Ensemble Undersampling to Handle Unbalanced Class on Cross-Project Defect Prediction

A Saifudin$^{1,2,*}$, Y Heryadi$^2$, Lukas$^2$

$^1$Deparment of Informatics Engineering, Pamulang University, Jalan Raya Puspitek 46, Banten 15310, Indonesia
$^2$Doctor of Computer Science Program, Bina Nusantara University, Jalan Kebon Jeruk Raya 27, Jakarta 11530, Indonesia

$^*$aries.saifudin@unpam.ac.id

Abstract. There has been much research which proposed for cross-project software defect prediction models but no models that perform very well with various datasets in general. Software defect dataset usually imbalanced because it contains far more the not defected modules than the defected modules. Class imbalances in the dataset can reduce the performance of classifiers in the software defect prediction model. In this study proposed a Random Undersampling algorithm to balance classes and ensemble techniques to reduce misclassification. The ensemble technique used is the AdaBoost and Bagging algorithm. The results showed that the software defect prediction model that integrates the Random Undersampling algorithm and AdaBoost provides better performance and can find more defects than other models.

1. Introduction

Technological advances are accompanied by elevated use and complexity of software program, in order that assured software quality is a critical and difficult challenge[1]. Software development has extended increasingly sophisticated and complexes, this is additionally accompanied through the level of software program defects which can be stated to growth proportionally[2]. A software defect is a bug that causes the software which develop can't meet expectation[3] or error, fault, flaw, or failure in the software that causes system produces an unexpected or incorrect outcome[4].

The prediction of software defect is used to assist developers to test properly and can discover defects speedy. There have been many studies that propose models for software defect prediction to assist developers to streamline efforts in testing and debugging[5]. The proposed model is used to predict whether project modules tend to contain defects or not which is generally base on quality metrics or static codes metrics [2].

Most software developers don't document their work properly so that making it difficult to analyse software development history data. The use of limited historical data for software defect prediction has attracted the attention of researchers and practitioners[6]. Using datasets for software defect prediction from different project historical data is called cross-project defect prediction. The purpose of developing a cross-project software defect prediction model is to analyse the defect tendency of a project module that is being worked on based on the artefacts of other previous projects[7].
In general, software metrics datasets are imbalance because the amount of defected data is far less than not defected data [8]. Parameter accuracy cannot be used to evaluate the performance of unbalanced class in datasets [9]. Building a model without pre-processing and use unbalanced data will tend to produce a majority class [10], so cannot find the defect because it is a minority class. To handle class imbalance can use three approaches, that are the algorithmic level approach, the data level approach, and the ensemble method [11]. The data level approach includes various resampling techniques, manipulating training data to improve class distribution bias, such as Random Oversampling or Random Undersampling [12]. The algorithmic level approach by developing new classification algorithms[13] or modify the existing algorithm. In this study, we propose the employment of Random Undersampling to balance classes is expected to reduce the influence of class imbalances. To reduce misclassification proposed the application of ensemble techniques, namely AdaBoost and Bagging. While the classification algorithm used is Naive Bayes.

Many papers on cross-project software defects prediction have been published. Researchers have proposed various classification, pre-processing, and optimization techniques but none have produced perfect results for all projects in general. Before starting the research, it is necessary to study the earlier research, to find out the methods, data, and models that have been used. This study review will be used as a research basis to find out the state of the art about research cross-project defects predictions.

Ryu and Baik [8] stated that if you don't have enough local data, you can use cross-project datasets. The software metrics dataset used for software defect prediction has class imbalance problems because the amount of data containing defects is far less than data that does not contain defects. In this context, classifiers must be able to provide high accuracy to predict defective classes without reducing accuracy to predict non-defective classes. In this study identified the effectiveness of multi-objective learning techniques to predict software defects using cross-project datasets. Here is a proposed novel multi-objective Naive Bayes learning technique that is modelled using harmony meta-heuristic search algorithms. The experimental results show that the proposed model is effective to predict software defect in the cross-project setting with an AUC value of 0.72.

Zhang, et. al [5] state that in reality many projects are not documented sufficiently, so it is proposed to use cross-project datasets. The use of cross-project datasets has become a new challenge for the prediction area of software defects. In this study proposed software defect prediction models using classification algorithms, namely Ave (Average Voting), Max (Maximum Voting), CODEP Logistic, Bagging Naive Bayes, Bagging J48, Boosting Naive Bayes, Boosting J48, and Random Forest (RF). To measure the performance of the classification algorithm, NASA datasets are used which are also part of the PROMISE repository. The dataset used is CM1, MW1, PC1, PC3, and PC4 which have the same features. The results showed that the Naive Bayes Bagging produced the best average performance with an average AUC value of 0.757.

