Journal of Engineering Science and Technology
Vol. 14, No. 6 (2019) 3294 - 3308

SOFTWARE DEFECT PREDICTION: ANALYSIS OF CLASS IMBALANCE AND PERFORMANCE STABILITY

ABDULLATEEF O. BALOGUN*[,1,2,5], SHUIB BASRI[1,5], SAID J. ABDULKADIR[1,6], VICTOR E. ADEYEMO[3], ABDULLAHI A. IMAM[1,4,5], AMOS O. BAJEH[2]

[1]Department of Computer and Information Sciences, Universiti Teknologi Petronas, 32610 Seri Iskandar, Perak, Malaysia
[2]Department of Computer Science, University of Ilorin, Ilorin, Nigeria
[3]School of Computing and IT, Taylor's University, Subang Jaya, Selangor, Malaysia
[4]Department of Computer Science, Ahmadu Bello University, Zaria, Nigeria
[5]Software Quality and Quality Engineering (SQ2E) Research Cluster, Universiti Teknologi Petronas, 32610 Seri Iskandar, Perak, Malaysia
[6]Centre for Research in Data Science (CERDAS), Universiti Teknologi Petronas, 32610 Seri Iskandar, Perak, Malaysia
*Corresponding Author: abdullateef_16005851@utp.edu.my; bharlow058@gmail.com

Abstract

The performance of prediction models in software defect prediction depends on the quality of datasets used for training such models. Class imbalance is one of data quality problems that affect prediction models. This has drawn the attention of researchers and many approaches have been developed to address this issue. In this study, an extensive empirical study is presented, which evaluates the performance stability of prediction models in SDP. Ten software defect datasets from NASA and PROMISE repositories with varying imbalance ratio (IR) values were used as the original datasets. New datasets are generated from the original datasets using undersampling (Random under Sampling: RUS) and oversampling (Synthetic Minority Oversampling Technique: SMOTE) methods with different IR values. The sampling techniques were based on the equal proportion (100%) of the increment (SMOTE) of minority class label or decrement (RUS) of the majority class label until each dataset is balanced. IR is the ratio of the defective instances to non-defective instances in a dataset. Each newly generated datasets with different IR values based on different sampling techniques were randomized before applying prediction models. Nine standard prediction models were used on the newly generated datasets. The performance of the prediction models was measured using the Area Under Curve (AUC) and Co-efficient of Variation (CV) is used to determine the performance stability. Firstly, experimental results showed that class imbalance had a negative effect on the performance of prediction models and the oversampling method (SMOTE) enhanced the performances of prediction models. Secondly, Oversampling method of balancing datasets is better than using Undersampling methods as the latter had poor performance as a result of the random deletion of useful instances in the datasets. Finally, among the prediction models used in this study, it appeared that Logistic Regression (LR) (RUS: 30.05; SMOTE: 33.51), Naïve Bayes (NB) (RUS: 34.18; SMOTE: 33.05), and Random Forest (RF) (RUS: 29.24; SMOTE: 64.25) with their respective CV values are more stable prediction models and they work well with imbalanced datasets.

Keywords: Class imbalance, Data quality, Software defect prediction.

1. Introduction

The goal of every software company is to produce software with little or no defects. This is a big challenge as defects can be injected at any or every phase of a software development cycle. This will, in turn, increase the overhead cost and time in completing a software product with expected quality. Identifying and fixing defects is time and resource consuming, which makes it practically impossible to remove all defects but reducing the magnitude of defects to the lowest level is achievable [1, 2]. Standard practices and methods such as unit testing and code inspection are used to improve software quality and reliability.

These methods are regarded as Software Quality Assurance (SQA) activities. They are continuous processes within the software development lifecycle (SDLC) and periodically check the quality and reliability of the developed software. Nonetheless, software engineers must exercise caution in the allocation of resources during this phase. As a result of this, prioritization of these activities will judiciously allocate limited resources to modules with defects [1, 3].

Software Defect Prediction (SDP) is an approach used for identifying defect-prone software modules or components. It helps software engineers to optimally allocate limited resources to defective software modules or components in the testing or maintenance phases of SDLC [4, 5]. This will, in turn, helps to assess software quality and also monitor software quality assurance [6, 7]. SDP models make use of the information such as software source code complexity, developer's information, and development history to predict software modules or component that may be defective [8, 9]. This information is quantified using software metrics to determine the level of software quality and reliability.

SDP can be seen as a classification problem since it involves the training of models with historical data to identify defect-prone modules [10, 11]. Data used for training defect prediction models have a large influence on the performance of prediction models. These data are highly complex and skewed, which can be attributed to class imbalance problem [10, 12, 13]. An imbalance software defects data has an equal representation of its classes with the majority class as the non-defective instances while the minority class as the defective instances [14, 15]. Concerns have been raised on this issue from prior works that models trained with imbalanced data tend to produce inaccurate results as prediction models are usually biased by identifying mostly the majority class at the expense of the minority class [1, 3, 10, 12, 16, 17].

Many solutions have been developed to solve the class imbalance problem such as sampling, cost-sensitive and ensemble methods [15, 18, 19]. However, these solutions are not equally effective as most empirical studies do not take into consideration the impact of class imbalance on prediction models and which, imbalance method works well or help to learn capabilities in software defect prediction. Selecting models, which are stable and efficient with class imbalance will give a better result.

