The Effectiveness of Resampling Method for Handling Class Imbalances in Software Defect Prediction

Frieyadie
Malaysian Institute of Information Technology (MIIT)
Universiti Kuala Lumpur
Kuala Lumpur, Malaysia
frieyadie@s.unikl.edu.my

Munaisyah Abdullah
Malaysian Institute of Information Technology (MIIT)
Universiti Kuala Lumpur
Kuala Lumpur, Malaysia
munaisyah@unikl.edu.my

Foni Agus Setiawan
Department of Informatics Engineering
Universitas Ibn Khaldun Bogor
Bogor, Indonesia
masagus@uika-bogor.ac.id

2023 International Conference on Information Technology Research and Innovation (ICITRI) | 979-8-3503-2494-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICITRI59340.2023.10249255

***Abstract*— Defect prediction is crucial for software products to be high-quality and reliable. Class imbalance, however, in which one class does much better than the other, poses a significant challenge to flaw prediction models. This inequality often results in discriminatory behavior towards the majority class, resulting in poor performance in identifying the defects of the minority class. By undersampling the dominant class, oversampling the minority class, or combining the two, resampling entails changing the distribution of the dataset. This study aims to develop a robust and accurate model that can overcome the limitations of class imbalance and improve overall crash prediction performance. Logistic regression, a widely used classification algorithm, offers interpretability and flexibility, making it suitable for defect prediction. This research investigates the effectiveness of the resampling technique in conjunction with logistic regression to deal with the class imbalance in defect prediction software. Accuracy and UAC measurement result from the t-test for 12 MDP datasets show that Logistic Regression+Sample (Bootstrapping) works much better than Logistic Regression, with** an **average accuracy of 90.78% and** an **average AUC of 0.81.**

Keywords—Resampling Technique, Class Imbalance, Software Defect Prediction

I. INTRODUCTION

Software defect prediction (SDP) is a crucial problem in software engineering. [1], [2], The goal of ensuring the quality and dependability of software products is critical. [3]–[5] identify and mitigate potential defects in software systems during the development and maintenance phases. Accurate defect prediction enables organizations to allocate resources effectively [6], prioritize testing efforts [7], and improve software quality [8].

However, defect prediction models face a class imbalance challenge [9], [10], where the number of instances belonging to one class significantly outweighs the other [11]. This class imbalance poses a significant obstacle [12] because traditional prediction models demonstrate discriminatory behavior towards the majority class. They perform poorly in spotting minority class flaws.

This mismatch in class distribution might result in biased models that fail to identify faulty cases [13]. Researchers and practitioners have proposed various techniques to address this issue, with resampling techniques and logistic regression emerging as promising countermeasures. Resampling strategies entail altering the dataset's distribution by oversampling the minority class, undersampling the majority class, or a mix of the two. On the other hand, logistic regression, a widely used classification algorithm, offers flexibility and interpretability, making it an attractive choice for defect prediction tasks.

The class imbalance issue in SDP has been addressed using various resampling strategies [14]. Resampling techniques manipulate the class distribution [15] either through an undersized majority class sample (non-defective examples) or a significant minority class sample (defective cases). These methods are intended to increase the accuracy of SDP models [16], [17] by creating a more balanced representation of the classes.

The advantages of the resampling technique include overcoming class disappointment in the data. Reduces variability in parameter estimates. It improves the stability of the resulting model and produces more consistent forecasts. The resampling technique can produce a smaller sample size than the original sample. This can lead to a loss of variation data which can affect the security of the resulting model.

The research objective is to evaluate the effectiveness of resampling techniques in handling class imbalances for SDP. The study investigates how different resampling techniques impact the predictive performance of SDP models.

II. RELATED WORK

Malhotra & Jain's [18] results showed that using resampling techniques significantly improved the model's predictive ability compared to the classical boosting model in dealing with the problem of class imbalance in software flaw prediction. Evaluate the model using stable performance evaluation metrics such as Balance, G-Mean, and Area Under the Curve (AUC). Using appropriate performance evaluation metrics such as G-Mean, Balance, and AUC can help build models when handling unbalanced data. Some of the shortcomings of this study include: 1) Only focusing on the use of boosting-based ensemble algorithms to deal with class imbalance problems in software flaw prediction. Many other techniques, such as cost-sensitive resampling and ensemble techniques, can solve this problem. 2) Only use three datasets of open-source software projects, so the generalization of research results is limited to these datasets. 3) Do not consider other factors affecting classification performance, such as data distribution, cross-validation, and dataset shifting. 4) Do not compare the performance of boosting-based ensemble algorithms with other techniques used in related literature.