2. Method
This work is experimental research. This experiment carried out by proposing software defect prediction models, then applying to software metrics. The results of model performance measurements are compared to get the best model.

In this experiment use secondary data, namely datasets that had been collected by other researchers. This work used NASA dataset because it is the most widely used dataset in this study so that it is easy to compare with other researchers. The NASA dataset is obtained from https://github.com/klainfo/NASADefectDataset which is a backup of http://nasa-softwaredefectdatasets.wikispaces.com/ from Shepperd et al. (2014). NASA datasets contain 10 datasets, but for this work, we use datasets which have the same attributes, namely CM1, MW1, PC1, PC3, and PC4.

The summarize of datasets specifications used are shown in Table 1. The datasets have been processed based on the initial processing algorithm proposed by Shepperd, Song, Sun, and Mair[14] to eliminate implausible value, inconsistent data, and conflicting feature value. The collected datasets
contain unbalanced class because of the total modules as many as 3579 and only 428 modules are defective or around 11.96%.

Table 1. Dataset Specification

| Dataset | Number of Modules | Number of Defected Modules | Defected Module Ratio |
|---------|-------------------|----------------------------|-----------------------|
| CM1     | 327               | 42                         | 12.84%                |
| MW1     | 250               | 25                         | 10.00%                |
| PC1     | 679               | 55                         | 8.10%                 |
| PC3     | 1053              | 130                        | 12.35%                |
| PC4     | 1270              | 176                        | 13.86%                |
| **Total** | **3579**         | **428**                    | **11.96%**            |

The purpose of this study is to apply a data level approach and ensemble technique to reduce the effect of class imbalanced on the dataset. This research is expected to produce software defect prediction models that better to find defect classes, and fewer false alarms, so that the accuracy of software defect prediction is better.

The proposed framework of the prediction model for this work is shown in Figure 1. Random Undersampling algorithm is implemented to handle class imbalanced in the software defect dataset. While to reduce misclassification of software defect prediction, an ensemble algorithm (Bagging/AdaBoost) is applied because it can improve the classification accuracy[15]. Naïve Bayes algorithm use as a classification algorithm because it is considered a classification algorithm that is efficient[16], effective[17], and generally performs well[18].

Naïve Bayes is a machine learning method that uses probability methods. Naïve Bayes shows high accuracy and speed when applied on large datasets. In other condition, Naïve Bayes should be used if it does not have enough data to calculate dependency accurately [19]. The Naïve Bayes equation is based on the Bayes theorem as follows:
Where C is class and x is feature value. For features with continuous values, it is considered to have a Gaussian distribution with mean (μ) and standard deviation (σ) [20]. So, the equation is as follows:

\[
P(x|C) = \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

The purpose of this model is to predict defect prone modules in other projects. The proposed model is applied using 5 datasets from NASA. These datasets will be chosen alternately as testing data and the other as training data until all datasets have been testing data. The distribution of the dataset as training data and testing data is shown in Figure 2.

![Figure 2. Dataset Distribution for Validation](image)

In the first validation, the first dataset is used as the testing data, while the second until fifth datasets are training data. In the second validation, the second dataset is used as testing data, and the other as training data. Validation is repeated until all datasets have been used as testing data.

Validation results are used to measure model performance. To measure the performance of the model used the confusion matrix. A confusion matrix is a useful tool for analyzing how well classifiers can recognize tuples/features of different classes [21]. Confusion matrix also provides performance appraisal of classification models based on the number of objects predicted correctly and incorrectly [22]. The confusion matrix is a 2-dimensional matrix shown in Table 2.

| Validation | Split   |
|------------|---------|
| 1          | Testing |
| 2          | Training Testing |
| 3          | Training |
| 4          | Training Testing |
| 5          | Training Testing |

Table 2. Confusion Matrix

| Class Prediction | True | False |
|------------------|------|-------|
| True             | TP (True Positive) | FP (False Positive) |
| False            | FN (False Negative) | TN (True Negative) |

The performance of the model can be seen from the value of Accuracy or AUC. To calculate the performance of the model the following equation can be used[22]:

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN}
\]

\[
TP_{rate} = \frac{TP}{TP + FN}
\]

\[
FP_{rate} = \frac{FP}{FP + TN}
\]
The AUC can be calculated based on the approximate average trapezoidal plane for curves made by \(TP_{rate}\) and \(FP_{rate}\)[23]. AUC is calculated as the area size of the ROC (Receiver Operating Characteristic) curve using equation (6)[24].

\[
AUC = \frac{1 + TP_{rate} - FP_{rate}}{2}
\]  

(6)

