International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6S3, September 2019

Parameter Tuning of a Sampling Technique for Change Prediction

Ankita Bansal, Abha Jain

Abstract: ***Change prediction is very essential for producing good quality software. It leads to saving of lots of resources in terms of money, manpower and time. Predicting the classes during early phases can be done with the help of model construction using machine learning techniques. Every technique requires approximately equal distribution of classes (balanced data) for an efficient prediction. In this study, we have used a sampling approach to balance the data. We observed the improvement in accuracy after the models are trained on the balanced data. To further improve the accuracy of the models, the default parameters of the sampling approach have been adjusted /tuned. The results show the improvement in accuracy after sampling and parameter tuning.***

Index Terms—Class Imbalance, Machine Learning, Metrics, Quality, Resampling, Sampling, Tuning.

I. INTRODUCTION

The sizes of software are increasing at an exponential rate. Almost every software consists of thousands of classes. At the same time, there is scarcity of resources like time, money , manpower etc. Thus, it is not possible to pay equal amount of attention to each class, resulting in a poor quality software. Poor quality software in-turn leads to unsatisfied customer and bad reputation in the market. Thus, in this study, the broad objective is to improve the quality of the software. This is done by predicting in advance, i.e. during early phases of Software Development Life Cycle (SDLC), some classes which need focused attention and dedicated resources. The selection of the classes is done by predicting the classes which have higher probability of being changed in the later phases of SDLC as compared to the other classes. Incorporating a change during the later phases leads to wastage of effort and resources spent during the earlier phases as the classes need to re-designed and all the further steps would be carried out again. Thus, we try to avoid such situation by identifying change prone classes during the design phase and then focusing our resources on them. Once such classes are identified, strict testing and verification activities can be employed on them. Moreover, the design of such classes can also be altered.

For the prediction of change prone classes, various object oriented metrics are taken as the internal attributes and models are constructed using Machine Learning (ML) techniques. In this study, we have used J48, Random Forest, Bagging, Logitboost and Adaboost to predict models. However, the performance of the models may suffer if the data is not balanced. The balanced data is the one in which the number of classes belonging to each category (change prone and not change prone) are almost equal. Software engineering data generally suffers from the class imbalance problem, i.e. having large variation in the number of change and not change prone classes. In this study, we have used a sampling technique, 'Resample with Replacement' to balance the dataset. We have compared the performance of the models on the imbalanced and the balanced data and observed improved classification performance on the balanced data (i.e. obtained after sampling). In addition, parameter tuning is also done to further improve the performance of the models. Parameter optimization is one of the known methods to improve the classification performance of the models. In this study, we have tuned the parameters of the sampling approach to bring out better results. The results show the further improvement in the performance of the ML models after the parameters have been tuned. The methodology followed is also shown diagrammatically in figure 1. For the purpose of validation, two versions of an open source software have been analyzed. The models are evaluated using area under the Receiver Operating Characteristic Curve.

The rest of the paper is organized as: Section II presents the related work. This is followed by section III which presents the basic concepts the research carried out in this work. Section IV gives a brief overview of the benefits of sampling the dataset and how tuning the parameters of the sampling technique can further increase the performance of the ML techniques. Finally the results are presented in section V.

Manuscript published on 30 September 2019.
***** Correspondence Author (s)
AnkitaBansal, Department of Information Technology, NetajiSubhas University of Technology, Delhi, India. (Email: ankita.bansal06@gmail.com)
Abha Jain, Department of Computer Science, ShaheedRajguru College of Applied Sciences for Women, University of Delhi, Delhi, India. (Email: me_abha@yahoo.com)