The research conducted by Bennin, Keung, & Monden [19] shows that resampling methods can improve software defect classification but are ineffective for defect prioritization. The Random Under-Sampling (RUS) method is considered the most stable and effective resampling method. In addition, it was found that the selection of resampling methods and evaluation criteria can significantly affect the performance of prediction models. The authors also recommend using the ADASYN method for defect prediction in unbalanced datasets. Some of the shortcomings of this study are: 1) The use of datasets consisting only of open-source projects, so the results cannot be generalized to commercial projects. 2) The use of only five flaw prediction models, so the results may not represent the performance of other flawed prediction models. 3) Limited use of evaluation metrics, such as accuracy, precision, and recall, without considering other metrics, such as F1-score or areas under the ROC curve.

Elahi, Kanwal, & Asif [20] investigated the effect of different resampling techniques on the performance of five classifications in software error prediction. This research shows that resampling techniques such as RUS, Random Under Sampling (ROS), and SMOTE help enhance classification efficiency on imbalanced datasets. In addition, it also shows that certain combinations of classification and resampling techniques can produce better performance than other combinations. Identify some threats to the validity of the results, such as using datasets from the PROMISE repository that may not be generalizable to actual software datasets. Some of the shortcomings of this study are: 1) Only used four datasets from the PROMISE repository, so the generalization of research results is limited to these datasets. 2) Only use the average model method for ensembles, so do not consider other ensemble methods such as stacking and voting.

Ruchika Malhotra & Kamal [21], This study's results suggest that using data resampling techniques, especially the Safe-Level-SMOTE method, can improve the performance of software maintenance prediction models on unbalanced datasets. Different machine learning techniques can give different results in predicting software maintenance efforts. Some of the shortcomings of this study are as follows: 1) The dataset used in this study consisted of only eight open-source software projects. Therefore, the generalization of the results of this research to other software projects may be limited. 2) Using only two machine learning techniques, C4.5 and MLP-CG, to build predictive software maintenance models.

Research conducted by Li et al. [22] led to creating of a software defect prediction model that might enhance classification efficiency using an ideal ensemble technique, the Random Forest algorithm mixed with data sampling technology. The study used five software flaw prediction data sets from NASA's MDP database as experimental research objects. The Adaboost Algorithm, the Bagging Algorithm, the Response Surface Methodology (RSM) Algorithm, the Random Forest (RF) Algorithm, and the Vote Algorithm were five software defect prediction algorithms in this study that were then compared through experiments, and the performance evaluation results were superior to using the J48 classification. Some of the shortcomings of this study include: 1) Only using five sets of software flaw prediction data from NASA's MDP database as experimental research objects. Thus, the generalization of the results of this study is limited to the dataset used. 2) Only use data sampling technology (SMOTE oversampling and Resample undersampling) to optimize data quality by balancing majority and minority categories.

Iqbal, Aftab & Matloob [14] examined several classification models on several datasets, showing that Random Over Sampling (ROS) performs best on most datasets with most classification models. However, in some datasets, such as MC2, SMOTE performed better. At the same time, Random Under Sampling (RUS) does not perform well except on the KC1 dataset with several classification models. In addition, resampling techniques can solve the class imbalance problem in most datasets, except for one PC2. Several research deficiencies can be identified: 1) an insufficient explanation of how the resampling technique is selected and implemented. As well as can affect the interpretation of software flaw prediction results. 2) There is no sufficient explanation of how the study can be applied to other cases beyond the dataset used in the study.

These advantages highlight the effectiveness of resampling techniques in improving predictive ability, classification performance, and software flaw prediction. They also emphasize the importance of selecting appropriate evaluation metrics, identifying optimal resampling methods, considering different classification and resampling technique combinations, and understanding the performance variations across different datasets and models.

