

The followings are answers to various research questions:

Q1. What are the performances of HBSA in fault prediction using SCM for android software?

Answer: In the both HBSA used for fault prediction the performance of GFS-LoogitBoost is better than other in terms of precision and recall.

Q2. What is the difference between the performances of HBSA vs. MLT based prediction models for android software?

Answer: High precision and high recall values are shown by HBSA for both datasets in comparison to MLT. So HBSA are good for fault prediction with SVM.

Q3. *Are the different techniques are statistically equal?*

Answer: As the null hypothesis is accepted for precision and recall, hence there is no statistical difference between the performances of different techniques.

7. Threats to the validity

As in this work android project is used, which is based on java so there is possibility of threat to generalize results for other languages like C, C++ etc. Internal threat also lies as software size may also affect the probability of a class being faulty. Software size is another factor which may influence the effect of use of SCM on prediction of defective modules. The metrics used in the study are well established and they are computed using MySQL query so there is no construct validity threat in the study.

8. Conclusions

Identification of defective classes in software has gained attention due to its effectiveness in reduction of testing efforts. Historical data is used to predict the defects in future release of the

software. In the present work, SCM have been explored to predict faulty classes using MLT and HBSA. On using Hybrid algorithms, GFS-loogitboost has shown best performance in terms of precision and recall. Friedman statistical test is used to assess whether the performances of different techniques are statistically different or not. It was found that different techniques used for defect prediction are statistically same in terms of their performance. There is need to perform experiments on different types of large data sets to generalize results.

Conflict of interest

None.

References

Choudhary, G.R., Kumar, S., Kumar, K., Mishra, A., Catal, C., 2018. Empirical analysis of change metrics for software fault prediction. Comput. Electr. Eng., Elsevier 67, 15–24.

Erturk, E., Sezer, E.A., 2015. A comparison of some soft computing methods for software fault prediction. Expert Syst. Appl., Elsevier 42 (4), 1872–1879.

Fenton, N., Bieman, J., 2015. Software Metrics. A Rigorous and Practical Approach. CRC Press, Taylor and Francis group.

Gondra, I., 2008. Applying machine learning to software fault-proneness prediction. J. Syst. Softw. 81, 186–195.

Jesus, M.J., Hoffmann, F., Navascues, L.J., Sunchez, L., 2004. Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms. IEEE Trans. Fuzzy Syst. 12 (3), 296–308.

Kaur, A., Kaur, I., 2014. Empirical evaluation of machine learning algorithms for fault prediction. LNSE Lecture Notes Software Eng. 2 (2), 176–180.

Kaur, A., Kaur, I., 2018. An empirical evaluation of classification algorithms for fault prediction in open source projects. J. King Saud Univ.-Comput. Inf. Sci., Elsevier 30, 2–17.

Kim, S., Whitehead, E.J., Zhang, Y., 2008. Classifying software changes: clean or buggy. IEEE Trans. Softw. Eng. 4 (2), 181–196.

Malhotra, R., 2016. An empirical framework for defect prediction using machine learning techniques with Android software. Appl. Soft Comput., Elsevier 49 (C), 1034–1050.

Manjula, C., Florence, L., 2018. Hybrid approach for software defect prediction using machine learning with optimization technique. Int. J. Comput. Inf. Eng., World Acad. Sci. Eng. Technol. 12 (1), 28–32.

Moser, R., Succi, Pedrycz W., 2008. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction May 10–18. ICSE'08, Leipzig, Germany, pp. 181–190.

Nam, J., Pan, S.J., Kim, S., 2013. Transfer defect learning. Proc. of Int'l Conf. on Softw. Eng. (ICSE'13), pp. 382–391.

Otero, J., Sanchez, L., 2006. Induction of descriptive fuzzy classifiers with the Logitboost algorithm. Soft Comput., Springer 10, 825–835.

Raschka, S., Mirjalili, V., 2017. Python Machine Learning. Published by Packt Publishing Ltd.

Rhmann, W., 2018b. Cross project defect prediction using hybrid search based algorithms. Int. J. Inf. Technol., Springer,. https://doi.org/10.1007/s41870-018-0244-7.

Rhmann, W., 2018a. Application of hybrid search based algorithms for software defect prediction. Int. J. Modern Educ. Comput. Sci., MECS 10 (4), 51–62. https://doi.org/10.5815/ijmecs.2018.04.07.

Sharma, D., Chandra, P., 2018. A comparative analysis of soft computing techniques in software fault prediction model development. Int. J. Inf. Technol., Springer 1–10. https://doi.org/10.1007/s41870-018-0211-3.

Yang, X., Tang, K., Yao, X., 2015. A learning-to-rank approach to software defect prediction. IEEE Trans. Reliab. 64 (1), 234–246.

Zhou, Y., Xu, B., Leung, H., 2010. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. J. Syst. Softw. 83, 660–674.