Improving Software Maintainability Predictions using Data Oversampling and Hybridized Techniques

Ruchika Malhotra
Discipline of Software Engineering, Department of Computer Science and Engineering
Delhi Technological University, Delhi, India
ruchikamalhotra2004@yahoo.com

Kusum Lata
Department of Computer Science and Engineering,
Delhi Technological University, Delhi, India
kusumlata@dtu.ac.in

***Abstract*— Software systems being developed in today's era are inherently large and complex. Maintaining these software is a big challenge before the software industry. Since software maintenance demands a high cost, this activity becomes even more challenging. In order to reduce the maintenance cost, it becomes crucial to know the maintainability of software modules/ classes in the initial stages of software development. Many efforts have been made to identify the maintainability of modules in the initial development stages in which prediction models have a lot of roles. Prediction models are trained from past historical data and should consist of an adequate number of instances of low maintainability and high maintainability class/modules. But this is not usually the case, and because of this, we are not able to train the prediction models properly. This situation shows data imbalance. In this direction, in this paper, we will handle the imbalanced data problem so that prediction models can be properly trained. We apply four oversampling techniques in this paper and train the prediction models for maintainability by hybridized techniques. The results of the paper advocate the effectiveness of examined oversampling techniques along with the hybridized classification techniques to develop competent maintainability prediction models.**

Keywords—Software Maintenance, Imbalanced data, Hybridized techniques.

I. INTRODUCTION

Maintenance is an essential dimension of software quality. Maintainability means how easily a software module or class can be modified [1]. Since the size and complexity of software systems are increasing due to technological progress, it becomes very important to develop maintainable software. It is also necessary to develop software in a maintainable way because maintenance charges a good cost and effort of software development [2]. But in reality, the maintainability of a software component can be determined in the actual maintenance phase. At that stage, it would be very problematic to control the cost of maintenance. The cost of maintenance can be controlled if we come to know in the initial development stages, how maintainable a particular module be? In the literature, researchers have made many efforts to forecast software maintainability in the initial phases of software development, and through their efforts, many maintainability prediction models have been developed [3-7]. Software metrics that are the measurable factors have been used as the independent variables to develop models to predict software maintainability with the application of different statistical and ML techniques [8-10]. The software maintainability prediction models are trained from past historical data comprising of data points. Each data point consists of software metrics capturing different characteristics of class/module and maintainability.

Maintainability of a software class/module will be low if more revisions in the lines of source code are needed to modify that module in the maintenance phase and vice versa [11]. It is very important that we have adequate of instances of low maintainability and high maintainability class in order to train any predictive model efficiently. In the real scenario, the low maintainability classes are far less in number as compare to high maintainable classes. This means we have less number of data points representing one class (low maintainability) and more number of data points representing another (high maintainability) class. If this is the situation, we will call it a data imbalance problem. In the data imbalance situation, the class with a small number of instances is called a minority class, and the class whose instances are more is called the majority class. In context to imbalanced data problem in this paper, low maintainability class is referred to as minority class, and high maintainability class is the majority class. in this paper, accurate prediction of low maintainability class is our prime concern as they increase the cost of maintenance.

Many competent machine learning (ML) and statistical techniques fail to predict the data points of each class accurately when the data is imbalanced [12]. Realizing how important it is to handle the data imbalance problem, in this paper, we apply four oversampling techniques. The examined oversampling techniques help to balance data by increasing the minority class data points. Also, we use hybridized classification techniques to construct prediction models for software maintainability. As we know that these days search-based techniques are becoming quite popular. These techniques search for a vast solution space and find an optimal solution [13-14]. For developing prediction models for maintainability, we use hybrid classification techniques in this study. These techniques associate search-based and ML techniques as one single technique to be used for a prediction task [15]. Studies in the past have examined the remarkable capability of hybrid classification techniques in software quality predictive modeling tasks [7][16-17].

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

However, very few studies in the past have applied these techniques for software maintainability prediction.