This study presents a method to empirically validate and evaluate the prediction performance stability of prediction models using sampling methods in addressing class imbalance problem. Undersampling and oversampling methods are applied to original imbalance software defect datasets to generate new datasets with varying imbalance ratio (IR). IR is the ratio of defective instances

to non-defective instances in a given dataset. Each newly generated dataset is further randomized for even distribution of class labels.

Thereafter, nine standard prediction models with different characteristics were used to classify the newly generated datasets. The Area Under Curve (AUC) was used to evaluate the predictive performance of each prediction models on the original and newly generated datasets. In addition, Co-efficient of Variation (CV) was used to measure the predictive performance stability of prediction models. This is to determine how prediction models behave with datasets with varying IR as a result of the class imbalance problem.

In summary, the main contributions of this study are as follows:

- A new method to empirically evaluate the predictive performance stability of prediction models using both undersampling and oversampling methods in addressing class imbalance problem.
- Evaluation of the impact of sampling (undersampling and oversampling) approaches as a re-balancing method on the predictive performance of prediction models in SDP.

The rest of the paper is structured as follows. Section 2 presents a review of related work in SDP and class imbalance problem. Section 3 presents the methodology of the study, and Section 4 presents a comprehensive experimental evaluation and discussion of results. Finally, Section 5 concludes the paper with some future research directions.

2. Literature Review

From existing studies, it can be seen that class imbalance greatly affects the performance of prediction models. Inaccurate predictions and interpretations are generated when prediction models are trained with imbalanced data sets. Many researchers have investigated and proposed class re-balancing techniques such as data sampling, cost-sensitive analysis and ensemble methods to deal with the class imbalance problem.

For example, Wang and Yao [19], carried out an empirical study on the performance of data sampling, cost-sensitive and ensemble method approaches for resolving the class imbalance problem. They indicated that ensemble methods performed best when compared to others.

Rodriguez et al. [20] in their study also gave a similar conclusion that ensemble methods deal with class imbalance better than data sampling and cost-sensitive approaches in software defect prediction. It is also to be noted that ensemble methods were not originally proposed to address class imbalance but due to their methodology of breaking a subset into small chunks for learning and their after aggregating their solution makes them work better on imbalanced datasets [21].

Recently, Yu et al. [22] carried out an empirical study on the stability of prediction models using undersampling approach. They constructed new datasets from the imbalanced data based on Imbalance Ratio (IR). From their experimental results, Random Forest (RF) and Naïve Bayes (NB) performance were stable with class imbalance, unlike the C4.5 decision tree, which was reported to be highly unstable with imbalance. However, they only considered the undersampling method, which is not the only sampling approach for balancing datasets.

Grbac et al. [23] also investigated the stability of prediction models with respect to different imbalance data level. They considered the stability of prediction models with feature selection techniques and their results indicated that data imbalance with high level makes feature selection unstable.

Wang et al. [24] looked into the stability of feature selection techniques and they concluded by highlighting some factors such as feature subset size, perturbation level and data imbalance that makes feature selection methods unstable.

Huda, et al. [25] proposed an ensemble method based on predictions models from multiple oversampling techniques. They aim to address the biases of using a single oversampling method in addressing the class imbalance problem.

Bennin, et al. [26] in their study proposed a new over-sampling method MAHAKIL to address the class imbalance problem. MAHAKIL was based on the biological theory of inheritance and it systematically generates new instance by finding the average of the Mahalanobis distance between two instances. The new instance will be based on the characteristics of its parent instances.

Goel, et al. [27] carried out an empirical study to evaluate the impact of SMOTE on the performance of prediction models in cross-project defect prediction (CPDP). Song, et al. [28] conducted a comprehensive investigation of the class imbalance problem in SDP.

Their study covered sixteen different types of imbalance methods with seven prediction models over twenty-seven defect datasets. From their results, they concluded that the occurrence of class imbalance has a negative effect on prediction models. In addition, they suggested that the right selection of imbalance method and prediction model can give good results.

Indeed, some studies have looked into the issue of prediction models performance stability with class imbalance. However, it is still an area that can still be explored more to guide software engineers, researchers, and further empirical studies in the selection of prediction models in SDP.

This study proposes to empirically evaluate the performance stability of prediction models considering both undersampling and oversampling methods in addressing the class imbalance problem and to evaluate the impact of these re-balancing methods on the performance of the prediction models in software defects prediction.

3. Methodology

In this paper, we adopted and extended the approach made by Yu et al. [22] as it a recent study, which addressed the class imbalance in SDP.

Suppose $X = \{x_1, x_2, \dots, x_n\}$ with n number of instances in dataset X and $\{a_1, a_2\}$ represents the number of defective and non-defective instances in dataset X respectively.

The Imbalance Ratio (IR) of X is given as:

$$IR = {}^{a_2}/_{a_1} \tag{1}$$

where $a_2 > a_1$ and $IR > 1$

Based on Fig. 1, with the original imbalance dataset X with a finite number of instances, new datasets are generated from original imbalanced by using data sampling methods. Random Under-Sampling (RUS) and Synthetic Minority Oversampling Technique (SMOTE) are used for the sampling technique respectively. The new dataset generation is based on the value of IR.

