International Journal of Engineering & Technology, 7 (4.15) (2018) 39-42

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Random Forest and Novel Under-Sampling Strategy for Data Imbalance in Software Defect Prediction

Utomo Pujianto

Universitas Negeri Malang, Indonesia
**Corresponding author E-mail: utomo.pujianto.ft@um.ac.id*

Abstract

Data imbalance is one among characteristics of software quality data sets that can have a negative effect on the performance of software defect prediction models. This study proposed an alternative to random under-sampling strategy by using only a subset of non-defective data which have been calculated as having biggest distance value to the centroid of defective data. Combined with random forest classification, the proposed method outperformed both the random under-sampling and non-sampling method on the basis of accuracy, AUC, f-measure, and true positive rate performance measures.

Keywords: *Data imbalance; Random forests; Software defect prediction; Under-sampling.*

1. Introduction

Building good software means building software with minimum defects. Unfortunately, the cost of finding and fixing defects is considered to be one of the highest among software engineering activities [1]. As the cost of software defect increases over the software development step [2], the growing size and complexity of software today make software testing and reviews become an important phase of the software engineering process.
Identification of software defects can be done in several ways. Two of them are software reviews and software defect predictions. Software testing and reviews are much less cost-effective as compared to software defect prediction approaches to identify software defects. Recent research has shown that the use of software defect prediction approaches has a greater probability of defect identification compared to software review techniques that have been commonly used in software industry [3]. Therefore, good defect prediction performance can help minimize costs, guide the testing effort effectively, and upgrade the testing process by simply concentrating on flawed software components [4], thereby ultimately improving overall software quality [5]. It is for that reason these days software defect prediction become a significant research topic in the domain of software development [6].
Software measurement data collected during software construction such as metrics from source code, execution logs, code change records can be used as a source of valuable information. Software developers can use data mining techniques to process the data so as to generate knowledge that enhances project management and production of quality software.
This study focused on using software metrics especially static code analysis with defect logs to build models of software defect prediction that classify software modules into two classes, defective and non-defective. With such information, software engineers can have a better priority list on the distribution of constrained project resources and concentrate only on evaluating parts of the program which are likely to have a higher number of defects.
In recent years, there has been a lot of research done to improve the performance of software defect prediction methods [7]. One of the facts obtained from these studies is the performance of these methods is influenced by the quality of data and selection of learning algorithms. We are more interested in discussing the quality of data in this study.
Data imbalance is one of the challenges faced by researchers in predicting software defects. That is, the number of software modules identified to have defects is just a small fraction compared to the number of non-defective modules in the data set.
The data imbalance problem in software defect prediction happens when non-defective modules in a data set significantly outnumber the defective ones. There are two types of advance issues can occur. The first issue is misclassification of non-defective modules as defective ones. The second issue refers to defective modules misclassified as non-defective one. Class imbalance usually results in more prediction errors on the later type of problem [8].
In software defect prediction, the latter issue is usually far more expensive than the previous one. It is because the cost of the first issue may only result in wasted effort evaluating program modules that already have high quality, while the latter issue may result in a loss of opportunity to correct a defective module before software distribution. Because the cost of fixing defective modules after software distribution is more expensive than when it fixed right after the testing phase, it is agreed that the latter problem of data imbalanced is considered to be producing higher cost.
There are three common approaches for dealing with unbalanced datasets. For example, approaches at the data level, algorithmic levels, and ensemble methods [9]. Data-level approaches include various resampling and data synthesis techniques to improve the inclination of the distribution of the training data class. At the algorithmic level, the main method is to adjust the operation of the existing algorithm to make the classifier to be more conducive to minority class classification. The algorithm and ensemble approach have the same objective, which is to improve the classification algorithm without altering the data so that there can be two approaches namely the data level approach and the algorithm level approach.