III. Research Methodology

A. *Software Defect Prediction*

A binary classification issue in machine learning, software defect prediction (SDP) determines whether a module is faulty or defect-free [20]. Software defect prediction remains vital research to improve software quality since software products' increasing complexity and dependability increase the cost of software testing. The SDP was a project focus because early defect detection enhances software quality while lowering costs and managing software effectively [21]. Most SDP research techniques employ machine learning algorithms to build predictive models from training data extracted from software repositories and then use those models to detect software issues in test data. Within-project defect prediction (WPDP) and cross-project defect prediction [22], [23] are two common research topics in software defect prediction based on distinct sources of training data. Within-Since training and test data are typically gathered from the same software project, WPDP believes that they must be delivered separately and identically. Many conventional categorization techniques are employed in WPDP to create prediction models.

B. Imbalance Class

The imbalance class in software defect prediction research refers to the unequal distribution of defective and non-defective modules in software defect datasets [24]. This imbalance occurs when the number of non-defective modules significantly outweighs the number of defective modules.

Furthermore, experts highlight the need for thorough evaluation and experimentation when dealing with a class imbalance in software defect prediction research. This includes comparing different resampling techniques, exploring the combination of resampling with other preprocessing methods (e.g., feature selection), and assessing the performance of predictive models using appropriate validation strategies.

Experts aim to develop more robust and accurate models that can effectively identify and predict software defects by addressing the class imbalance in software defect prediction research. The ultimate goal is to enhance software quality, minimize the occurrence of defects, and improve the reliability and stability of software systems.

C. Resampling Technique

Resampling techniques are potent methods used in data analysis and machine learning to address the challenges of imbalanced datasets. These techniques have gained significant attention from experts in the field due to their effectiveness in improving the performance of predictive models and tackling class imbalance issues.

A class imbalance occurs when the distribution of classes in a dataset is severely skewed. One class is notably underrepresented in comparison to the others. This presents difficulties for machine learning algorithms, which are biased towards the majority class and struggle to catch patterns and produce correct predictions for the minority class.

Resampling techniques offer a solution by manipulating the dataset to create a more balanced representation of the classes . As a result, either more members of the minority class will be represented (oversampling), fewer members of the majority class will be represented (undersampling), or a hybrid strategy will be used [3].

The bootstrap resampling method estimates a statistic's sampling distribution by resampling from the available data. The basic equation for the bootstrap resampling method is as follows:

1. Start with a dataset of size n, denoted as D.
2. Repeat the following steps B times (where B is the number of bootstrap iterations):
   a. Draw a random sample (with replacement) from the original dataset D, creating a bootstrap sample D* of the same size n.
   b. Compute the desired statistic (e.g., the sample mean, standard deviation, etc.) using the bootstrap sample D*.
3. Collect the computed statistics from each bootstrap iteration to obtain the bootstrap distribution.
4. Analyze the bootstrap distribution to estimate properties such as confidence intervals or standard errors.

Denote the original dataset as $D = \{x_1, x_2, ..., xn\}$, where xi represents an individual data point. The bootstrap resampling method involves creating bootstrap samples D* by randomly selecting n data points (with replacement) from D. For example, to estimate the mean using the bootstrap resampling method, the equation would be:

$$mean(D^*) = (1/n) * \sum X_i \tag{1}$$

where x_i is a data point in the bootstrap sample D*.

Similarly, the bootstrap resampling method can estimate statistics such as standard deviation, median, or any other desired measure. The key idea is to repeatedly sample from the original dataset and compute the statistic of interest on each bootstrap sample to approximate the sampling distribution.