Unlike in the work of Yu et al. [22], this study considered both the undersampling and oversampling method in the generation of new datasets. For the SMOTE technique, an increment of 100% is used in the generation of new datasets and each generated dataset is randomized to avoid over-fitting.

While RUS is used for the undersampling technique in the generation of new datasets with respect to the IR value. The sampling techniques were based on the equal proportion (100%) of the increment (SMOTE) of minority class label or decrement (RUS) of the majority class label until each dataset is balanced.

Figure 2 gives a detailed algorithmic description of the experimental process used in this study.

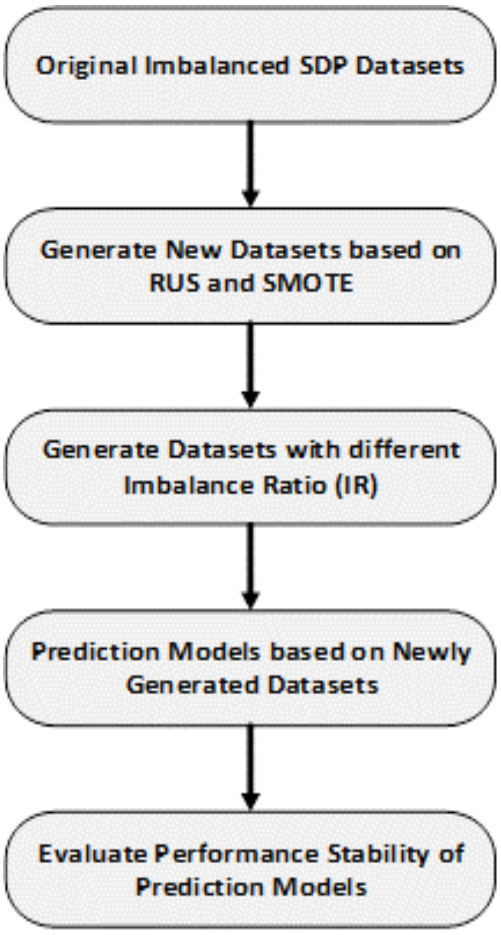

Fig. 1. Experimental framework.

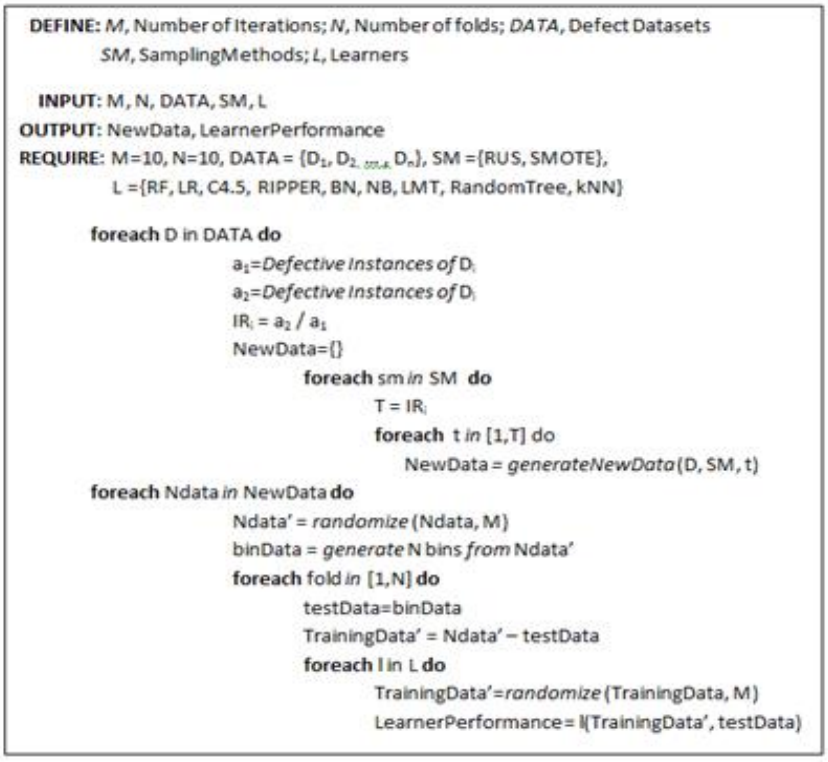

Fig. 2. Pseudocode for experimental process.

3.1. Prediction models

For the prediction models, nine models were selected, which are widely used for prediction processes in SDP. As shown in Table 1, four more prediction models (RIPPER, Bayesian Network, Random Tree, and Logistic Model Tree) were added to that of Yu et al. [22] since we are considering mostly models that have been used in empirical studies. In addition, these prediction models are picked based on different underlining classification characteristics with the aim of introducing heterogeneity.

Table 1. Prediction models.