Therefore, the prime contributions of this study are:

(i) Effective pre-processing the imbalanced data by oversampling before the development of models

(ii) To construct prediction models for predicting the low maintainability classes with the aid of hybrid classification techniques after data oversampling. We address the following research questions.

RQ#1) When the data is imbalanced, what is the predictive performance of software maintainability prediction models? Do oversampling techniques improve the performance of software maintainability prediction models?

RQ#2) Which is the best oversampling to develop competent software maintainability prediction models? Is the best oversampling techniques, statistically better than other examined oversampling techniques?

We use three application packages of Apache open-source software, namely Apache Betwixt, Apache IO, and Apache Ivy, in this study for our experiment. We validate the training and testing of models by ten-fold cross-validation and assess the performance of constructed models with the help of GMean and Balance metrics.

The layout of the various section of the study is depicted as follows: Section II is designated for related work. Section III explains the research methodology. Section IV is dedicated to the experimental setup. We present results and their analysis in section V. In section VI, we document threats to validity and finally conclude the paper in section VII.

II. Related Work

There are a lot of studies in the literature wherein the software maintainability prediction is made by validating the software metrics quantify different characteristics of the software. The data imbalance problem is not addressed in any study as these studies predict the maintainability in the form of a continuous variable (number of changes made to code in the maintenance period)[3-7]. The aim of the present study is to predict low maintainability classes efficiently by addressing the data imbalance problem. The data imbalance problem has been dealt with in software quality prediction for software defect prediction and change prediction by various researchers.

The study by Pelayo and Dick [18] used synthetic minority oversampling technique (SMOTE) to develop competent software defect prediction from imbalanced defect datasets. Siers and Islam [19] employed a cost-sensitive classification for defect-prediction in the imbalanced scenario. The study by Zheng [20] proposed a cost-sensitive boosting technique to boost the performance of neural networks trained from imbalanced defect data. Tan et al. [21] pre-processed imbalanced datasets by using different data resampling techniques and constructed competent change-prediction models. A study by Malhotra and Kamal [22] applied various data oversampling methods to improve the performance of defect prediction models. Bashir et al. [23] pre-processed the imbalanced defect prediction datasets by taking care of borderline and noisy datapoints. It is to be noted that all of the above studies addressed the issue of imbalanced data while predicting defective module/classes.

For constructing software quality prediction modes, many studies have applied hybridized classification techniques. In the literature, there are some studies [7][16-17] that ascertain the applicability of hybridized classification techniques for developing software quality prediction modes. According to a literature review conducted by Malhotra and Khanna [22] on 61 studies in the field of software quality domain, hybrid techniques have the remarkable predictive capability. Still, very few studies have used these techniques for software maintainability prediction. It inspires us to use these techniques to construct maintainability prediction models. So in this study, we first balance the imbalanced datasets by using oversampling techniques namely: Synthetic Minority Oversampling Technique: SMOTE [25], Borderline Synthetic Minority Oversampling Technique: BSMOTE [26], Safe-Level Synthetic Minority Oversampling Technique: SSMOTE [27], and Adaptive Synthetic Oversampling technique: Adasyn [28]. We then develop maintainability prediction models by four hybridized techniques namely Fuzzy Learning based on Genetic Programming Grammar Operators and Simulated Annealing (GFS-SP) [29], Fuzzy Learning based on Genetic Programming Grammar Operators (GFS-GPG) [29], Fuzzy Learning based on Genetic Programming (GFS-GP) [29], and Fuzzy rule approach based on genetic cooperative-competitive learning (GFS-GCCL)[30].

III. Research Methodology

In this section, the procedure of data collection, software metrics used as predictor variables to construct software maintainability models, and performance metrics to evaluate the performance of models is briefly explained.