3. Results and Discussion

According to the proposed model in Figure 1, to find out the performance of the basic model that applies the Naïve Bayes algorithm as a classification without being optimized, the dataset is applied alternately as testing data and training data. The second model is integrating Naïve Bayes with Random Under sampling. The third model is integrating Naïve Bayes, Random Undersampling and AdaBoost. The fourth model is integrating Naïve Bayes, Random Undersampling and Bagging. The validation results of all models are shown in Table 3 and Table 4.

| Testing Dataset | Naïve Bayes | RUS + Naïve Bayes |
|-----------------|-------------|-------------------|
| TP   | FP | FN | TN | TP   | FP | FN | TN | Acc | AUC | TP   | FP | FN | TN | Acc | AUC |
| CM1  | 21 | 67 | 21 | 218 | 42 | 274 | 0  | 11 | 16,21% | 0,682 |
| MW1  | 13 | 13 | 12 | 212 | 13 | 13 | 13 | 212 | 89,60% | 0,808 |
| PC1  | 35 | 147 | 20 | 477 | 49 | 261 | 6  | 363 | 60,68% | 0,769 |
| PC3  | 27 | 70 | 103 | 853 | 32 | 79 | 98 | 844 | 83,19% | 0,749 |
| PC4  | 26 | 68 | 150 | 1026 | 23 | 64 | 153 | 1030 | 82,91% | 0,726 |
| Sum  | 122 | 365 | 306 | 2786 | 158 | 691 | 270 | 2460 |  |
| Average | 80,98% | 0,762 |  |

| Testing Dataset | RUS + AdaBoost + Naïve Bayes | RUS + Bagging + Naïve Bayes |
|-----------------|-----------------------------|-----------------------------|
| TP   | FP | FN | TN | TP   | FP | FN | TN | Acc | AUC | TP   | FP | FN | TN | Acc | AUC |
| CM1  | 40 | 219 | 2 | 66 | 42 | 268 | 0 | 17 | 18,04% | 0,692 |
| MW1  | 12 | 12 | 13 | 212 | 13 | 10 | 13 | 15 | 88,80% | 0,807 |
| PC1  | 44 | 211 | 11 | 412 | 49 | 250 | 6 | 374 | 62,30% | 0,79 |
| PC3  | 32 | 79 | 98 | 844 | 34 | 82 | 96 | 841 | 83,10% | 0,755 |
| PC4  | 129 | 462 | 47 | 632 | 23 | 64 | 153 | 1030 | 82,91% | 0,721 |
| Sum  | 257 | 985 | 171 | 2166 | 158 | 677 | 270 | 2474 |  |
| Average | 66,46% | 0,668 | 67,03% | 0,753 |

The performance of the model is calculated based on the results of validation using equation (3)-(6). The results of the model performance calculations are compiled in Table 5. For visualization, the comparison of the performance of the model presented using the graph in Figure 3.

| Model               | AVG ACC | AVG AUC | SUM ACC | SUM AUC |
|---------------------|---------|---------|---------|---------|
| Naïve Bayes         | 80,98%  | 0,762   | 81,25%  | 0,585   |
| RUS + Naïve Bayes   | 66,52%  | 0,747   | 73,15%  | 0,575   |
| RUS + AdaBoost + Naïve Bayes | 66,46% | 0,668 | 67,70% | 0,644 |
| RUS + Bagging + Naïve Bayes | 67,03% | 0,753 | 73,54% | 0,577 |

Based on the comparison chart in Figure 3, it can be seen that the software defect prediction model that applies the Naïve Bayes classification algorithm without optimization has a better performance.
than other models when performance is calculated from the average value. If the performance is calculated from the total value of the validation results, the general accuracy of the Naïve Bayes model is still higher, but the performance is based on a lower AUC value.

Figure 3. Model performance comparison

For unbalanced data, accuracy is dominated by accuracy in minority class data then the right metric is AUC[25]. AUC is a popular performance measure in unbalanced class, a model that has a high AUC value as a better performing model[26].

If the model's performance is seen from the AUC value based on the sum of the validation results, it can be seen that the model that implements Random Undersampling integration with Naïve Bayes and optimized AdaBoost shows better performance. Although the Naïve Bayes model has high accuracy, it cannot find software defects better (based on True Positive values). In our opinion, if the performance is measured from the average value of the prediction model is not appropriate, because each dataset has a different number of modules. If you want to calculate the average performance, it needs to be weighted.

4. Conclusion
Software defect prediction models are important research topics because the use of software continues to grow and quality assurance must be followed. Based on the proposed model, it has not been obtained a model that produces a very good performance. The results of the study show that the model that integrates Random Undersampling and AdaBoost provides better performance. The proposed model can find software defects better.