Name	References
Decision Tree C4.5	[29-32]
Random Forest (RF)	[29, 30, 33, 34]
Random Tree (RT)	[35-37]
RIPPER	[36, 38]
Bayesian Network (BN)	[30, 39]
Naïve Bayes (NB)	[29-31, 33, 34]
K Nearest Neighbour (kNN)	[33, 40, 41]
Logistic Model Tree (LMT)	[36, 42]
Logistics Regression (LR)	[30, 31, 40]

3.2. Datasets

For empirical evaluation, 10 software defect datasets were selected from NASA and PROMISE repositories to conduct the experiment. These datasets have different levels of imbalance ratio, which makes them suitable for our study and they have been widely used by researchers for the same process. We used the cleaned version of NASA and Promise dataset provided by [35, 36, 43].

Detailed information on the selected datasets is in Table 2, which consists of the 10 datasets with 5 columns describing the name of the dataset, number of instances, number of defective instances, number of the non-defective instance and their respective imbalance ratio (the of defective instances to non-defective instances in the dataset).

Table 2. Software defect datasets.

Dataset	Number of samples	Number of defective samples	Number of non-defective samples	Imbalance ratio (IR)
CM1	327	42	285	6
JM1	7720	1612	6108	3
KC3	194	36	158	4
MW1	250	25	225	8
PC1	679	55	624	11
PC3	1053	130	923	7
PC4	1270	176	1094	6
ANT 1.5	292	32	260	8
JEDIT 4.2	367	48	319	6
TOMCAT	852	77	775	10

3.3. Performance metrics

The selection of evaluation metric is very important since it has already been proven that using some conventional measures such as accuracy rate often leads to an inaccurate interpretation of results due to the characteristic of the dataset (imbalance) [44, 45].

Based on this, Area Under Curve (AUC) is selected as our performance metric due to its wide usage [29-31, 33, 39] and proven to be more accurate and reliable [29]. AUC represents the Receiver Operating Characteristics (ROC). ROC is a measure of True Positive Rate and False Positive Rate [46].

For determining the performance stability of prediction models, Co-efficient of Variation (CV) was applied to the results of the prediction models. CV, which is the percentage ratio of standard deviation (SD) and average (AVE) is used to remove the effect of average difference on the comparison stability [16, 22]. The formula for CV is given as thus:

$$C.V = {}^{SD}/_{AVE} \times 100\% \quad (2)$$

Prediction models with high CV values are regarded to be unstable.

4. Results and Discussion

The experiments were carried out according to the experimental framework (Fig. 1) and experimental process (Fig. 2). A dataset is selected from Table 2 and its IR value is derived by finding the ratio of defective to non-defective instances in the dataset. New datasets are generated from the original dataset by applying the RUS and SMOTE algorithms with different IR values.

Each newly generated dataset is randomized before each prediction model from Table 1 is applied in to avoid overfitting in case of datasets generated by SMOTE technique.

All experiments are carried out based on 10-fold cross-validation, which is a standard approach [22, 47, 48]. In this study, we took into consideration the distribution difference between the original and the newly generated datasets.

Each experiment is repeated 10 times to ensure reliable results and the average AUC value is recorded for each IR value. According to Yu et al. [22], Mabayoje et al. [48] and Petric et al. [49], WEKA environment was used to conduct all experiments. Default parameters of prediction models as specified in WEKA were used in this study.

From the experimental results, we compared the performances of the prediction models on the normal and balanced datasets. Prediction models used on SMOTE generated datasets gave the best results when compared with the Normal and the RUS generated datasets.

As presented in Fig. 3, the RUS method depleted the performance of the prediction models due to the random removal of instances in the generation of its datasets.

That is, good instances have been removed by RUS in the line of balancing the datasets and hence, the poor performance of predictive models. This further strengthens the position that using the oversampling method for resolving class imbalance is better than undersampling method.

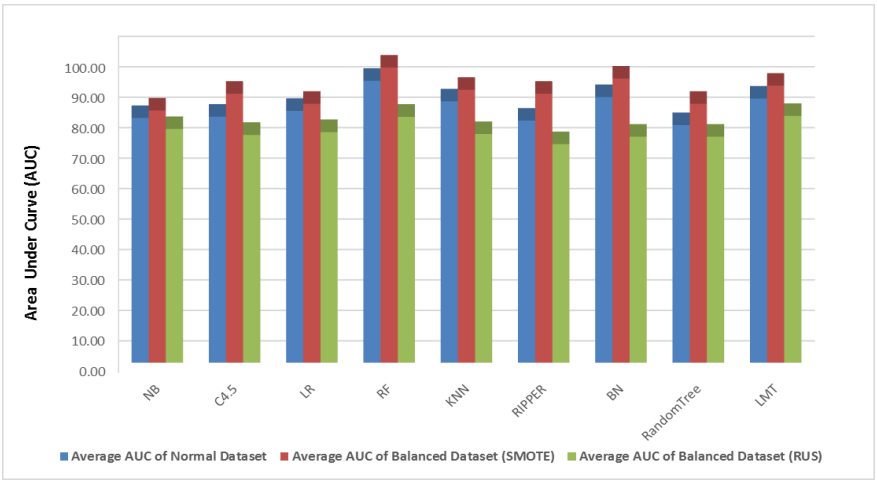

Fig. 3. Comparison of average AUC of prediction models on normal and balanced datasets (SMOTE and RUS).