A. Data Collection Procedure

We have used three application packages of Apache, namely Apache Betwixt, Apache Io, and Apache Ivy, in this study. We have collected datasets corresponding to these application packages with the Defect Collection and Reporting (DCRS) tool [31]. Two consecutive versions of the software fed to the DCRS tool as input. In the output, we get as many data points as there are the common classes in the analyzed versions. Each such datapoint consist of eighteen software metrics, number of lines of code added (*LSoC_added*) from the previous version to next version, number of lines of code deleted (*LSoC_deleted*), number of lines of code modified (*LSoC_modified*), and the total number of lines of code changes (*LSoC_total*). We then transform *LSoC_total* variable into the Maintainability variable after discretizing. The variable Maintainability would be a binary variable with two values: Low-maintainability (LOWM) and High-maintainability (HIGHM). Here, LOWM means that a particular class will demand more revisions in its lines of code as compared to HIGHM class. In Table I, we enlist all the three software used in this paper including the name of the software, its version, number of common classes in the versions analyzed (NCC), number of low maintainability classes (NLM), Number of high maintainability classes (NHM), and imbalance ratio (IBR) where IMR is (NHM/NLM).

B. Software Metrics used as Predictors

Eighteen software metrics qualifying different characteristics are used as predictors in this study.

Table I. Software project details

Software	Version	NCC	NLM	NHM	IBR
Apache Betwixt	0.6 – 0.7	245	27	218	08.07
Apache IO	2.0.1 – 2.2	248	10	238	23.80
Apache Ivy	1.4.1 – 2.2.0	371	28	343	13.25

These metrics are reported in Table II. Due to space constraint, we have not given the detailed definition of each metric. The detailed definition of metrics described in Table II can be referred from [32]. The response or dependent variable is binary variable: Maintainability that consists of two values, LOWM and HIGHM.

C. Assessment in Software Maintainability Prediction

For a binary classification problem like in this paper, the performance of models is judged with the confusion matrix given in Table III. The confusion matrix contains four quantities: True Positive (t-pos), False Positive (f-pos), True Negative (t-neg), and False Negative (f-neg). The values of various performance measures like True Positive Rate (TPRate), False Positive Rate (FPRate), True Negative Rate (TNRate), GMean, and Balance, etc. can be computed from the values given in confusion matrix. We use GMean and Balance performance measures in this study to record the performance of software maintainability prediction models as the use of these metrics is advocated for the imbalanced data problem [12]. Further, in this study, Positive class is corresponding to LOWM class, and the Negative class corresponds to HIGHM class.

GMean is the geometric mean of TPRate and TNRate. Balance measures the Euclidian distance between the pair of TPRate and FPRate and the optimal value of TPRate and FPRate.

$$TPRate = \frac{t-pos}{t-pos+f-neg} \qquad \text{(i)}$$

$$NRate = \frac{t-neg}{t-neg+f-pos} \qquad \text{(ii)}$$

$$GMean = \sqrt{TPRate * TNRate} \qquad \text{(iii)}$$

$$Balance = 1 - \sqrt{\frac{(0-FPRate)^2 + (1-TPRate)^2}{2}} \qquad \text{(iv)}$$

$$where\ FPRate = \frac{f-pos}{f-pos+ t-neg} \qquad \text{(v)}$$

IV. EXPERIMENTAL SETUP

This segment of the study provides information regarding the experimental setup. As discussed in section III, we have used three datasets extracted from Apache application packages details of which are provided in Table I. As the datasets are highly imbalanced as depicted from the values of IBR in Table I, we pre-process the datasets with the help of four oversampling techniques. The description of the oversampling techniques given ahead in this section. After data oversampling, for developing the software maintainability prediction models, we use hybridized classification techniques described in the last part of this section. During the course of training the models, we use a ten-fold cross-validation approach that divides the training dataset into ten partitions, out of which nine partitions are used for training the model while on the remaining one partition model is tested. We assess the performance of developed models with the performance measures discussed in section III and perform statistical analysis with the Friedman test [33] and the Wilcoxon Signed-Rank test [33].

