Tackling Imbalanced Class on Cross-Project Defect Prediction Using Ensemble SMOTE

A Saifudin¹²*, S W H L Hendric², B Soewito², F L Gaol², E Abdurachman², Y Heryadi²

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

*aries.saifudin@unpam.ac.id

Abstract. The dataset with imbalanced class can reduce the performance of the classifiers. In this study proposed a cross-project software defect prediction model that applies the SMOTE (Synthetic Minority Oversampling Technique) to balance classes in datasets and ensembles technique to reduce misclassification. The ensemble technique using AdaBoost and Bagging algorithms. The results of the study show that the model that integrates SMOTE and Bagging provides better performance. The proposed model can find more software defects and more precise.

1. Introduction

The use of software has increased in line with technological developments. The software has been developed increasingly large and complex to provide more benefits. Software that has a very large and complex size requires large resources for testing and quality assurance. By considering time and cost, testing cannot be performed on each software module[1]. Many models have been proposed for software defects prediction to estimate which modules contain defects[2]. 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 produce an unexpected or incorrect outcome[4].

Software defect prediction aims to estimate which modules contains defect-prone so that software engineers can reduce the resources needed to correct defects and improve software quality[5]. There have been many studies that propose models for software defect prediction to help developers streamline efforts in testing and debugging[6]. The proposed model is used to predict whether project modules tend to contain defects or not.

Studied on software defect detection is generally based on quality metrics or static codes metrics[7]. Evaluation of software quality can use software metric values, for example from the Cyclomatic Complexity (CC) metrics value can be known whether the class/module has a tendency to contain defects and need to be modified, or not[8].

The publicly available NASA (National Aeronautics and Space Administration) datasets is software metrics that are very popular in the development of prediction models for software defects, because 62 studies from 208 studies have used NASA datasets[9]. The NASA publicly available dataset has been widely used as part of software defect research[10]. If using the same dataset, the results of the study can be easily compared.
The use of limited historical data for software defect prediction has attracted the attention of researchers and practitioners [11]. 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 [12].

In this study proposed to joining several datasets from different projects to predict cross-project software defects. By joining several different projects to train the proposed model, the size of the datasets becomes large. In other side, software metrics datasets 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 [13].

Parameter accuracy cannot be used to evaluate the performance of unbalanced class in datasets [14]. Building a software quality model without initial processing of data will not produce a predictive model of effective software defects, because if the data used is not balanced then the prediction results tend to produce a majority class [15]. Because software defects are a minority class, many defects cannot be found.

Class imbalances can be tackling with three approaches, that are the algorithmic level approach, the data level approach, and the ensemble method [16]. The data level approach includes various resampling techniques, manipulating training data to improve class distribution bias, such as Random Oversampling and Random Undersampling, and SMOTE (Synthetic Minority Oversampling Technique) [17]. Algorithmic level approach by developing new classification algorithms [18] or modify the existing algorithm.

In this study, we propose the employment of SMOTE 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 Naïve Bayes.

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 dataset consists of 38 attributes which are divided into four groups, namely:
1) Six LOC Counts Attributes, consists of Blank LOC, LOC Code and Comment, LOC Comments, LOC Executable, LOC Total, and Number of Lines
2) Twelve Halstead Attributes, consists of Content, Difficulty, Effort, Error Estimate, Length, Level, Program Time, Volume, Number of Operands, Number of Operators, Number of Unique Operands, and Number of Unique Operators
3) Four McCabe Attributes, consists of Cyclomatic Complexity, Cyclomatic Density, Design Complexity, and Essential Complexity
4) Seventeen Miscellaneous Attributes, consists of Count of Branch, Call Pairs, Count of Condition, Count of Decision, Decision Density, Design Density, Count of Edge, Essential Density, Maintenance Severity, Count of Modified Condition, Count of Multiple
Condition, Count of Node, Normalized Cyclomatic Complexity, Count of Parameter, Percent Comments, Defective

The summarize of datasets specifications used are shown in Table 1. The datasets has been processed based on the initial processing algorithm proposed by Shepperd, Song, Sun, and Mair[10] to eliminate implausible value, inconsistent data, and conflicting feature value.