References
[1] S. Aleem, L. F. Capretz, and F. Ahmed, “Benchmarking Machine Learning Techniques for Software Defect Detection,” Int. J. Softw. Eng. Appl., 6(3), pp. 11–23, 2015.
[2] M. F. Adak, “Software defect detection by using data mining based fuzzy logic,” 2018 Sixth Int. Conf. Digit. Information, Networking, Wirel. Commun., pp. 65–69, 2018.
[3] R. Malhotra and S. Kamal, “Tool to handle imbalancing problem in software defect prediction using oversampling methods,” 2017 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2017, 2017–Janua, pp. 906–912, 2017.
[4] M. C. M. Prasad, L. Florence, and A. Arya, “A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques,” 8(3), pp. 179–190,
2015.

[5] Y. Zhang, D. Lo, X. Xia, and J. Sun, “Combined Classifier for Cross-project Defect Prediction: An Extended Empirical Study,” Front. Comput. Sci., 12(2), pp. 280–296, 2018.

[6] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, “Cross-project Defect Prediction Using a Connectivity-based Unsupervised Classifier,” in 2016 IEEE/ACM 38th IEEE International Conference on Software Engineering, 2016, pp. 309–320.

[7] Q. Yu, S. Jiang, and Y. Zhang, “A Feature Matching and Transfer Approach for Cross-Company Defect Prediction,” J. Syst. Softw., 132, pp. 366–378, 2017.

[8] D. Ryu and J. Baik, “Effective multi-objective naïve Bayes learning for cross-project defect prediction,” Appl. Soft Comput. J., 49, pp. 1062–1077, 2016.

[9] C. Catal, “Performance evaluation metrics for software fault prediction studies,” Acta Polytech. Hungarica, 9(4), pp. 193–206, 2012.

[10] T. M. Khoshgoftaar, K. Gao, and N. Seliya, “Attribute selection and imbalanced data: Problems in software defect prediction,” Proc. - Int. Conf. Tools with Artif. Intell. ICTAI, 1, pp. 137–144, 2010.

[11] B. W. Yap et al., “An Application of Oversampling , Undersampling , Bagging and Boosting in Handling Imbalanced Datasets,” pp. 13–23, 2014.

[12] N. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” J. Artif. Intell. Res., 16, pp. 321–357, 2002.

[13] Q. Wang, “A Hybrid Sampling SVM Approach to Imbalanced Data Classification,” Abstr. Appl. Anal., 2014(4), pp. 1–7, 2014.

[14] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the NASA software defect datasets,” IEEE Trans. Softw. Eng., 39(9), pp. 1208–1215, 2013.

[15] S. Huda et al., “An ensemble oversampling model for class imbalance problem in software defect prediction,” IEEE Access, 3536(c), 2018.

[16] H. Zhang, L. Jiang, and J. Su, “Augmenting naive Bayes for ranking,” Proc. 22nd Int. Conf. Mach. Learn. - ICML '05, 1, pp. 1020–1027, 2005.

[17] W. Tao, L. I. Wei-hua, and A. Overview, “Naïve Bayes Software Defect Prediction Model,” 2006, pp. 0–3, 2010.

[18] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A Systematic Review of Fault Prediction Performance in Software Engineering,” Softw. Eng. IEEE Trans., PP(99), p. 1, 2011.

[19] E. Alpaydin, Introduction to Machine Learning, 53(9). 2013.

[20] M. M. Jain and P. V. Richariya, “An Improved Techniques Based on Naive Bayesian for Attack Detection,” Int. J. Emerg. Technol. Adv. Eng., 2(1), pp. 324–331, 2012.

[21] H. Jiawei, M. Kamber, J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques. 2012.

[22] F. Gorunescu, Data mining: concepts and techniques. 2011.

[23] R. Dubey, J. Zhou, Y. Wang, P. M. Thompson, and J. Ye, “Analysis of sampling techniques for imbalanced data: An n=648 ADNI study,” Neuroimage, 87, pp. 220–241, 2014.

[24] V. López, A. Fernández, and F. Herrera, “On the importance of the validation technique for classification with imbalanced datasets: Addressing covariate shift when data is skewed,” Inf. Sci. (Ny), 257, pp. 1–13, 2014.

[25] H. Zhang and Z. Wang, “A Normal Distribution-Based Over-Sampling Approach to Imbalanced Data Classification,” Adv. Data Min. Appl., pp. 83–96, 2011.

[26] I. Press, L. Shafer, G. W. Arnold, and D. Jacobson, Imbalanced Learning: Foundations, Algorithms, and Applications. New Jersey: Wiley, 2013.