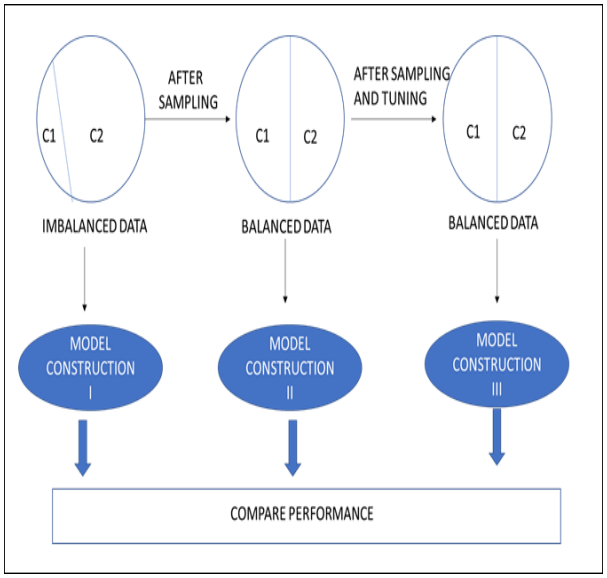

Figure 1: Basic Methodology of the Work

II. RELATED WORK

In this section, we briefly state the work done in this field. There have been studies in literature showing the use of metrics to predict change prone classes [1-4]. The study [3] considered evolution metrics for change prediction. Instead of using metrics, the studies [5-9] have used other measures for prediction of change prone classes. For example, Han 08, 10 calculated the value of Behavioral Dependency Measure (BDM) to predict change proneness in UML models. Developer related factors are also considered for change prediction by few recent studies [8,10]. The work has also been done in the field of cross project change prediction where different projects are used for training and validation to yield more generalized results [11-13].

However, the studies discussed have not taken into account the class imbalance problem. There are various different subject areas such as fraud text mining [14], fraud detection [15], analyzing sentiments [16], bioinformatics applications [17], video mining [18] which have dealt with the class imbalance problem. In the area of software engineering, class imbalance problem has been addressed for fault prediction by few studies [19-22]. The related field of fault prediction is change prediction which is being discussed in this study. The studies [23,24] have considered class imbalance problem and have proved that the performance of classification models improve when the data is balanced.

To improve the performance of classification models, parameter optimization, i.e. using tuned parameters instead of the default parameters for the purpose of classification can also be employed. There have been studies in literature which have worked on various techniques to tune the parameter of search based or evolutionary algorithms, their optimization and the selection of appropriate parameters [25-28]. However, the literature showing the tuning of sampling approaches is a nascent field. Moreover, there are studies [29,30] which have shown that the classifiers on the balanced data give the comparable performance as the classifiers on the imbalanced data. Thus, more empirical research is required in this field to generalize the results. Hence, in this study, we have balanced the data using a sampling approach and also tuned the default parameter settings of the sampling approach to bring out further improvement in the performance of the classifiers.

III. BASICS OF THE RESEARCH

This section highlights the variables used in this study to predict the model. Also the dataset used to validate the results has been discussed.

A. Variables Used

The independent variables used in this study are various object oriented metrics. Among a number of metric suites proposed in literature, we have used a famous metric suite proposed by Chidamber and Kemerer [31]. It consists of 6 metrics (WMC, NOC, DIT, LCOM, CBO, RFC) which measure different concepts of object oriented paradigm such as coupling, cohesion, inheritance. In addition to Chidamber and Kemerer metric suite, a metric commonly used to measure the size of the software known as Lines of Code (LOC) is also used.

The dependent variable used in this study measures the number of changes in terms of number of lines of code added, deleted and modified in the later release of the software when compared with the former release. In this study, it is termed as change proneness, which measures the probability of change in the future release of the software. It is considered to be binary in this study, denoted by a 1 for a change prone class and a 0 for a not change prone class.

B. Data for Validation

For the purpose of empirical validation, we have used an open source, Apache software known as MyFaces. It is written in Java and is platform independent. MyFaces is basically a web framework which is used to support different web applications which includes services of web, resources of web and APIs. “Apache MyFaces is an Apache Software Foundation project that creates and maintains an open-source Java Server Faces implementation, along with several libraries of JSF components that can be deployed on the core implementation” [https://en.wikipedia.org/wiki/Apache_MyFaces]. We have analysed two latest releases of Myfaces, MyFaces Core 2.2.12and MyFaces Core 2.3.3. For analysing these releases and collecting data between the two releases, a tool was developed by one of the authors, known as CRG (Change Report Generator). The details of the tool can be obtained from [32]. The tool takes the complete path address of both the releases and gives the values of independent and the dependent variables.