| Table 1. Dataset Specification |
|--------------------------------|
| NASA Dataset Repository        | CM1 | MW1 | PC1 | PC3 | PC4 |
| Attribut Count                 | 38  | 38  | 38  | 38  | 38  |
| Module Count                   | 327 | 250 | 679 | 1053| 1270|
| Defected Module Count          | 42  | 25  | 55  | 130 | 176 |
| Defected Module Ratio          | 12.84% | 10.00% | 8.10% | 12.35% | 13.86% |

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. SMOTE 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[19]. Naïve Bayes algorithm use as a classification algorithm because it is considered a classification algorithm that is efficient[20], effective[21], and generally performs well[9].
Naïve Bayes is a machine learning method that uses probability methods. The probability of class membership can be used by classifiers for prediction and the process is based on the Bayes theorem. 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[22]. The Naïve Bayes equation is based on the Bayes theorem as follows:

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

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)[23]. So, the equation is as follows:

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 1. Software Defect Prediction Model Framework](image)

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

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 [24]. Confusion matrix also provides performance appraisal of classification models based on the number of objects predicted correctly and incorrectly [25]. The confusion matrix is a 2-dimensional matrix shown in Table 2.

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

![Table 2. Confusion Matrix](image)
When doing validation, if both the prediction and the actual are True, it is called True Positive (TP). If the prediction is True and the actual is False, it is called False Positive (FP). If both the prediction and the actual are False, it is called True Negative (TN). If the prediction is False and the actual is True, it is called False Negative (FN). A good classifier’s performance will produce high True Positive (TP) and True Negative (TN) values, while the numbers of False Positive (FP) and False Negative (FN) values are low is a weak classifier.

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 [25]:

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

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

\[
FP_{rate} = \frac{FP}{FP + TN}
\]

The AUC can be calculated based on the approximate average trapezoidal plane for curves made by TP_{rate} and FP_{rate}[26]. AUC is calculated as the area size of the ROC (Receiver Operating Characteristic) curve using equation (6) [27].

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

### 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 results of model validation are shown in Table 3. Then the values of TP, FP, FN, and TN are summed and put on the confusion matrix in Table 4.

| Testing Dataset | Confusion Matrix | Acc | AUC |
|-----------------|------------------|-----|-----|
| CM1             | 21 67 21 218     | 73.09% | 0.695 |
| MW1             | 13 13 12 212     | 90.00% | 0.823 |
| PC1             | 35 147 20 477    | 75.41% | 0.81  |
| PC3             | 27 70 103 853    | 83.57% | 0.745 |
| PC4             | 26 68 150 1026   | 82.83% | 0.736 |
| Sum             | 122 365 306 2786 | 80.98% | 0.762 |

### Table 4. Confusion Matrix of Naïve Bayes model

| Confusion Matrix | Actual | Y    | N    |
|------------------|--------|------|------|
| Predict          |        | 122  | 365  |
|                  |        | 306  | 2786 |
|                  |        | 28.50% | 88.42% |
Based on the values in the confusion matrix of the Naive Bayes model in Table 4, the model performance of the model can be calculated as follows:

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{122 + 2786}{122 + 365 + 306 + 2786} = 81.25\%
\]

\[
TP_{rate} = \frac{TP}{TP + FN} = \frac{122}{122 + 306} = 0.285
\]

\[
FP_{rate} = \frac{FP}{FP + TN} = \frac{365}{365 + 2786} = 0.1158
\]

\[
AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} = \frac{1 + 0.285 - 0.1158}{2} = 0.5846
\]

The second model is integrating Naïve Bayes with SMOTE. The dataset for training is balanced first using SMOTE then used to train Naïve Bayes classifiers. The validation results are shown in Table 5 and inserted into the confusion matrix table 6.