D. Logistic Regression

A straightforward logistic regression model may be used to obtain the OR and the 95% Confidence Intervals (CI) for any predictor, whether continuous or dichotomous [25]. The expected result variable is G, where G = 1 means something like renal failure has happened. G = 0 signifies the absence of the event. Set H_1 as the predictor variable. When there are numerous predictors, generalization is made using subscript 1. Given the predictor's value, the probability that the event will occur may be stated using the logistic model, denoted as $F(G = 1 \mid H_1)$. The fundamental assumption is that there is a linear relationship between the predictor variable(s) and the log of the probability that G = 1 occurs. The probability of the event or illness occurring is represented below, given the predictor variable, X1 (G = 1). This might be stated as

$$\log odds[G = 1|H_1] = log\left[\frac{F(G=1|H_1)}{1-F(G=1|H_1)}\right]\beta_0 + \beta_1 G_0, \tag{2}$$

Where β_0 is the intercept and β_1 is the regression coefficient of H_1. The odds log is the logit transformation, and the coefficients are on a logarithmic scale. The model is a linear regression model in the log odds that G = 1. transformation derived from Equation 2. The model is a linear regression model in the log chances that G = 1 derived from Equation 2.

$$F(G = 1|XH_1) = \frac{1}{1+\exp[-(\beta_0+\beta_1 H_1)]} = \frac{exp(\beta_0+\beta_1 H_1)}{1+exp(\beta_0+\beta_1 H_1)} \tag{3}$$

E. Hypothesis

Value 0.05 (or 5%) was used as the significance level for a statistical t-test. The following criteria were used to determine whether to accept or reject this hypothesis test:

- If the significant result exceeds 0.05, the alternative hypothesis (H_1) is rejected, and the null hypothesis (H_0) is accepted. It shows that the independent variable's influence on the dependent variable is minimal.
- The alternative hypothesis (H_1) is accepted, and the null hypothesis (H_0) is rejected if the significant value is less than 0.05. It means that the independent variable only partially and substantially affects the dependent variable.

F. Data Collection

The data understanding used is public data. This dataset is publicly available on the internet, namely 12 Downloadable datasets from the NASA (National Aeronautics and Space Administration) Metrics Data Programme (MDP) repository https://github.com/klainfo/NASADefectDataset. The NASA MDP Repository dataset is readily available and publicly available. The dataset used is public data uses dataset D'' from NASA MDP, namely 12 Nasa MDP Repository Datasets consisting of classes (CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1 PC2, PC3, PC4, PC5).

G. Proposed Method

The method proposed in this study is to improve the performance of the Logistic Regression algorithm with the resampling method to handle a class imbalance in predicting software defects, furthermore, for validation using 20-fold cross-validation. The results of measuring the algorithm's performance use the t-test (t-test) to determine differences in model performance after and before the resampling model is applied.

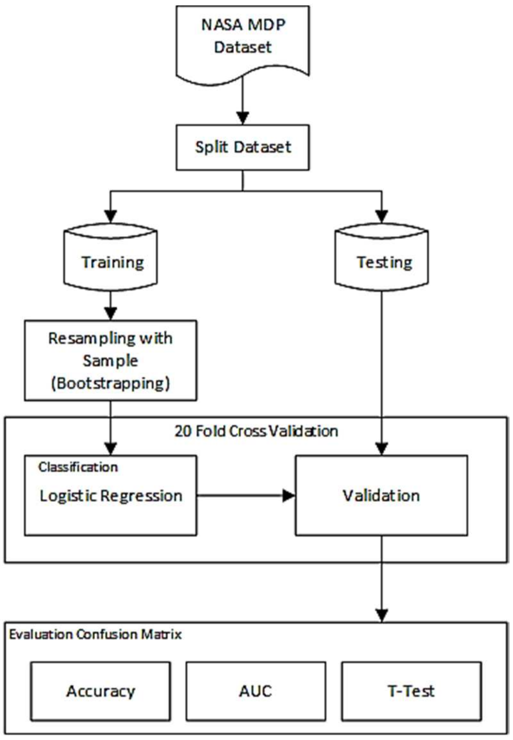

Fig. 1. Proposed Model

Fig. 1 displays the use of the NASA MDP public dataset repository that researchers have commonly used in Software Engineering research [26]. NASA MDP data is devoted to software defects and failures research topics. Table 1 shows the Nasa MDP Repository Dataset.

TABLE I. PERCENTAGE 12 DATASET

Data Set	Attribute	Instances	Defect Class	Non-Defect Class
CM1	38	327	42	285
JM1	22	7782	1672	6110
KC1	22	1183	314	869
KC3	40	194	36	158
MC1	39	1988	46	1942
MC2	40	125	44	81
MW1	38	253	27	226
PC1	38	705	61	644
PC2	37	145	16	729
PC3	38	1077	134	943
PC4	38	1287	177	1110
PC5	39	1711	417	1240

Table I shows the number of disabled classes from the characteristics and the number of modules, faulty modules, and non-disabled modules from each dataset. Class jealousy (Imbalance Class) from the defect and non-defect classes influence high or low percentage outcomes.