A. Techniques used for data oversampling

To deal with imbalanced data, we have used four oversampling techniques whose concise description is as follows:

(i) Synthetic Minority Oversampling Technique (SMOTE): This technique generates artificial or synthetic data points of minority class along the direction line joining minority class datapoint and its k-nearer neighbours.

Table II. Software metrics used

Software Characteristic	Metric	Description
Size	WMC NPM LOC DAM AMC	Weighted methods per class Number of public methods Lines of source code Data access metric Average method complexity
Coupling	CBO RFC Ca Ce IC CBM	Coupling between objects Response for class Afferent coupling Efferent coupling Inheritance coupling Coupling between the class methods
Cohesion	LCOM, LCOM3	Lack of cohesion among methods
Inheritance	CAM NOC DIT MFC	Cohesion among methods of class Number of children Depth of inheritance Measure of functional abstraction
Composition	MOA	Measure of aggregation

Table III. The Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	t-pos	f-neg
Actual Negative	f-pos	t-neg

(ii) Borderline-Synthetic Minority Oversampling technique (BSMOTE): As an extension of SMOTE, in this technique, only borderline minority class datapoints are increased. If *k*-nearer neighbors class datapoints of a minority belong to majority class, that minority class datapoint is said to be borderline.

(iii) Safe-Level Synthetic Minority Oversampling technique (SSMOTE): This technique is another enhancement of basic SMOTE. Before performing oversampling of minority class datapoint, its safe-level is determined. For a minority class datapoint, D, safe-level of D would be equal to the number of minority class instances in the *k*-nearer neighbors of D.

(iv) Adaptive Synthetic Oversampling technique (Adasyn): Unlike SMOTE and its variants, in this technique oversampling of minority class is done in accordance with its degree of difficulty in learning. A difficult to learn datapoint is oversampled by creating more number of data points corresponding to it and vice versa.

B. *Hybridized Classification Techniques.*

The techniques used in this study for learning training software maintainability prediction models are fuzzy rule-based classifications techniques, whose brief description is as follows: GFS-GP and GFS-GCCL techniques use genetic programming (GP) for classification with fuzzy rules. GP starts with a population of randomly generated solutions and iteratively converts a population of solutions into a new generation of the population by applying genetic operations. GFS-GPG uses grammar-based GP for discovering effective fuzzy rules. GFS-SP uses a simulated annealing based search in order to search for an effective solution in fuzzy rule bases.

V. RESULTS AND ANALYSIS

A. *RQ#1) When the data is imbalanced, what is the predictive performance of software maintainability prediction models? Do oversampling techniques improve the performance of software maintainability prediction models?*

In order to answer this research question, we develop software maintainability prediction models using hybridized techniques in two steps (i) by using original imbalanced datasets (ii) after applying oversampling techniques to imbalanced datasets. In both of the scenarios, we assess the performance of the developed models with respect to GMean and Balance.

Table IV. GMean results of Software Maintainability Prediction Models

	GFS-GCCL	GFS-GP	GFS-GPG	GFS-SP	Mean
Apache Betwixt					
Adasyn	71.18	66.50	69.60	68.41	**68.92**
BSMOTE	66.40	56.53	60.73	57.51	**60.29**
SSMOTE	70.67	65.95	66.50	68.63	**67.94**
SMOTE	71.68	61.33	64.28	68.72	**66.50**
No-resampling	0.00	19.25	19.20	18.93	14.34
Apache IO					
Adasyn	74.81	79.28	72.56	78.39	**76.26**
BSMOTE	73.79	68.60	53.84	68.75	**66.25**
SSMOTE	78.00	79.28	77.27	76.50	**77.76**
SMOTE	77.35	78.00	79.07	66.10	**75.13**
No-resampling	0.00	54.66	54.54	54.31	40.88
Apache Ivy					
Adasyn	69.58	85.59	68.49	64.56	**72.06**
BSMOTE	69.19	65.85	63.10	66.27	**66.10**
SSMOTE	61.33	61.36	68.91	66.08	**64.42**
SMOTE	65.85	67.49	68.86	65.31	**66.88**
No-resampling	26.73	26.73	26.71	26.65	26.70