After the collection of data between MyFaces Core 2.2.12and MyFaces Core 2.3.3 release (referred as MyFaces 2.2.12 in the study), the following insights are obtained:

- The total number of common classes between the two releases is 1378
- The number of change prone classes is 536
- The number of not change prone classes is 842

IV. SAMPLING AND TUNING

The model prediction consists of two important phases: training and testing. Training is a process of constructing a model using the data with the known dependent variable, i.e. the training data. Different classification algorithms are available each of which has a different learning algorithm. Depending on the classifier selected, appropriate algorithm is applied to construct or build the model. Once the model is built, the second phase is carried out where testing of the model is performed to determine its accuracy. For a good classification model, the training data should be balanced. In other words, the training data should have approximately the equal number of both the classes (for binary problems). In this study, the two classes are 'change prone' (denoted by 1) and 'not change prone' (denoted by 0). However, it is observed that software engineering data suffers from the class imbalance problem, where there is huge variation in the number of both the types of classes. There are various approaches proposed in literature such as ensemble methods, cost sensitive algorithms, kernel based method, Mahalanobis-Taguchi method (MTS) etc. to rectify the class imbalance problem.The literature shows that the performance of the classifiers improve after the data is balanced. However, few studies also concluded that the performance of the models remained either the same or even decreased in their studies after the data is balanced [29,30]. Hence, more evidence is required in the concerned field. Thus, in this study, we have used a sampling approach known as 'Resample with Replacement' to balance the data. All the classifiers are validated on the imbalanced (i.e. before sampling) as well as the balanced data (i.e. after sampling) so that a comparison can be drawn. Resample with Replacement is based on the concept of oversampling where instances are added to the minority class to equalize both the classes. It is a type of bootstrapping mechanism where subsamples are made with replacement and the accuracy is calculated for each subsample. This process continues for many such subsamples. Finally, the subsample giving the maximum accuracy is used.

As discussed, there are a number of classification algorithms which can be used for model prediction. Every algorithm has a set of variables known as parameters, which are used in the algorithm. Each of these parameters is given a pre-defined value which is called as a default value. This default value may not give good result on all the data. The type and the nature of the data may differ according to the problem statement and hence, the values of the parameters should also change. We adjust the parameters according to the circumstances to bring out the improved accuracy of the model. The method of adjusting default parameter settings is called as tuning of parameters. In other words, the process of optimization of parameters in order to obtain the best results is called as tuning an algorithm or tuning the parameters of an algorithm. In this study, instead of tuning the parameters of the classification algorithms, we have tuned the default parameter settings of the sampling approach used.

Table 1 shows the parameters for the sampling technique used in this paper i.e.'re-sampling'. The table depicts the default values of the parameters and the changed values obtained after tuning the parameters.

Table 1.Parameters for the Sampling Technique

Parameters	Default Value	Changed Value
biasToUniformClass()	0	1
randomSeed()	1	4

Following are the two parameters for the re-sampling technique whose values have been tuned:

- **Biastouniformclass():** This parameter represents thebias factor towards uniform class distribution. The default value is 0 and 1 specifies the uniform distribution.
- **RandomSeed():** This parameter specifies the value of the seed forthe random number generator. The value of seed determines whether the previous calculation needs to be repeated. The value is set to the same number both the times if a previous calculation needs to be exactly repeated. But, if each time different calculation is required, then the value of the seed is set to different numbers each time.

V. RESULT ANALYSIS

This section depicts how the performance of different ML techniques viz. J48, Random Forest, Bagging, Logitboost, Adaboost improves significantly in predicting the change-prone classes of the dataset which has been sampled in comparison to the same un-sampled dataset. The performance of these techniques further shows a significant improvement when the parameters of the sampling technique have been tuned.