IV. RESULTS AND DISCUSSION

Using the Logistic Regression (L.R.) classifier method, the approach was put to the test. The experimental findings are shown in Table II. The data are precision, positive predictive value (PPV) or accuracy, recall, specificity, negative predictive value (NPV) or F.P. rate, F-Measure, and AUC.

TABLE II. LOGISTIC REGRESSION EXPERIMENT RESULT

Dataset	Accuracy	AUC	TP	TN	FP	FN	Recall	Specificity	PPV	NPV	F-Measure
CM1	85,64%	0,77	6	36	11	274	12,50%	96,14%	35,29%	88,50%	16,67%
JM1	79,69%	0,69	182	1430	138	5970	11,29%	97,74%	56,68%	80,60%	18,62%
KC1	77,03%	0,71	69	225	40	828	23,54%	95,17%	61,98%	78,60%	34,07%
KC3	81,50%	0,57	10	26	9	149	94,38%	23,33%	85,24%	52,60%	89,38%
MC1	97,80%	0,73	1	35	8	1908	5,00%	99,58%	11,11%	98,20%	4,44%
MC2	68,10%	0,69	19	25	15	66	40,83%	81,50%	55,88%	73,90%	48,72%
MW1	90,55%	0,67	7	20	5	231	26,67%	97,91%	58,33%	92,14%	35,90%
PC1	91,18%	0,84	12	612	7	48	98,08%	15,00%	92,77%	36,84%	95,32%
PC2	97,23%	0,50	0	16	4	702	0,00%	99,43%	0,00%	97,77%	0,00%
PC3	88,04%	0,83	25	105	21	902	18,98%	97,72%	54,35%	89,59%	28,41%
PC4	89,53%	0,85	61	115	18	1076	34,92%	98,36%	75,24%	90,41%	47,84%
PCS	97,17%	0,94	104	354	63	1173	94,90%	22,65%	76,88%	61,91%	84,91%

According to the experimental findings in Table II, the 12 datasets' average accuracy is 86.96%, and their average AUC is 0.73.

TABLE III. LOGISTIC REGRESSION+SAMPLE (BOOTSTRAPPING) EXPERIMENT RESULT

Dataset	Accuracy	AUC	TP	TN	FP	FN	Recall	Specificity	PPV	NPV	F-Measure
CM1	88,03%	0,869	17	23	16	271	44,17%	94,50%	51,52%	92,52%	46,58%
JM1	82,63%	0,711	170	1424	149	5977	10,66%	97,57%	52,98%	80,76%	17,61%
KC1	88,12%	0,831	77	215	39	831	26,46%	95,53%	71,64%	79,50%	37,18%
KC3	85,56%	0,788	12	20	9	153	94,49%	37,50%	88,88%	57,14%	91,28%
MC1	99,01%	0,882	4	33	7	1908	12,50%	99,64%	36,36%	98,30%	16,67%
MC2	88,21%	0,877	27	8	7	83	78,33%	92,50%	82,08%	92,00%	78,26%
MW1	92,39%	0,788	5	19	7	232	20,00%	97,08%	41,67%	92,59%	27,78%
PC1	91,70%	0,888	12	46	12	609	98,06%	20,83%	93,02%	50,00%	95,44%
PC2	98,51%	0,508	0	13	4	705	0,00%	99,43%	0,00%	98,18%	0,00%
PC3	88,04%	0,82	27	103	23	900	21,17%	97,50%	54,00%	89,75%	30,00%
PC4	89,93%	0,806	55	121	25	1069	31,10%	97,71%	70,27%	89,84%	42,37%
PCS	97,22%	0,944	108	320	54	1212	95,73%	25,17%	79,13%	66,87%	86,63%

Table III displays the experimental findings, which reveal that the average accuracy in the 12 datasets is 90.78%, and the average AUC is 0.81. Fig. 2 shows that dealing with class imbalance improves model performance outcomes in the NASA MDP dataset. As seen in Fig. 2, the resulting performance difference is negligible.