The performance of the models after employing oversampling techniques along with No-resampling is recorded in Table IV and Table V, respectively, for GMean and Balance. In Tables IV and V, No-resampling refers to the case when Software maintainability prediction models are developed using imbalanced datasets. As shown in Table IV and Table V, in the No-resampling situation, the performance of the models is very poor. The average GMean for Apache Betwixt, IO, Ivy was 14.34, 40.88, and 26.70, respectively. Similarly, average Balance values were reported to be merely 31.25, 45.20, and 34.34 for Apache Betwixt, IO, and Ivy, respectively. For Apache Betwixt dataset, GMean values ranged from 56.53 to 71.68 after oversampling. For Apache IO and Ivy datasets, GMean values ranged from 53.84 to 79.28 and 61.33 to 85.59, respectively, as shown in Table IV. Similarly, as reported in Table V, For Apache Betwixt dataset, Balance values ranged from 55.41 to 71.31 after oversampling. For Apache IO and Ivy datasets, Balance values ranged from 50.45 to 79.27 and 60.54 to 82.25, respectively. After oversampling, for Apache Betwixt dataset, mean GMean values ranged from 60.29 to 68.92, and the Adasyn technique has reported the highest GMean of 68.92. For Apache IO, the range of average GMean values was 66.25 to 77.76 after applying oversampling techniques, and SSMOTE has the highest average of GMean of 77.76, and in case of Apache Ivy, the range of average GMean values was 64.42 to 72.06 with Adasyn of highest average GMean of 72.06 as shown in Table IV. Similarly, on analyzing Table V, we observe very much improvement in the performance of software maintainability prediction models with respect Balance performance metric.

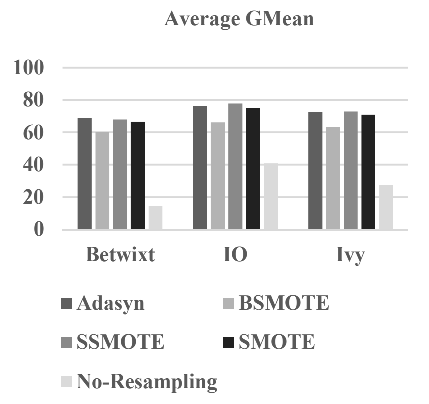

Fig.1 Average GMean of all datasets after oversampling.

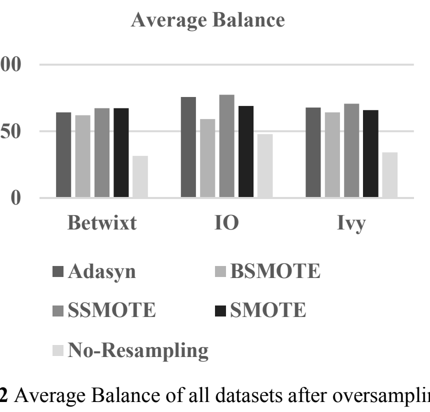

Fig.2 Average Balance of all datasets after oversampling.

For Apache Betwixt dataset, average Balance values ranged from 59..15 to 68.68 and Adasyn technique has reported highest average Balance of 68.68 For Apache IO, the of was average Balance values ranged from 62.56 to 77.36 and SSMOTE has highest average Balance of 77.76 and in case of Apache Ivy the range of average Balance was 64.19 to 71.04 with Adasyn of highest average Balance of 71.04, as shown in Table V. Fig.1 and Fig.2 show the average GMean and average Balance performance of all three datasets after applying oversampling along with No-resampling. Therefore, it is clearly evident from Fig.1 and Fig.2 that investigated oversampling techniques improve the predictive performance of the software maintainability prediction models.