By dividing the solutions into two approaches can simplify the focus of improvement objects, the data level approach is focused on initial processing of data, while the algorithm level approach is focused on improving the algorithm or incorporating (ensemble).
In this study, we propose modifications to random under-sampling techniques to address the negative effects of data imbalances in order to achieve a balance of distributions of the majority and minority data classes. Our primary goal is to evaluate the impact of the data sampling approach when combined with Random Forest classification models built with imbalanced data. We generate three different subsets in order to apply defect prediction on the acquired dataset. We conduct three different combinations of sampling and prediction model. We then compare the performances of the selected prediction models built over the three-sampling mechanism of the sampled datasets.
The rest of the paper is organized as follows. Section 2 provides more detailed information about the Random Forest model, our proposed strategy for data sampling, and the performance metrics used to evaluate prediction models we use. Section 3 contains the description of the datasets used in the experiments. Section 4 presents the experimental results. Finally, we summarized the conclusion within Section 5.

2. Methodology

2.1. Data Sampling Techniques

Our experiments were carried out on ten datasets from National Aeronautics and Space Administration (NASA). The distributions between the defective and non-defective class data are significantly skewed. In nine of the ten data sets, between 2% and 28% of the instances were a member of the defective class while in the MC2 dataset the defective class data is 35%. There are two data sampling techniques used in this study. After each sampling, the new dataset contains each 50% non-defective modules.
The first data sampling technique is the Random Under-Sampling (RUS). The principle of the under-sampling method is to reduce the population of the majority class in an amount such that balanced with the minority. RUS remove majority class' instances randomly. While the distribution of both minority and majority class can be more balanced, it is risked to be losing some information.

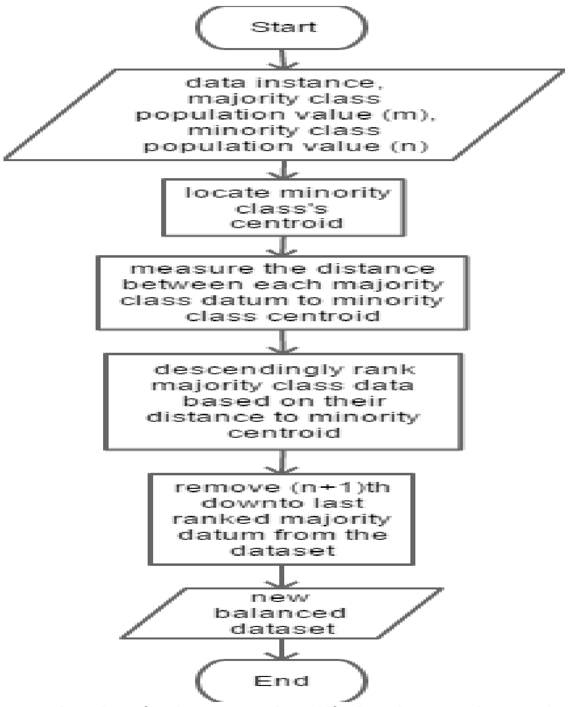
Start
data instance, majority class population value (m), minority class population value (n)
locate minority class's centroid
measure the distance between each majority class datum to minority class centroid
descendingly rank majority class data based on their distance to minority centroid
remove (n+1)th downto last ranked majority datum from the dataset
new balanced dataset
End

Fig. 1: Flowchart for the proposed modified Under-Sampling method

The second data sampling technique is a modification of the Random Under-Sampling (RUS) method. The proposed strategy in this data sampling technique is by using the distance to the centroid of the minority class to build a subset data in majority class. The first step in this method is to normalize the data. This is done with the consideration that the attributes of software metrics we are using identified as having a range of values that are not uniform. The next step is to locate the centroid of minority class in the dataset. The third step is to perform resampling based on the distance of each data in the majority class against the minority class's centroid. Data with the largest distance value are then selected as the data representation of the majority class in the training set. These alternative steps are taken because we assumed that those data will form a separate cluster such that it can be easily recognized by the classifier. Figure 1 shows the flowchart of the proposed under-sampling algorithm. In addition, the no-sampling technique is also used in combination with the chosen classifier as one of scenario used in this study.