Table 2 depicts the performance of these techniques in terms of their AUC values against the un-sampled and sampled dataset. As can be seen from the table, performance of almost all ML techniques has been improved when sampling technique has been applied on the dataset. An overall best performance on the un-sampled dataset has been depicted by J48 technique with AUC value being 0.87. However, the performance of this technique remains consistent even on the sampled dataset. But there is a slight improvement in the performance of the technique (J48) with AUC value increased to 0.89 when sampling with tuning the parameters has been done. On the contrast, Adaboost technique shows a drastic improvement in its performance with the value of AUC increased from 0.76 on the un-sampled dataset to 0.90 on the sampled dataset. However, its performance shows a huge decline with AUC value falling down to 0.77 when the parameters of the sampling technique have been tuned. An overall best performance on the sampled dataset has been depicted by Random Forest technique with the value of AUC being as high as 0.96. This technique has performed consistently well even on the sampled data that has been obtained by tuning the parameters of the sampling technique. The results are also represented in the bar graph a (figure 2) for better visuality.

Table 2. Results (AUC values) of ML techniques

ML techniques	Without sampling	With sampling	Sampling with tuning parameters
J48	0.87	0.87	0.89

Random Forest	0.84	0.96	0.96
Bagging	0.79	0.77	0.91
Logitboost	0.77	0.76	0.79
Adaboost	0.76	0.90	0.77

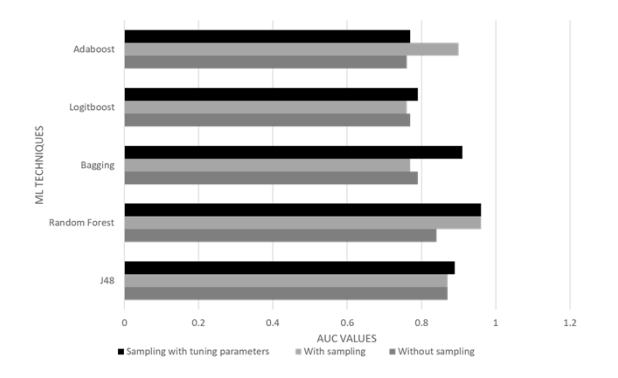

Figure 2: Graphical Comparison of the Performance

VI. CONCLUSION

The timely identification of the change prone classes of the software under development is highly beneficial in today's scenario. As there is shortage of resources in terms of time, money and manpower, it is not possible to pay equal amount of attention of all the changes in the software. As a result, we have incorporated different ML techniques in order to predict the change prone classes in early phases of SDLC. The performance of these techniques was evaluated and compared using AUC values. Further, to improve the performance of the ML techniques sampling was performed on the dataset leading to equal distribution of classes belonging to each category (change prone and not change prone). In addition, parameter tuning is also done to further improve the performance of the models so predicted. The results indicated that the best performance has been depicted by Random Forest technique which showed a huge increase in the value of AUC when the data was sampled from an un-sampled dataset.