From Figs. 4 and 5, Random Forest (RF) performed best in both cases (RUS and SMOTE generated datasets) compared with other prediction models. Bayesian Network (BN), Logistic Model Tree (LMT) and k Nearest Neighbour (kNN) performed well on both cases with no much difference while RandomTree algorithm performed worst averagely in both cases. As presented in Figs. 5 and 6, the respective performance of prediction models on balanced datasets was analyzed. Most of the prediction models had a good result with the SMOTE balanced datasets with Random Forest ranking best and Naïve Bayes coming last even with an average AUC of 82.84%.

In Figs. 6 and 7, the performance of the prediction models on RUS generated datasets is not as high as the SMOTE generated datasets. This is as a result of inherent information loss with random undersampling. Logistic Model Tree (LMT) and Random Forest (RF) performed well while Random Tree performed poorly.

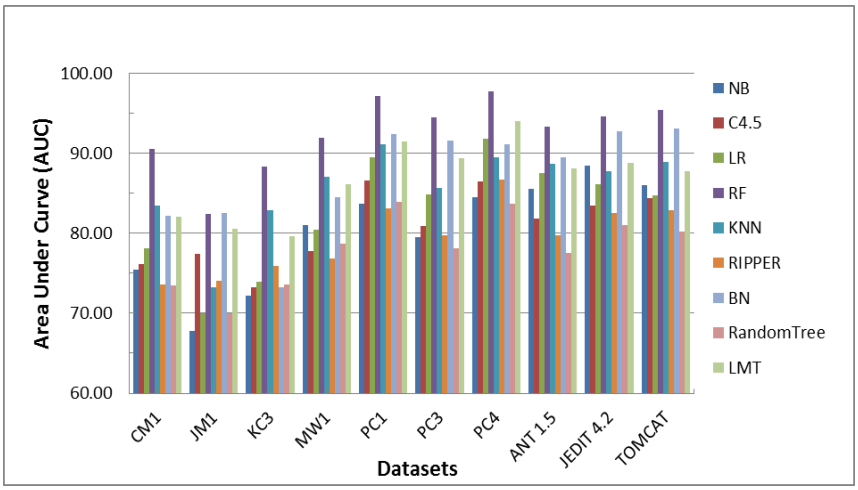

Fig. 4. The average performance of prediction models on SMOTE generated datasets based on different IR.

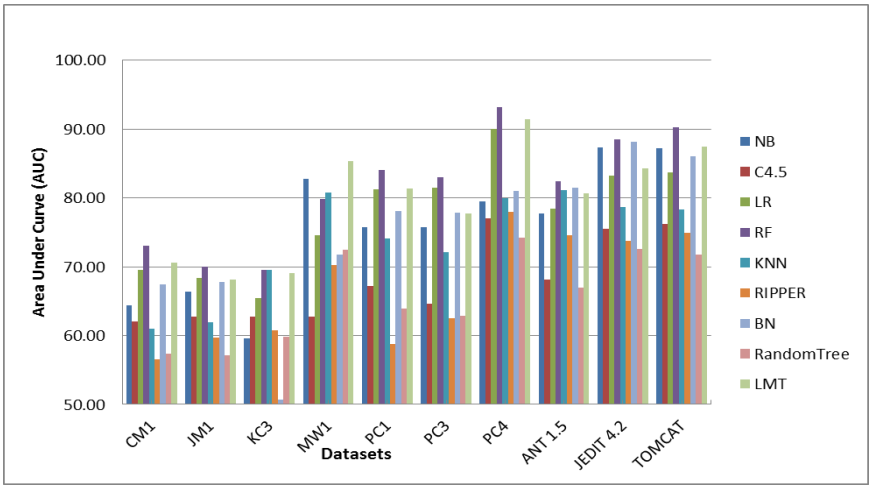

Fig. 5. Average performance of prediction models on RUS generated datasets based on different IR.

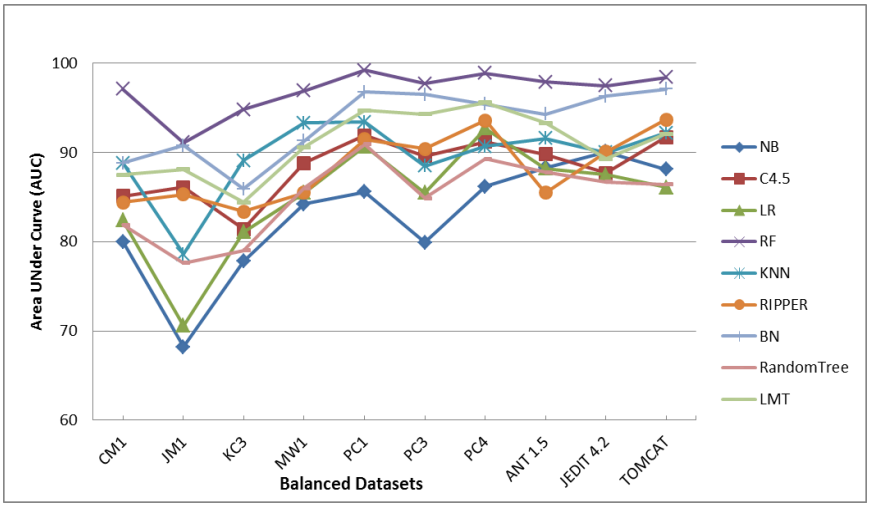

Fig. 6. Average performance of prediction models on balanced SMOTE generated datasets.