2.2. Random Forest

Random forest which was developed by [10] is one of the today's popular classification technique in data mining. It is a decision tree-based algorithm that is enhanced by using an ensemble procedure called Bagging. Bagging is an acronym of Bootstrap Aggregating, also introduced by [11] with the aim of reducing predictor variance. The basic idea of this ensemble method is to use random sampling with replacements in the initial dataset to obtain a new dataset, which is then called in-bag partitions, to produce various versions of the classification tree. Some other smaller partitions called out-of-bag partitions are then used to test the built trees. All tree classification results are then combined using majority voting to obtain final predictions. Because each tree is considered a weak learner, the combination of many tree class versions is expected to be a strong learner and improve classification performance compared to single classification trees. It can be seen that the performance measure in the Random Forest algorithm depends on the approximate method of out-of-bag error. Figure 2 shows the simplified protocol of random forest algorithm.

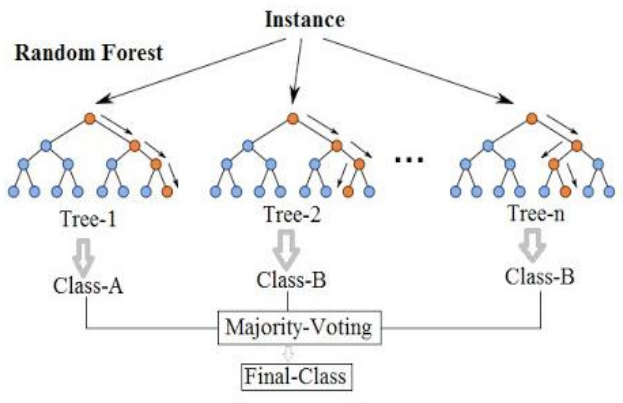

Fig. 2: Protocol of Random Forest Method.

One of the things that affect the error rate of a random forest algorithm is the correlation between two trees in a forest. High correlation value will decrease the performance of random forest algorithm. One way that can be done to overcome the problem is to increase the diversity of the classification trees. The more varied, the lower the correlation between trees. This can be done by optimizing the selection of random attributes for the formation of each tree. Increasing the generated tree population also helps improve the accuracy of random forest.
The second thing that affects the error rate of random forest is the classification power of each tree in the forest. The stronger an individual tree in the forest classifies the input data, the lower the

error rate generated by the algorithm. In [10] also stated that overfitting is not a problem when using random forest algorithms in the classification. This is because the result of combining trees results tends to converge.

2.3. Performance Metrics

There are four performance metrics used in this study. They are overall accuracy, sensitivity, f-measure, and AUC measures. The metrics are calculated after the classification results in the experiment are summarized in Confusion Matrix format. Confusion matrix is a two-dimensional matrix that describes the comparison between predicted and reality results in the form of True Positive, True Negative, False Positive, and False Negative values. Equations (1) to (4) respectively are equations used to calculate accuracy, sensitivity or True Positive Rate, F-Measure, and AUC. Equation (5) is an equation used to calculate precision in order to obtain the f-measure value.

$$Accuracy = \frac{TN + TP}{FP + TP + FN + TN} \quad (1)$$

$$Sensitivity = \mathrm{Re}call = TP_{rate} = \frac{TP}{TP + FN} \quad (2)$$

$$FMeasure = \frac{(1+\beta^2) \times recall \times precision}{\beta \times recall + precision} \quad (3)$$

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} \quad (4)$$

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} \quad (5)$$

where FP = False Positive, TN = True Negative, TP = True Positive, FN = False Negative and β = Control Parameter, 0 < β.

In addition, our selection of performance metrics is based on the work of [12] which states that for data whose class distribution is unbalanced, accuracy is more dominated by accuracy in minority class data. Therefore, it is suggested to better use the AUC than the overall accuracy measure. To measure minority class accuracy in this study, we used TPrate (Sensitivity) metrics.