**Table 5. Measurement of the SMOTE+Naïve Bayes model**

| Testing Dataset | Confusion Matrix | Acc | AUC |
|-----------------|------------------|-----|-----|
| CM1             | 30 118 12 167    | 60.24% | 0.69 |
| MW1             | 17 36 8 189      | 82.40% | 0.824 |
| PC1             | 46 373 9 251     | 43.74% | 0.718 |
| PC3             | 39 86 91 837     | 83.19% | 0.752 |
| PC4             | 77 145 99 949    | 80.79% | 0.745 |
| Sum             | 209 758 219 2393 | 70.07% | 0.746 |

**Table 6. Confusion Matrix of SMOTE+Naïve Bayes model**

| Confusion Matrix | Actual |        |
|-----------------|--------|--------|
|                 | Y      | N      |
| Predict         | 209    | 758    |
|                 | 48.83% | 75.94% |

Based on the values in the confusion matrix of the SMOTE+Naïve Bayes model in Table 6, the model performance of the model can be calculated as follows:

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{209 + 2393}{209 + 2393 + 758 + 219} = 72.70\%
\]

\[
TP_{rate} = \frac{TP}{TP + FN} = \frac{209}{209 + 758} = 0.4883
\]

\[
FP_{rate} = \frac{FP}{FP + TN} = \frac{758}{758 + 2393} = 0.2406
\]
The third model is integrating Naïve Bayes, SMOTE and AdaBoost. The dataset for training is balanced first using SMOTE then used to train Naïve Bayes classifiers optimized with the AdaBoost algorithm. The validation results are shown in Table 7 and inserted into the confusion matrix Table 8.

### Table 7. Measurement of the AdaBoost+SMOTE+Naïve Bayes model

| Testing Dataset | Confusion Matrix | Acc  | AUC   |
|-----------------|------------------|------|-------|
| CM1             | TP: 27, FP: 162, FN: 15, TN: 123 | 45.87% | 0.553 |
| MW1             | TP: 15, FP: 124, FN: 10, TN: 101 | 46.40% | 0.611 |
| PC1             | TP: 46, FP: 373, FN: 9, TN: 251 | 43.74% | 0.699 |
| PC3             | TP: 94, FP: 348, FN: 36, TN: 575 | 63.53% | 0.726 |
| PC4             | TP: 57, FP: 128, FN: 119, TN: 966 | 80.55% | 0.731 |
| Sum             | TP: 239, FP: 1135, FN: 189, TN: 2016 | Average 56.02% | 0.664 |

### Table 8. Confusion Matrix of AdaBoost+SMOTE+Naïve Bayes model

| Confusion Matrix | Actual  | TP: 239, FP: 1135, FN: 189, TN: 2016 | | |
|------------------|---------|------------------------------------|--|--|
| Predict | Y | 239 | 1135 | 17.39% |
|         | N | 189 | 2016 | 91.43% |
|         |      | 55.84% | 63.98% |

Based on the values in the confusion matrix of the AdaBoost+SMOTE+Naïve Bayes model in Table 8, the model performance of the model can be calculated as follows:

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{239 + 2016}{239 + 2016 + 1135 + 189} = 63.01\%
\]

\[
TP_{rate} = \frac{TP}{TP + FN} = \frac{239}{239 + 189} = 0.5584
\]

\[
FP_{rate} = \frac{FP}{FP + TN} = \frac{1135}{1135 + 2016} = 0.3602
\]

\[
AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} = \frac{1 + 0.5584 - 0.3602}{2} = 0.599
\]

The fourth model is integrating Naïve Bayes, SMOTE and AdaBoost. The dataset for training is balanced first using SMOTE then used to train Naïve Bayes classifiers optimized with the Bagging algorithm. Validation results are shown in Table 9 and inserted into the confusion matrix Table 10.

### Table 9. Measurement of the Bagging+SMOTE+Naïve Bayes model

| Testing Dataset | Confusion Matrix | Acc  | AUC   |
|-----------------|------------------|------|-------|
| CM1             | TP: 30, FP: 120, FN: 12, TN: 165 | 59.63% | 0.699 |
Based on the values in the confusion matrix of the Bagging+SMOTE+Naïve Bayes model in Table 10, the model performance of the model can be calculated as follows:

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{208 + 2406}{208 + 2406 + 745 + 220} = 73.04\%
\]

\[
TP_{rate} = \frac{TP}{TP + FN} = \frac{208}{208 + 220} = 0.486
\]

\[
FP_{rate} = \frac{FP}{FP + TN} = \frac{745}{745 + 2406} = 0.2364
\]

\[
AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} = \frac{1 + 0.486 - 0.2364}{2} = 0.625
\]

The results of the model performance calculations are compiled in Table 11. For visualization the comparison of the performance of the model presented using the graph in Figure 3.