REFERENCES

1. Eski S, Buzluca F (2011) An empirical study on object-oriented metrics and software evolution in order to reduce testing cost by predicting change prone classes. In Proc. of International Conference on Software Testing, Verification and Validation Workshop, 566–571.
2. Lu H, Zhou Y, Xu B, Leung H, Chen L (2012) The ability of object-oriented metrics to predict change-proneness: a meta-analysis. Empir Softw Eng J 17(3):200–242.
3. Elish MO, Al-Khiaty MA (2013) A suite of metrics for quantifying historical changes to predict future change prone classes in object-oriented software. J Softw: Evol Process 25(5):407–437.
4. Malhotra R, Khanna M (2013) Investigation of relationship between object-oriented metrics and change proneness. Int J Mach Learn Cybern. Springer-Verlag 4(4): 273–286.
5. Han AR, Jeon SU, Bae DH, Hong JE (2008) Behavioral dependency measurement for change- proneness prediction in UML 2.0 design models. In: Proceedings of the 32nd IEEE International Computer Software and Applications Conference, 76–83.
6. Han AR, Jeon SU, Bae DH, Hong JE (2010) Measuring behavioral dependency for improving change-proneness prediction in UML-based design models. The Journal of Systems and Software, 83(2), 222–234.
7. Sharafat AR, Tahvildari L (2008) Change prediction in object oriented software systems: a probabilistic approach. J Softw, 3(5), 26–39.
8. Catolino G, Palomba F, Lucia AD, Ferrucci F, Zaidman A (2017) Developer-related factors in change prediction: an empirical assessment. In Proc. of the 25th International Conference on Program Comprehension, Argentina.
9. Arvanitou EM, Ampatzoglou A, Chatziogeorgiou A, Avgeriou P (2017) A Method for Assessing Class Change Proneness, Evaluation and Assessment in Software Engineering. ACM, Sweden.
10. Catolino G, Palomba F, Lucia AD, Ferrucc F, Zaidman A (2018) Enhancing change prediction models using developer-related factor. Journal of Systems and Software Volume 143, 14-28.
11. Xia X, Lo D, McIntosh S, Shihab E, Hassan AE (2015) Cross-project build co-change prediction. SANER, 311-320.
12. Kumar L (2017) Transfer Learning for Cross-Project Change-Proneness Prediction in Object-Oriented Software Systems: A Feasibility Analysis. ACM SIGSOFT Software Engineering Notes, 42(1), 1-11.
13. Bansal A, Jajoria S (2019) Cross-Project Change Prediction Using Meta-Heuristic Techniques. International Journal of Applied Metaheuristic Computing, 10(1).
14. Munkhdalai T, Namsrai OE, Ryu KH (2015) Self-training in significance space of support vectors for imbalanced biomedical event data, BMC Bioinf., 16(7), 1-2.
15. Phua C, Alahakoon D, Lee V (2004) Minority report in fraud detection: classification of skewed data. SIGKDD Explorations, 6(1), 50–59.
16. Xu R, Chen T, Xia Y, Lu Q, Liu B, Wang X (2015) Word embedding composition for data imbalances in sentiment and emotion classification. Cogn. Comput.,7(2), 226–240.
17. Yang P, Yoo PD, Fernando J, Zhou BB, Zhang Z, Zomaya AY (2014) Sample subset optimization techniques for imbalanced and ensemble learning problems in bioinformatics applications. IEEE Trans. Cybern, 44(3), 445–455.
18. Apandi ZFM, Mustapha N, Affendey LS (2011) Evaluating integrated weight linear method to class imbalanced learning in video data. In 3rd Conference on Data Mining and Optimization, 243–247.
19. Liu Y, An A, Huang X (2006) Boosting prediction accuracy on imbalanced datasets with SVM ensembles. In Advances in Knowledge Discovery and Data Mining, 107–118.
20. Shatnawi R (2012) Improving software fault-prediction for imbalanced data. In Proc. of International Conf. on Innovations in Information Technology, 54–59 .
21. Wang S, Yao X (2013) Using class imbalance learning for software defect prediction. IEEE Trans Reliab 62:434–443 .
22. Seiffert C, Khoshgoftaar TM, Hulse JV, Folleco A (2014) An empirical study of the classification performance of learners on imbalanced and noisy software quality data. Inf Sci 259:571–595.
23. Tan M, Tan L, Dara S, Mayeux C (2015) Online defect prediction for imbalanced data. In Proc. of 37th International Conf. on Software Engineering.
24. Malhotra R, Khanna M An empirical study for software change prediction using imbalanced data. Empir Software Eng.
25. Bartz-Beielstein T, Markon S (2004) Tuning search algorithms for real-world applications: A regression tree based approach. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1111–1118 .

26. Conrad A, Roos R, Kapfhammer G (2010) Empirically studying the role of selection operators during search-based test suite prioritization. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 1373–1380. ACM .
27. Jong KD (2007) Parameter setting in EAs: a 30 year perspective. Parameter Setting in Evolutionary Algorithms, pp. 1–18.
28. Eiben A, Michalewicz Z, Schoenauer M, Smith J (2007) Parameter control in evolutionary algorithms. Parameter Setting in Evolutionary Algorithms pp. 19–46 .
29. Batista GEAPA, Prati RC, Monard MC (2004) A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. ACM SIGKDD Explorations Newsletter, vol. 6, no. 1, pp. 20-29.
30. Japkowicz N, Stephen S (2002) The Class Imbalance Problem: A Systematic Study. Intelligent Data Analysis, vol. 6, no. 5, pp. 429-449.
31. Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Tran Softw Eng 20(6):476–493.
32. Malhotra R, Bansal A, Jajoria S (2016) An automated tool for generating change report from open-source software. Advances in Computing, Communications and Informatics (ICACCI), International Conference IEEE, pp. 1576-1582.