The under the ROC (Receiving Operating Characteristic) Curve (AUROC or AUC) area is a numerical measure to distinguish model performance and shows how successful and true the model rank is by separating positive and negative observations. AUC provides a single measure of classifier performance to assess, which models are better on average. AUC summarizes classifier performance information into a single number that makes model comparison easier when no dominating ROC curve. AUC is a good way to get the performance value of the classifier in general and to compare it with other classifiers. AUC is a popular performance measure in class imbalance, high AUC values show better performance. So, to choose which model is the best can be done by analyzing the value of AUC.

The third and fourth performance measure used in this study is the f-measure (also called f-score) and the true positive rate (also called sensitivity). f-measure is defined as the weighted harmonic average of the precision and recall of a measurement. f-measure is commonly used as a performance metric of the classification models, including in the case of software defect prediction.

3. Data Set Description

NASA dataset used in this study was obtained from the MDP (Metric Data Program) Repository [13], which can also be obtained from the PROMISE Repository. NASA datasets have been widely used for research in the field of software engineering [2]. This dataset is devoted to research on the topic of software defect and software failures. Therefore, most studies use this dataset to do some research, ranging from doing predictions, associations, and to the development of a model for research.

NASA dataset currently available from many Repository (MDP and PROMISE). Therefore, the researchers used a dataset derived from the MDP Repository who has been repaired by [13] with the following specifications [14] in Table 1. All attributes in NASA MDP dataset are numeric. Dataset has been fixed by removing data null or no value. Accordingly, this study used the dataset to do some research in determining the proposed model in the prediction of software defects.

Datasets that used in this study are JM1, CM1, KC3, KC1, MC1, MC2, PC5, PC4, PC3, and PC1.

Table 1: Dataset Characteristics

Dataset	Number of Attribute	Number of Module	Number of Defect
CM1	37	327	42
JM1	21	7782	1672
KC1	21	1183	314
KC3	39	194	36
MC2	39	125	44
PC1	37	705	61
PC2	37	729	16
PC3	37	1077	134
PC4	37	1287	177
PC5	38	1711	471

4. Experiment

The main objective of the experimental study is to examine the effectiveness of three data sampling scenarios when combined with Random Forest classifier. The scenarios are:

- Scenario 1 (S1): Random Forests model is used alone in combination with the original data as the training data set.
- Scenario 2 (S2): Random Forests model is used in combination with Random Under-Sampling technique.
- Scenario 3 (S3): Random Forests model is used with the proposed under sampling technique, which then be called DtCUS method.

Random Forest classifier used in this study is implemented using Weka [15], a Java-written suite of machine learning tool, developed by scientists at the University of Waikato. Evaluation of the performance by each scenario is also done using Weka. We chose the 10-fold Cross Validation technique, which is dividing each dataset into ten equal and complementary sections. Nine of the ten sections are used as training sets to form decision trees in the random forest and the remainder of the dataset is the test set. The process is repeated ten times, as each section is assigned as test set alternately. In each iteration, a random forest is build and trained using the remaining nine sections.

5. Results and Discussion

The results of our experiments are shown in Table 2 to Table 5. We can see from Table 2 the performance comparison of the three scenarios on the basis of their accuracy.

Overall, the proposed strategy of under sampling method outperforms both the non-sampling and random under-sampling technique with some exception on CM1, KC3, and PC2 dataset. However, a significant difference between the proposed strategies with the best performer only happens in the case of KC3 dataset which differs by almost twelve percent.

Table 2: Total accuracy measurements

	Scenarios		
Dataset	RF	RUS+RF	DtCUS+RF
CM1	85.84	70.03	83.64
JM1	79.26	63.79	85.10
KC1	75.69	61.54	82.90
KC3	79.22	69.66	67.36
MC2	70.47	63.83	73.97
PC1	92.00	81.47	94.43
PC2	97.83	67.08	90.25
PC3	87.38	76.34	91.19
PC4	89.92	86.22	94.94
PC5	77.36	70.85	85.07