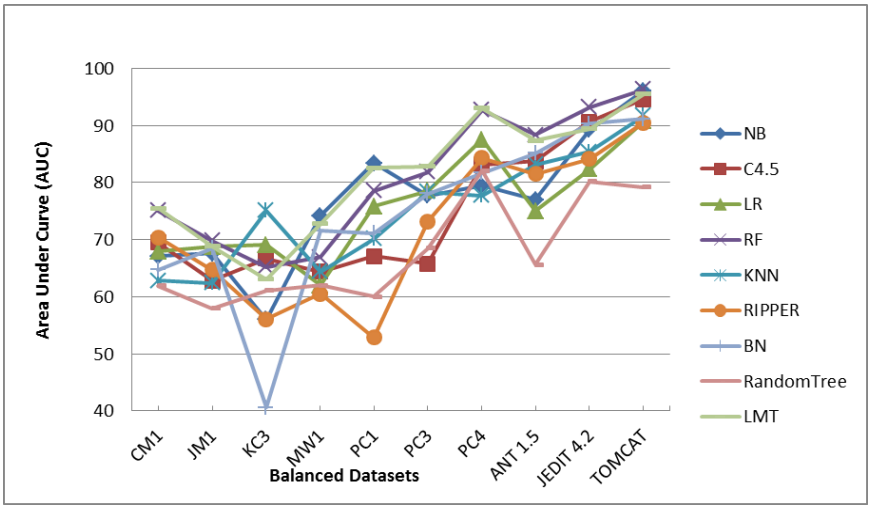

Fig. 7. Average performance of prediction models on balanced RUS generated datasets.

Tables 3 and 4 shows the CV values of prediction models on the generated datasets. Evidently, the CV values of all the prediction models are of high value and this is due to our consideration of distribution differences between the original and newly generated datasets.

Yu et al. [22] in their study also mentioned that considering distribution differences may affect the CV value of the prediction models. C4.5 had the highest CV values in both (RUS and SMOTE generated) sets of generated datasets, which makes it highly unstable for the class imbalance problem. LR (33.51) and NB (34.18) had the lowest CV values in SMOTE datasets while LR (30.05) and RF (29.24) had the lowest CV values in RUS generated datasets.

Table 3. CV value of prediction models on SMOTE generated datasets.

Datasets	NB	C4.5	LR	RF	KNN	RP	BN	RT	LMT
CM1	6.03	13.26	5.29	9.87	10.62	14.55	7.70	13.83	8.09
JM1	0.49	11.69	0.79	11.39	7.81	15.00	10.54	11.06	9.43
KC3	6.88	13.16	8.50	9.23	7.38	11.37	13.72	7.60	6.62
MW1	3.64	17.21	6.71	7.84	8.63	11.19	6.96	9.72	3.66
PC1	3.36	7.63	2.83	3.58	4.52	10.67	5.44	8.62	4.28
PC3	1.28	12.09	1.46	4.66	4.65	15.01	5.48	10.37	4.96
PC4	2.68	5.58	1.00	1.73	3.19	7.45	5.26	7.64	1.92
ANT 1.5	4.50	10.84	2.47	6.98	5.98	8.25	4.52	11.08	3.94
JEDIT 4.2	2.03	8.67	2.07	4.53	5.81	11.44	3.62	7.12	3.91
TOMCAT	3.30	7.73	2.40	4.44	4.33	12.53	5.10	9.82	4.93
TOTAL	**34.18**	107.87	**33.51**	64.25	62.93	117.46	68.35	96.86	51.75

Table 4. CV value of prediction models on RUS generated datasets.

Datasets	NB	C4.5	LR	RF	KNN	RP	BN	RT	LMT
CM1	6.10	8.70	3.09	3.56	6.36	12.91	4.01	6.47	1.50
JM1	2.53	0.92	1.52	0.63	1.46	5.86	1.90	1.07	1.55
KC3	3.07	12.51	3.59	5.26	5.29	5.58	15.55	3.92	5.45
MW1	1.28	16.21	5.10	3.15	5.26	5.06	2.17	5.34	2.56
PC1	4.23	5.44	2.97	2.72	3.53	6.56	3.18	4.07	1.75
PC3	2.75	4.46	1.93	1.00	5.07	12.28	1.22	5.96	10.83
PC4	0.59	4.68	1.17	0.41	1.88	6.55	0.48	5.00	1.62
ANT 1.5	3.21	16.56	4.16	4.72	4.83	7.90	2.20	5.31	3.74
JEDIT 4.2	2.25	12.54	1.33	3.60	6.40	11.38	1.45	8.30	3.42
TOMCAT	7.05	15.03	5.20	4.20	9.73	13.13	3.37	10.41	5.29
TOTAL	**33.05**	97.06	**30.05**	**29.24**	49.82	87.23	35.52	55.84	37.71

5. Conclusion

Data quality problem has always undermined prediction processes and software defect prediction is no exception. Datasets in SDP are highly skewed and this is a form of class imbalance. This paper empirically evaluates the performance stability of nine widely used prediction models in SDP based on undersampling (RUS) and oversampling (SMOTE) approaches. The results reveal that SDP datasets suffer class imbalance problem and it showed a negative impact on prediction models in SDP. Oversampling method (SMOTE) should be used in balancing such datasets. LR, NB, and RF are good prediction models and should be used for empirical studies as they are more stable in the presence of class imbalance than other models used in this study. This study only considers class imbalance as a data quality problem, other kinds of data quality problem can be looked into in further studies.