The analysis the results of the developed software maintainability prediction models before and after oversampling, it is evident that the oversampling of the datasets improved the performance of the models.

Table V. Balance results of Software Maintainability Prediction Models

	GFS-GCCL	GFS-GP	GFS-GPG	GFS-SP	Mean
Apache Betwixt					
Adasyn	70.77	66.40	69.16	68.38	**68.68**
BSMOTE	65.35	55.41	59.78	56.04	**59.15**
SSMOTE	70.56	65.53	66.40	67.41	**67.48**
SMOTE	71.31	61.20	64.07	68.49	**66.27**
No-resampling	29.29	31.91	31.91	31.87	31.25
Apache IO					
Adasyn	72.34	79.27	72.48	76.48	**75.14**
BSMOTE	70.97	64.4	50.45	64.43	**62.56**
SSMOTE	77.94	79.27	76.38	75.83	**77.36**
SMOTE	77.25	77.71	78.84	63.54	**74.33**
No-resampling	29.29	50.5	50.5	50.49	45.20
Apache Ivy					
Adasyn	69.55	82.25	67.90	64.47	**71.04**
BSMOTE	69.18	65.65	62.88	66.25	**65.99**
Safe-SMOTE	60.54	61.27	68.90	66.06	**64.19**
SMOTE	65.65	67.40	68.66	65.16	**66.72**
No-resampling	34.34	34.34	34.34	34.34	34.34

Table VI. Friedman Test Results

Oversampling Techniques	Rank with respect to GMean	Rank with respect to Balance
Adasyn	Rank 1 (3.96)	Rank 1 (4.04)
SSMOTE	Rank 2 (3.79)	Rank 2 (3.71)
SMOTE	Rank 3 (3.67)	Rank 3 (3.67)
BSMOTE	Rank 4 (2.5)	Rank 4 (2.50)
No Resampling	Rank 5 (1.08)	Rank 5 (1.08)
p-value	0.00	0.00

B. RQ#2) Which is the best oversampling technique to develop competent software maintainability prediction models? Is the best oversampling techniques, statistically better than other examined oversampling techniques?

In order to answer this research question, we analyze the performance of each oversampling technique on the software maintainability prediction models by two non-parametric statistical tests, Friedman Test and Wilcoxon Signed Rank Test on the GMean and Balance results at a significance level of α = 0.05. First, we statistically analyze the performance of prediction models with the aid of the Friedman test that assigns a mean rank to each technique according to the performance of each oversampling technique in terms of GMean and Balance. The results of the Friedman test are documented in Table VI for GMean and Balance. As shown in Table VI, the results of the Friedman test for investigated oversampling are significant both for GMean and Balance, p-value < 0.05. Adasyn technique has obtained the highest rank with respect to GMean and Balance. The second best rank is obtained by SSMOTE technique. The results for the Friedman test indicate that No Resampling situation has obtained the worst rank both for GMean and Balance measures. To investigate the second part of this RQ, we further carry our investigation with the help of the Wilcoxon Signed-Rank test with Bonferroni correction (= 0.05/4=0.0125). We make a pairwise comparison for the best oversampling (Adasyn) technique with other oversampling techniques used in the study. The results of the Wilcoxon Signed-Rank test are recorded in Table VII.

Table VII. Wilcoxon Test Results for Oversampling Techniques

Resampling Technique Pair	p-value : GMean	p-value : Balance
Adasyn-SSMOTE	0.477 (Not Sig+)	0.657(Not Sig+)
Adasyn-SMOTE	0.308 (Not Sig+	0.347(Not Sig+)
Adasyn-BSMOTE	0.005 (Sig+)	0.005(Sig+)

As shown in Table VII, the Adasyn technique is statistically different (Sig+) from BSMOTE (p-value < 0.0125), both with respect to GMean and Balance. However, from Table VII, it can be seen that the Adasyn technique is not statistically different (Not Sig+) in terms of its performance from SSMOTE and SMOTE.