Table 3: Area under ROC Curve

	Scenarios		
Dataset	RF	RUS+RF	DtCUS+RF
CM1	0.75	0.78	0.89
JM1	0.70	0.69	0.91
KC1	0.70	0.66	0.91
KC3	0.74	0.73	0.80
MC2	0.72	0.70	0.82
PC1	0.87	0.88	0.99
PC2	0.79	0.79	0.98
PC3	0.83	0.82	0.97
PC4	0.94	0.94	0.99
PC5	0.80	0.78	0.92

Table 4: F-Measure

	Scenarios		
Dataset	RF	RUS+RF	DtCUS+RF
CM1	0.03	0.72	0.84
JM1	0.29	0.64	0.86
KC1	0.41	0.61	0.83
KC3	0.13	0.69	0.63
MC2	0.46	0.61	0.71
PC1	0.32	0.82	0.94
PC2	0.00	0.69	0.86
PC3	0.20	0.77	0.91
PC4	0.53	0.87	0.95
PC5	0.50	0.72	0.85

Table 5: True Positive Rate

	Scenarios		
Dataset	RF	RUS+RF	DtCUS+RF
CM1	0.02	0.79	0.86
JM1	0.20	0.64	0.88
KC1	0.33	0.61	0.83
KC3	0.10	0.71	0.63
MC2	0.39	0.59	0.70
PC1	0.23	0.87	0.93
PC2	0.00	0.79	0.87
PC3	0.14	0.80	0.92
PC4	0.42	0.93	0.96
PC5	0.41	0.74	0.88

This result shows that data sampling approach can also produce higher accuracy rate, compared to some other study which accuracy performance are sacrificed in order to get better results on either AUC or F-Measure.

Our new proposed model of software defect prediction gain improvements on total accuracy in the range of 2.20 to 7.21 when compared against second-best performer, which is the combination of random forest and no-sampling scenario. The proposed models also gain improvements in the context of AUC, which differ 0.1 points on average. The other two performance metrics, f-measure and TPrate, also show significant improvement on the proposed model.

6. Conclusion

The experimental results show that the proposed data sampling strategy on NASA defect prediction datasets provides a significant improvement on the classification performance when combined with Random Forest classifier. This classification model showed significantly better performance than the combination of Random Forest either with the natural unsampled training datasets nor the basic random under-sampling method. Our experiments also show that the total accuracy did not have to be sacrificed in order to gain better results on other performance metrics of imbalance learning.

References

[1] Jones, C., & Bonsignour, O. (2011). The economics of software quality. Addison-Wesley Professional.

[2] Boehm, B., & Basili, V. R. (2001). Top 10 list [software development]. Computer, 34(1), 135-137.

[3] Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., & Bener, A. (2010). Defect prediction from static code features: Current results, limitations, new approaches. Automated Software Engineering, 17(4), 375-407.

[4] Catal, C. (2011). Software fault prediction: A literature review and current trends. Expert Systems with Applications, 38(4), 4626-4636.

[5] Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38(6), 1276-1304.

[6] Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering, 37(3), 356-370.

[7] Wahono, R. S. (2015). A systematic literature review of software defect prediction. Journal of Software Engineering, 1(1), 1-16.

[8] López, V., Fernández, A., García, S., Palade, V., & 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.

[9] Yap, B. W., Rani, K. A., Rahman, H. A. A., Fong, S., Khairudin, Z., & Abdullah, N. N. (2014). An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. Proceedings of the First International Conference on Advanced Data and Information Engineering, pp. 13-22.

[10] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

[11] Breiman, L. (1996). Bagging predictors. Machine Learning, 24(2), 123-140.

[12] Zhang, H., & Wang, Z. (2011). A normal distribution-based oversampling approach to imbalanced data classification. Proceedings of the International Conference on Advanced Data Mining and Applications, pp. 83-96.

[13] Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data quality: Some comments on the NASA software defect datasets. IEEE Transactions on Software Engineering, 39(9), 1208-1215.

[14] Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2012). Reflections on the NASA MDP data sets. IET Software, 6(6), 549-558.

[15] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. ACM SIGKDD Explorations Newsletter, 11(1), 10-18.