Nomenclatures

a_1	Defective instances
a_2	Non-defective instances
M	Number of iterations
N	Number of folds
X	Defect datasets

Abbreviations

AUC	Area Under Curve
BN	Bayesian Network
CV	Coefficient of Variation
IR	Imbalance Ratio
NASA	National Aeronautics and Space Administration
ROC	Receiver Operation Characteristics
RP	Ripper
RT	Random Forest
RUS	Random Under Sampling
SDLC	Software Development Life Cycle
SDP	Software Defect Prediction
SMOTE	Synthetic Minority Over-sampling TEchnique
SQA	Software Quality Assurance
WEKA	Waikato Environment for Knowledge Analysis

References

1. Hall, T.; Beecham, S.; Bowes, D.; Gray, D.; and Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6), 1276-1304.
2. Akintola, A.G.; Balogun, A.O.; Lafenwa-Balogun, F.B.; and Mojeed, H.A. (2018). Comparative analysis of selected heterogeneous classifiers for software defects prediction using filter-based feature selection methods. *FUOYE Journal of Engineering and Technology*, 3(1), 133-137.
3. Menzies, T.; Greenwald, J.; and Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1), 2-13.
4. Ali, M.M.; Huda, S.; Abawajy, J.; Alyahya, S.; Al-Dossari, H.; and Yearwood, J. (2017). A parallel framework for software defect detection and metric selection on cloud computing. *Cluster Computing*, 20(3), 2267-2281.
5. Yadav, H.B.; and Yadav, D.K. (2015). A fuzzy logic based approach for phase-wise software defects prediction using software metrics. *Information and Software Technology*, 63, 44-57.
6. Huda, S.; Alyahya, S.; Ali, M. M.; Ahmad, S.; Abawajy, J.; Al-Dossari, H.; and Yearwood, J. (2018). A framework for software defect prediction and metric selection. *IEEE Access*, 6, 2844-2858.
7. Li, Z.; Jing, X.-Y.; and Zhu, X. (2018). Progress on approaches to software defect prediction. *IET Software*, 12(3), 161-175.
8. Jing, X.-Y.; Wu, F.; Dong, X.; and Xu, B. (2017). An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Transactions on Software Engineering*, 43(4), 321-339.
9. Tantithamthavorn, C.; McIntosh, S.; Hassan, A.E.; and Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1), 1-18.
10. Sun, Z.; Song, Q.; Zhu, X.; Sun, H.; Xu, B.; and Zhou, Y. (2015). A novel ensemble method for classifying imbalanced data. *Pattern Recognition*, 48(5), 1623-1637.

11. Oluwagbemiga, B.A.; Shuib, B.; Abdulkadir, S.J.; and Sobri, A. (2019). A hybrid multi-filter wrapper feature selection method for software defect predictors. *International Journal of Supply Chain Management*, 8(2), 916-922.
12. Peng, L.; Zhang, H.; Yang, B.; and Chen, Y. (2014). A new approach for imbalanced data classification based on data gravitation. *Information Sciences*, 288, 347-373.
13. Al-Tashi, Q.; Rais, H.; and Jadid, S. (2018). Feature selection method based on grey wolf optimization for coronary artery disease classification. *Proceedings of the International Conference of Reliable Information and Communication Technology (IRICT)*. Kuala Lumpur, Malaysia, 257-266.
14. Forkman, J. (2009). Estimator and tests for common coefficients of variation in normal distributions. *Communications in Statistics-Theory and Methods*, 38(2), 233-251.
15. López, V.; Fernández, A.; García, S.; Palade, V.; and Herrera, F. (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250, 113-141.
16. Japkowicz, N.; and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5), 429-449.
17. Al-Tashi, Q.; Kadir, S.J.A.; Rais, H.M.; Mirjalili, S.; and Alhussian, H. (2019). Binary optimization using hybrid grey wolf optimization for feature selection. *IEEE Access*, 7, 39496-39508.
18. Arar, Ö.F.; and Ayan, K. (2015). Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, 33, 263-277.
19. Wang, S.; and Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62(2), 434-443.
20. Rodriguez, D.; Herraiz, I.; Harrison, R.; Dolado, J.; and Riquelme, J.C. (2014). Preliminary comparison of techniques for dealing with imbalance in software defect prediction. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. London, England, 10 pages.
21. Laradji, I.H.; Alshayeb, M.; and Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, 58, 388-402.
22. Yu, Q.; Jiang, S.; and Zhang, Y. (2017). The performance stability of defect prediction models with class imbalance: An empirical study. *IEICE Transactions on Information and Systems*, 100(2), 265-272.
23. Grbac, T.G.; Mausa, G.; and Basic, B.D. (2013). Stability of Software defect prediction in relation to levels of data imbalance. *Proceedings of the Second Workshop on Software Quality Analysis, Monitoring, Improvement and Applications (SQAMIA)*. Novi Sad, Serbia, 10 pages.
24. Wang, H.; Khoshgoftaar, T.M.; and Napolitano, A. (2012). An empirical study on the stability of feature selection for imbalanced software engineering data. *Proceedings of the 11th International Conference on Machine Learning and Applications (ICMLA)*. Boca Raton, Florida, United States of America, 317-323.