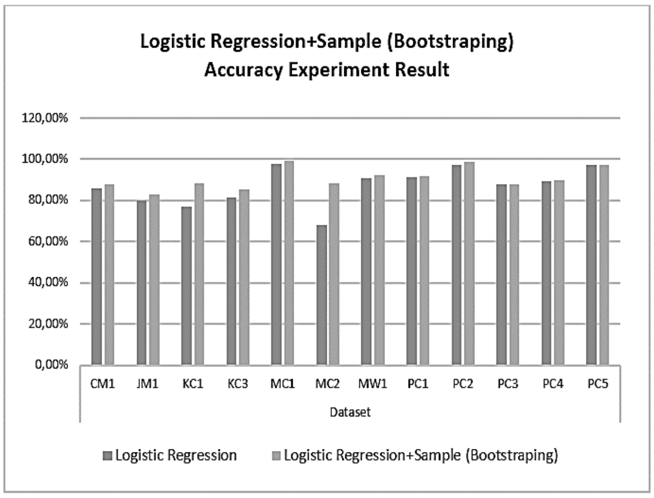

Fig. 2. Image of Accuracy Measurement Recap on Software Defect Prediction

Tables II and table III give the findings of a comparison of Area Under Curve (AUC) L.R. and L.R. with Resampling (L.R. + Sample (Bootstrapping) after Fig. 3.

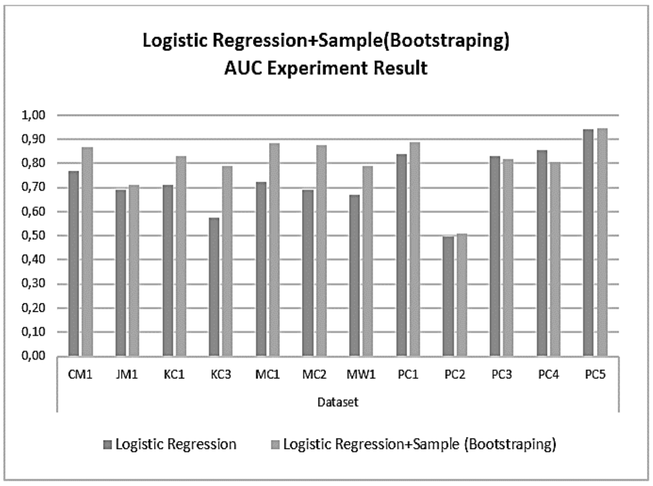

Fig. 3. Image of AUC Measurement Recap on Software Defect Prediction

Table III compares AUC LR and LR + Sample (Bootstrapping). The AUC measurement graph shows improved performance after applying the resampling approach to datasets with class imbalance, as shown in Fig. 3, with improved performance on CM1, JM1, KC1, KC3, MC1, MC2, MW1, and PC1. The PC2 and PC5 datasets showed no substantial increase in value.

The paired sample t-test for the Logistic Regression accuracy variable and the Logistic Regression + Sample (Bootstrapping) variable can be seen in Table IV.

TABLE IV. PAIRED SAMPLE T-TEST ACCURACY LOGISTIC REGRESSION DAN LOGISTIC REGRESSION+SAMPLE (BOOTSTRAPPING)

	Logistic Regression	*Logistic Regression + Sample (Bootstrapping)*
Mean	0,86955	0,907791667
Variance	0,008221832	0,002686744
Observations	12	12
Pearson Correlation	0,783631153	
Hypothesized Mean Difference	0	
df	11	
t Stat	-2,225745863	
P(T<=t) one-tail	**0,023940787**	
t Critical one-tail	1,795884819	
P(T<=t) two-tail	0,047881574	
t Critical two-tail	2,20098516	

From the results of the paired sample t-test in Table IV, hypothetical conclusions can be drawn based on the comparison of t count and t table, also based on probability values. The calculated t value represented by t Stat is 2.225745863. The table t value represented by t Critical two-tail is 2.20098516, so it can be ascertained that the calculated t value of the table t > which means H_0 is not accepted and H_1 is accepted, meaning that there is a difference between the accuracy results of Logistic Regression and Logistic Regression+Sample (Bootstrapping). In contrast, the probability value is known to be 0.047881574. Then it can be seen that the probability value < 0.05, which means H_0 failed to be accepted and H_1 was accepted, meaning that there is a significant difference from the average accuracy of Logistic Regression and Logistic Regression + Sample (Bootstrapping). The accuracy results show Logistic Regression+Sample (Bootstrapping) higher than Logistic Regression.