On statistically analysing on the performance of the software maintainability prediction models we observed that Adasyn, SSMOTE, and SMOTE are equally competent oversampling techniques for developing effective models to predict low maintainability classes accurately.

VI. THREATS TO VALIDITY

We have developed software maintainability prediction models in this study by pre-processing imbalanced datasets with the help if oversampling techniques and evaluated the performance of the developed modes with respect to robust and stable evaluators, GMean and Balance. Therefore, there is no internal validity threat. Also, we don't have the threat to construct validity in this study as the software metrics that are used to construct the prediction models have been widely validated in the software engineering domain. We have reported the results after rigorous statistical analysis performed with the non-parametric statistical test. Therefore, the threat to conclusion validity not exists in this study. The datasets used in this study are extracted from the application packages of Apache written in Java programming language. So, there may be the threat that our result can vary for software developed in other programming languages.

VII. CONCLUSIONS OF THE STUDY

Developing the competent prediction models to correctly predict the low maintainability classes is the prime objective of this study. Because the early prediction of such classes help to enhance the software quality by rigorous testing such classes. However, the imbalanced data poses difficulty to train a model for this task. This motivates us to develop competent models by pre-processing the imbalanced datasets.

With this objective, this paper examines the effect of oversampling techniques for developing prediction models to predict software maintainability. We conducted experiments on three datasets extracted from different Apache application packages, four oversampling techniques, and four hybridized classification techniques. The predictive of software maintainability prediction models that use No resampling (imbalanced datasets) were compared with the models that use oversampling methods with respect to GMean and Balance. The results are statistically validated with the help of the Friedman test and Wilcoxon Signed Rank test. The results of our experiment support that oversampling increases the predictive performance of the software maintainability prediction models. The Adaysn, SSMOTE, and SMOTE techniques proved to be the best oversampling techniques to balance the data distribution and develop competent software maintainability prediction models. In order to enhance the generalizability of our results, in future we plan to broaden the present investigation by collecting more diverse datasets. In this study, we investigated the data level solution to imbalanced data problem. We plan to undertake a few algorithmic level solutions to the imbalanced data problem for software maintainability prediction. We also plan the extend our work for inter-project and inter-release validation.

REFERENCES

[1] IEEE Std. 610.12-1990. Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, Los Alamitos, CA, 1993.

[2] I. Sommerville, Software Engineering, Addison-Wesley, 1995.

[3] M. O. Elish, H. Aljamaan and I. Ahmad, I. "Three empirical studies on predicting software maintainability using ensemble methods," *Soft Computing*, vol. 19, no. 9, 2015, pp. 511-2524.

[4] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *Journal of Systems and Software*, vol. 80, no. 8, 2007, pp. 1349-1361.

[5] T. S. Quah and & M. M. T. Thwin, "Application of neural networks for software quality prediction using object-oriented metrics, " *In Proceeding of IEEE International Conference on Software Maintenance,* ICSM 2003, pp. 116-125.

[6] H. Alsolai and M. Roper, "Application of Ensemble Techniques in Predicting Object-Oriented Software Maintainability," *In Proceedings of the Evaluation and Assessment on Software Engineering,* ACM, 2019, pp. 370-373.

[7] L. Kumar and S.K. Rath, "Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 2, pp. 1487-1502.

[8] A. Kaur and K. Kaur, K. "Statistical comparison of modelling methods for software maintainability prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 6, 2013, pp. 743-774.

[9] W. Zhang, L. Huang, V. Ng, and J. Ge, "SMPLearner: learning to predict software maintainability". *Automated Software Engineering*, vol. *22,* no. 1, 2015,pp. 111-141

[10] C. Van Koten and A. Gray, "An application of bayesian network for predicting object-oriented software maintainability," *Information and Software Technology*, vol. 48, no. 1, 2006, pp. 59-67.