25. Huda, S.; Liu, K.; Abdelrazek, M.; Ibrahim, A.; Alyahya, S.; Al-Dossari, H.; and Ahmad, S. (2018). An ensemble oversampling model for class imbalance problem in software defect prediction. *IEEE Access*, 6, 24184-24195.
26. Bennin, K.E.; Keung, J.; Phannachitta, P.; Monden, A.; and Mensah, S. (2018). Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering*, 44(6), 534-550.
27. Goel, L.; Sharma, M.; Khatri, S.K.; and Damodaran, D. (2018). Implementation of data sampling in class imbalance learning for cross project defect prediction: An empirical study. *Proceedings of the Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT*). Amman, Jordan, 1-6.
28. Song, Q.; Guo, Y.; and Shepperd, M. (2018). A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, 45(12), 1253-1269.
29. Catal, C.; and Diri, B. (2009). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences-Informatics and Computer Science, Intelligent Systems, Applications*, 179(8), 1040-1058.
30. He, P.; Li, B.; Liu, X.; Chen, J.; and Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, 59, 170-190.
31. He, Z.; Shu, F.; Yang, Y.; Li, M.; and Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, 19(2), 167-199.
32. Mabayoje, M.A.; Balogun, A.O.; Bajeh, A.O.; and Musa, B.A. (2018). Software defect prediction: Effect of feature selection and ensemble methods. *FUW Trends in Science & Technology Journal*, 3(2), 518-522.
33. Gao, K.; Khoshgoftaar, T.M.; Wang, H.; and Seliya, N. (2011). Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software-Practice and Experience*, 41(5), 579-606.
34. Khoshgoftaar, T.M.; Gao, K.; Napolitano, A.; and Wald, R. (2014). A comparative study of iterative and non-iterative feature selection techniques for software defect prediction. *Information Systems Frontiers*, 16(5), 801-822.
35. Kalai Magal, R.; Jacob, S.G. (2015). Improved random forest algorithm for software defect prediction through data mining techniques. *International Journal of Computer Applications*, 117(23), 18-22.
36. Tantithamthavorn, C.; McIntosh, S.; Hassan, A.E.; and Matsumoto, K. (2018). The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, 45(7), 683-711.
37. Jimoh, R.G.; Balogun, A.O.; Bajeh, A.O.; and Ajayi, S. (2018). A promethee based evaluation of software defect predictors. *Journal of Computer Science and its Application*, 25(1), 106-119.
38. Cohen, W.W. (1995). Fast effective rule induction. *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, Tahoe City, California, United States of America, 115-123.

39. Okutan, A.; and Yıldız, O.T. (2014). Software defect prediction using Bayesian networks. *Empirical Software Engineering*, 19(1), 154-181.
40. Li, L.; and Leung, H. (2011). Mining static code metrics for a robust prediction of software defect-proneness. *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*. Banff, Alberta, Canada, 207-214.
41. Balogun, A.O.; Bajeh, A.O.; Orie, V.A.; and Yusuf-Asaju, W.A. (2018). Software defect prediction using ensemble learning: An ANP based evaluation method. *FUOYE Journal of Engineering and Technology*, 3(2), 50-55.
42. Ratzinger, J.; Sigmund, T.; and Gall, H.C. (2008). On the relation of refactorings and software defect prediction. *Proceedings of the International Working Conference on Mining Software Repositories.* Leipzig, Germany, 35-38.
43. Kakkar, M.; and Jain, S. (2016). Feature selection in software defect prediction: A comparative study. *Proceedings of the 6th International Conference on Cloud System and Big Data Engineering* (Confluence). Noida, India, 658-663.
44. He, H.; and Garcia, E.A. (2008). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263-1284.
45. Shepperd, M.; Song, Q.; Sun, Z.; and Mair, C. (2013). Data quality: Some comments on the NASA software defect datasets. *IEEE Transactions on Software Engineering*, 39(9), 1208-1215.
46. Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861-874.
47. Abdulkadir, S.J.; and Yong, S.-P. (2015). Scaled UKF–NARX hybrid model for multi-step-ahead forecasting of chaotic time series data. *Soft Computing*, 19(12), 3479-3496.
48. Mabayoje, M.A.; Balogun, A.O.; Bello, M.S.; Atoyebi, J.O.; Mojeed, H.A.; and Ekundayo, A. (2019). Wrapper feature selection based heterogeneous classifiers for software defect prediction. *Adeleke University Journal of Engineering and Technology*, 2(1), 1-11.
49. Petrić, J.; Bowes, D.; Hall, T.; Christianson, B.; and Baddoo, N. (2016). Building an ensemble for software defect prediction based on diversity selection. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Ciudad Real, Spain, 1-10.