TABLE V. PAIRED SAMPLE T-TEST AUC LOGISTIC REGRESSION AND LOGISTIC REGRESSION+ SAMPLE (BOOTSTRAPPING)

	Logistic Regression	*Logistic Regression + Sample (Bootstrapping)*
Mean	0,732583333	0,809333333
Variance	0,01549372	0,012701697
Observations	12	12
Pearson Correlation	0,750156902	
Hypothesized Mean Difference	0	
df	11	
t Stat	-3,144597019	
P(T<=t) one-tail	**0,004665809**	
t Critical one-tail	1,795884819	
P(T<=t) two-tail	0,009331619	
t Critical two-tail	2,20098516	

In Table V, it is known that the probability value is 0.009331619, so it can be seen that the probability value < 0.05, which means H_0 is not accepted and H_1 is accepted, meaning that there is a significant difference from the average AUC Logistic Regression and Logistic Regression + Sample (Bootstrapping), AUC results show Logistic Regression + Sample (Bootstrapping) higher than Logistic Regression.

V. CONCLUSION

This study assessed the effectiveness of various resampling methods in dealing with a class imbalance in predicting software defects. It does not go too far into how this technique can improve predictive model performance and reduce the impact of class imbalance. Based on the t-test results concerning the accuracy, the probability value (p-value) obtained (0.047881574) is less than 0.05, further supporting the rejection of H_0 and acceptance of H_1. At the same time, it is stated that the accuracy of Logistic Regression + Sample (Bootstrapping) is higher than Logistic Regression. In the t-test on AUC, the probability value obtained from the paired sample t-test (0.009331619) is also smaller than 0.05, causing H_0 to be rejected and H_1 to be accepted. Therefore, there is a significant difference in the average AUC between Logistic Regression and Logistic Regression+ Sample (Bootstrapping). The AUC results showed that Logistic Regression+Sample (Bootstrapping) has a higher AUC than Logistic Regression. In conclusion, Logistic Regression + Sample (Bootstrapping) performs much better than Logistic Regression in accuracy and AUC.

REFERENCES

[1] C. L. Prabha and N. Shivakumar, "Software Defect Prediction Using Machine Learning Techniques," *Proc. 4th Int. Conf. Trends Electron. Informatics, ICOEI 2020*, pp. 728–733, Jun. 2020.

[2] M. A. Khan, N. S. Elmitwally, S. Abbas, S. Aftab, M. Ahmad, M. Fayaz, et al., "Software defect prediction using artificial neural networks: A systematic literature review", *Sci. Program.*, vol. 2022, pp. 1-10, May 2022.

[3] S. Goyal, "Handling Class-Imbalance with KNN (Neighbourhood) Under-Sampling for Software Defect Prediction," *Artif. Intell. Rev.*, vol. 55, no. 3, pp. 2023–2064, Mar. 2022.

[4] M. Pandit *et al.*, "Towards Design and Feasibility Analysis of DePaaS: AI-Based Global Unified Software Defect Prediction Framework," *Appl. Sci. 2022, Vol. 12, Page 493*, vol. 12, no. 1, p. 493, Jan. 2022.

[5] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools," *Eng. Appl. Artif. Intell.*, vol. 111, p. 104773, May 2022.

[6] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *Int. J. Appl. Sci. Eng.*, vol. 17, no. 4, pp. 331–344, Jan. 2020.

[7] C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu, "Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 48, no. 3, pp. 786–802, Mar. 2022.

[8] S. Wang, J. Wang, J. Nam and N. Nagappan, "Continuous software bug prediction", *Proc. 15th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, pp. 1-12, Oct. 2021

[9] A. Balaram and S. Vasundra, "Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm," *Autom. Softw. Eng.*, vol. 29, no. 1, pp. 1–21, May 2022.