[11] J. Al Dallal, "Object-oriented class maintainability prediction using internal quality attributes," *Information and Software Technology*, vol. *55,* no. 11,2013, pp. 2028-2048.

[12] H. He, E. A. Garcia, "Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*," vol. 21*,* no.9, 2009, pp. 1263-1284.

[13] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, , ... and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proceedings-software*, vol. 150, no. 3, pp. 161-175.

[14] M. Harman, "The relationship between search based software engineering and predictive modeling," In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ACM, 2010, p. 1.

[15] R. Malhotra, "Search based techniques for software fault prediction: current trends and future directions," In *Proceedings of the 7th International Workshop on Search-Based Software Testing,* ACM, 2014, pp. 35-36.

[16] D. Rodríguez, R. Ruiz, J. C. Riquelme and J. S. Aguilar–Ruiz, "Searching for rules to detect defective modules: A subgroup discovery approach, "*Information Sciences*, 2012, pp. 14-30.

[17] L. L. Minku and X. Yao, "Software effort estimation as a multiobjective learning problem," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 4, 2013, 35.

[18] L. Pelayo, S. Dick, "Applying novel resampling strategies to software defect prediction" *In IEEE 2007 Annual Meeting of the North American Fuzzy Information Processing Society*, NAFIPS 2017, pp. 69-72.

[19] M. J. Siers and M. Z. Islam, M. Z. "Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem," *Information Systems*, vol. 51, 2015, 62-71.

[20] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. *37,* no. 6, 2010, pp. 4537-4543.

[21] M. Tan, L. Tan, S. Dara and C. Mayeux, "Online defect prediction for imbalanced data," In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, 2015, pp. 99-108. IEEE.

[22] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data." *Neurocomputing*, vol. 343, pp. 120-140.

[23] K. Bashir, T. Li, C. W. Yohannese and M. Yahaya, SMOTEFRIS-INFFC: Handling the challenge of borderline and noisy examples in imbalanced learning for software defect prediction. *Journal of Intelligent & Fuzzy Systems*, vol. 38, no. 1, 2020, pp. 917-933.

[24] R. Malhotra, M. Khanna and R. R. Raje, "On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions," *Swarm and Evolutionary Computation*, vol. 32, 2017, 85-109.

[25] N. V. Chawla, K.W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique*," Journal of artificial intelligence research,* vol. 16, 2002, pp. 321-357.

[26] H. Han, W. Y. Wang and B. H. Mao, "Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning," In *International conference on intelligent computing*. Springer, Berlin, Heidelberg, August 2005, pp. 878-887.

[27] C. Bunkhumpornpat, K. Sinapiromsaran and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," *In Pacific-Asia conference on knowledge discovery and data mining*, Springer, Berlin, Heidelberg, April 2009, pp. 475-482.

[28] H. He, Y. Bai, E. A. Garcia and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," In *2008 IEEE*

International Joint Conference on Neural Networks, June 2008, pp. 1322-1328.

[29] L. Sánchez, I. Couso and J. A. Corrales, "Combining GP operators with SA search to evolve fuzzy rule based classifiers," *Information Sciences*, vol. 136 no.4, 2001,pp. 175-191.

[30] H. Ishibuchi, T. Nakashima and T. Murata, T. "Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 5, 1999, pp. 601-618.

[31] R. Malhotra, N. Pritam, K. Nagpal and P. Upmanyu, (2014, September). "Defect collection and reporting system for git based open source software," *In IEEE International Conference on Data Mining and Intelligent Computing*, ICDMIC September 2014, pp. 1-7.

[32] http://gromit.iiar.pwr.wroc.pl/p_inf/ ckjm/metric.html

[33] D. W. Zimmerman and B. D. Zumbo, "Relative power of the Wilcoxon test, the Friedman test, and repeated-measures ANOVA on ranks. *The Journal of Experimental Education*, vol. 62, no. 1, 1993, pp. 75-86.