Table 11. Summary of model performance

| Model                          | AVG ACC | AVG AUC | SUM ACC | SUM AUC |
|--------------------------------|---------|---------|---------|---------|
| Naïve Bayes                    | 80.98%  | 0.762   | 81.25%  | 0.585   |
| SMOTE+Naïve Bayes              | 70.07%  | 0.746   | 72.70%  | 0.624   |
| AdaBoost+SMOTE+Naïve Bayes     | 56.02%  | 0.664   | 63.01%  | 0.599   |
| Bagging+SMOTE+Naïve Bayes      | 70.61%  | 0.755   | 73.04%  | 0.625   |

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.
For unbalanced data, accuracy is dominated by accuracy in minority class data then the right metric is AUC[28]. AUC is a popular performance measure in class imbalance, a model that has a high AUC value as a better performing model[29].

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 applies SMOTE integration with Naïve Bayes, and the model that implements SMOTE integration with Naïve Bayes and optimized Bagging 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.

![Figure 3. Model performance comparison](image)

### 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 SMOTE and Bagging provides better performance. The proposed model can find software defects better.

### References

[1] S. Moustafa, M. Y. Elnainay, N. El Makky, and M. S. Abougabal, “Software bug prediction using weighted majority voting techniques,” *Alexandria Eng. J.*, vol. 57, no. 4, pp. 2763–2774, 2018.

[2] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, “A general software defect-proneness prediction framework,” *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 356–370, 2011.

[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*, vol. 2017–January, 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,” vol. 8, no. 3, pp. 179–190, 2015.

[5] R. Özakınç and A. Tarhan, “Early Software Defect Prediction: A Systematic Map and Review,” *J. Syst. Softw.*, vol. 144, pp. 216–239, 2018.

[6] Y. Zhang, D. Lo, X. Xia, and J. Sun, “Combined Classifier for Cross-project Defect Prediction:
An Extended Empirical Study,” *Front. Comput. Sci.*, vol. 12, no. 2, pp. 280–296, 2018.

[7] 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.

[8] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal, “Empirical Analysis of Change Metrics for Software Fault Prediction,” *Comput. Electr. Eng.*, vol. 67, pp. 15–24, 2018.

[9] 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.*, vol. PP, no. 99, p. 1, 2011.

[10] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the NASA software defect datasets,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013.

[11] 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.

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

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

[14] C. Catal, “Performance evaluation metrics for software fault prediction studies,” *Acta Polytech. Hungarica*, vol. 9, no. 4, pp. 193–206, 2012.

[15] 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*, vol. 1, pp. 137–144, 2010.

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

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

[18] Q. Wang, “A Hybrid Sampling SVM Approach to Imbalanced Data Classification,” *Abstr. Appl. Anal.*, vol. 2014, no. 4, pp. 1–7, 2014.

[19] S. Huda *et al.*, “An ensemble oversampling model for class imbalance problem in software defect prediction,” *IEEE Access*, vol. 3536, no. c, 2018.

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

[21] W. Tao, L. I. Wei-hua, and A. Overview, “Naïve Bayes Software Defect Prediction Model,” no. 2006, pp. 0–3, 2010.

[22] E. Alpaydin, *Introduction to Machine Learning*, vol. 53, no. 9, 2013.

[23] M. M. Jain and P. V. Richaria, “An Improved Techniques Based on Naive Bayesian for Attack Detection,” *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 1, pp. 324–331, 2012.

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

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

[26] 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*, vol. 87, pp. 220–241, 2014.

[27] 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)*, vol. 257, pp. 1–13, 2014.

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

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