[10] S. Sharma, A. Gosain, and S. Jain, "A Review of the Oversampling Techniques in Class Imbalance Problem," pp. 459–472, 2022.

[11] R. Sauber-Cole and T. M. Khoshgoftaar, "The use of generative adversarial networks to alleviate class imbalance in tabular data: a survey," *J. Big Data*, vol. 9, no. 1, pp. 1–37, Dec. 2022.

[12] A. Z. Zakaria, A. Selamat, L. K. Cheng, and O. Krejcar, "Improving Class Imbalance Detection And Classification Performance: A New Potential of Combination Resample and Random Forest," *2022 IEEE Int. Conf. Comput. ICOCO 2022*, pp. 316–323, 2022.

[13] L. Gong, S. Jiang, and L. Jiang, "Tackling Class Imbalance Problem in Software Defect Prediction through Cluster-Based Over-Sampling with Filtering," *IEEE Access*, vol. 7, pp. 145725–145737, 2019.

[14] A. Iqbal, S. Aftab, and F. Matloob, "Performance analysis of resampling techniques on class imbalance issue in software defect prediction," *I.J. Inf. Technol. Comput. Sci.*, vol. 11, no. 11, pp. 44–53, 2019.

[15] S. Feng, J. Keung, X. Yu, Y. Xiao, and M. Zhang, "Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction," *Inf. Softw. Technol.*, vol. 139, p. 106662, Nov. 2021.

[16] A. F. Iswafaza and S. Rochimah, "Software Defect Prediction Using a Combination of Oversampling and Undersampling Methods," pp. 127–132, Mar. 2023.

[17] A. O. Balogun, R. O. Oladele, H. A. Mojeed, B. Amin-balogun, V. E. Adeyemo, and T. O. Aro, "Performance Analysis of Selected Clustering Techniques for Software Defects Prediction," *African J. Comput. ICT*, vol. 12, no. 2, pp. 30–42, 2019.

[18] R. Malhotra, "Handling imbalanced data using ensemble learning in software defect prediction," *Proceedings of the Confluence 2020 - 10th International Conference on Cloud Computing, Data Science and Engineering*. pp. 300–304, 2020.

[19] K. E. Bennin, J. W. Keung, and A. Monden, "On the relative value of data resampling approaches for software defect prediction," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 602–636, 2019.

[20] Y. Liu, W. Zhang, G. Qin, and J. Zhao, "A comparative study on the effect of data imbalance on software defect prediction," *Procedia Comput. Sci.*, vol. 214, no. C, pp. 1603–1616, Jan. 2022.

[21] M. Prashanthi, G. Sumalatha, K. Mamatha, and K. Lavanya, "Software Defect Prediction Survey Introducing Innovations with Multiple Techniques," *Cogn. Sci. Technol.*, pp. 783–793, 2023.

[22] Y. Zhao, Y. Zhu, Q. Yu, and X. Chen, "Cross-Project Defect Prediction Considering Multiple Data Distribution Simultaneously," *Symmetry 2022, Vol. 14, Page 401*, vol. 14, no. 2, p. 401, Feb. 2022.

[23] Y. Z. Bala, P. Abdul Samat, K. Y. Sharif, and N. Manshor, "Improving Cross-Project Software Defect Prediction Method Through Transformation and Feature Selection Approach," *IEEE Access*, vol. 11, pp. 2318–2326, 2023.

[24] R. Malhotra, A. A. Khan, and A. Khera, "Simplify Your Neural Networks: An Empirical Study on Cross-Project Defect Prediction," *Lect. Notes Data Eng. Commun. Technol.*, vol. 75, pp. 85–98, 2022.

[25] T. G. Nick and K. M. Campbell, "Logistic Regression," in *Methods in Molecular Biology, vol. 404: Topics in Biostatistics*, W. T. Ambrosius, Ed. Totowa, NJ: Humana Press Inc., 2007, pp. 273–301.

[26] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software Defect Prediction Analysis Using Machine Learning Techniques," *Sustain. 2023, Vol. 15, Page 5517*, vol. 15, no. 6, p. 5517, Mar. 2023.