Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics e-ISSN: 2656-8624
Homepage: https://ijeeemi.org/; Vol. 7, No. 3, pp. 514-525, August 2025

RESEARCH PAPER

OPEN ACCES

An Empirical Study of Cross-Project and Within-Project Performance in Software Defect Prediction Models Using Tree-Based and Boosting Classifiers

Raidra Zenianto, Rudy Herteno, Radityo Adi Nugroho, Andi Farmadi, Setyo Wahyu Saputro

Department of Computer Science, Lambung Mangkurat University, Kalimantan Selatan, Indonesia

Abstract

Software Defect Prediction (SDP) is a vital process in modern software engineering aimed at identifying faulty components in the early stages of development. In this study, we conducted a comprehensive evaluation of two widely employed SDP approaches, Within-Project Software Defect Prediction (WP-SDP) and Cross-Project Software Defect Prediction (CP-SDP), using identical preprocessing steps to ensure an objective comparison. We utilized the NASA MDP dataset, where each project was split into 70% training and 30% testing data, and applied three distinct resampling strategies no sampling, oversampling, and undersampling to address the challenge of class imbalance. Five classification algorithms were examined, including Support Vector Machine (SVM), Random Forest (RF), Gradient Boosting (GB), XGBoost (XGB), and LightGBM (LGBM). Performance was measured primarily using Accuracy and Area Under the Curve (AUC) metrics, resulting in 360 experimental outcomes. Our findings revealed that WP-SDP, combined with oversampling and Random Forest, demonstrated superior predictive capability on most projects, achieving an Accuracy of 89.92% and an AUC of 0.931 on PC4. Nonetheless, CP-SDP excelled in certain small-scale projects (e.g., MW1), underscoring its potential when local historical data is scarce but inter-project characteristics remain sufficiently similar. This study's results underscore the importance of selecting a prediction scheme tailored to specific project attributes, class imbalance levels, and available historical data. By establishing a standardized methodological framework, our work contributes to a clearer understanding of the strengths and limitations of WP-SDP and CP-SDP, paving the way for more effective defect detection strategies and improved software quality.

Paper History
Received Jan 02, 2025
Revised July 10, 2025
Accepted August 3, 2025
Published August 5, 2025

Keywords
Software Defect;
WP-SDP;
CP-SDP;
Classification

Author Email
raidrazeni@gmail.com
rudy.herteno@ulm.ac.id
radityo.adi@ulm.ac.id

1. INTRODUCTION

As software complexity increases, so does the likelihood of defects, adversely affecting quality and security [1]. Ensuring quality before release is therefore essential [2]. The presence of multiple faults in source code necessitates repeated testing, driving up costs and resource demands [3]. Recognizing its importance, Software Defect Prediction (SDP) plays a vital role in preserving software quality by detecting defects early and preventing more significant losses [4], [5]. Automated SDP techniques have been developed that leverage historical data and relevant metrics to detect potential defects, thereby enhancing reliability and prioritizing testing efforts [6], [7]. By utilizing previous versions of source code and defect records, SDP automates the identification process and reduces manual effort [8]. However, selecting effective predictive attributes remains challenging [9], prompting the use of feature selection methods to remove redundant or irrelevant features and enhance classification performance [10].

In the face of rapid technological advancements and increasing system complexity, approaches in SDP evolved with two main approaches, namely Within-Project Software Defect Prediction (WP-SDP) and Cross-Project Software Defect Prediction (CP-SDP). WP-SDP utilizes historical defect data from the same project to build predictive models, assuming that past defect patterns are representative of future defects within the same project [11]. This approach often yields high accuracy but may struggle when there is insufficient labeled data [12]. On the other hand, CP-SDP leverages defect data from external projects to predict defects in a target project, making it useful when project-specific defect data is limited [13]. However, challenges such as differences in data distribution and feature misalignment between source and target projects can impact prediction performance [14].

Problems in Software Defect Prediction (SDP) are often faced with data imbalance and feature distribution diversity between projects, which can significantly affect the performance of predictive models [15]. Due to these data disparities, evaluating WP-SDP and CP-SDP approaches requires a special strategy to ensure objective and reliable analysis. Therefore, applying

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

resampling techniques is an important aspect of SDP research, as it helps stabilize data variability and ensures that each model is evaluated fairly [16].

Random Resampling is one of the commonly used resampling techniques and has been proven effective in various classification scenarios [17]. This technique handles data imbalance through two main approaches. First, oversampling increases the number of samples in the minority class so that the algorithm can better understand the patterns from the underrepresented data [18]. This approach is effective in reducing model bias but may increase the risk of overfitting due to the added synthetic data [19]. Secondly, undersampling reduces the number of samples in the majority class to create a more balanced distribution [20]. While it can speed up the training process and reduce storage requirements, it runs the risk of omitting important information that can affect model accuracy [21].

Once the dataset is balanced using resampling techniques, the next step in Software Defect Prediction (SDP) is to build a machine-learning model to perform classification. Various methods can be used in this process, including algorithms such as Support Vector Machine, Random Forest, Gradient Boosting, XGBoost, and LightGBM, each of which has advantages in handling complex and diverse data. The selection of the right model is an important factor in improving the accuracy of defect prediction, thus enabling more effective identification of potential software defects.

Support Vector Machine (SVM) is a machine learning algorithm that is widely used in various classification tasks in Software Defect Prediction (SDP). SVM works by finding the optimal hyperplane that separates the data classes by the largest margin, so that it can produce more accurate decisions, especially on high-dimensional data [22]. The main advantage of SVM lies in its ability to handle data that is not linearly separable through the use of kernel tricks [23], which allows the transformation of data into a higher dimensional space to find a more optimal separation. In the context of SDP, SVM has been widely applied to identify possible defects in source code, with its reliability in overcoming the problems of data imbalance and feature complexity in software projects.

Besides SVM, another widely used algorithm in classification is Random Forest (RF), which also has an important role in Software Defect Prediction (SDP). RF is an ensemble learning-based method consisting of several decision trees that work simultaneously to improve prediction accuracy [24]. By combining the results of many decision trees, RF can reduce the risk of overfitting and improve model generalization to new data [25]. Another advantage of RF is its ability to handle complex data sets and non-linear features, making it an effective choice for predicting software defects [26]. In SDP, RF is often used because of its robustness to noise in the data and its ability to identify the features that contribute most to defect prediction, thus improving the reliability of software defect detection systems.

Ensemble learning-based approaches are widely used in classification, including Gradient Boosting and XGBoost, which play an important role in Software Defect Prediction (SDP). Gradient Boosting builds a model incrementally with a series of decision trees that mutually correct previous prediction errors, resulting in a more accurate and outlier-resistant model [27]. Its advantage lies in its ability to handle unstructured data as well as non-linear relationships between features, making it an effective solution in improving defect prediction performance [28]. A further development of this method is XGBoost (Extreme Gradient Boosting), which is designed to improve efficiency and accuracy through decision tree-based optimization and regularization to reduce overfitting [29]. XGBoost excels in handling large datasets and capturing complex relationships between features [30], making it a reliable choice in various classification tasks, including SDP.

As a more efficient alternative to XGBoost, LightGBM (Light Gradient Boosting Machine) was developed to improve speed and scalability in the machine learning process. The algorithm is designed with a leaf-wise growth approach, which allows the processing of large amounts of data with less computation time than traditional boosting methods [31]. The main advantage of LightGBM is the extraordinary performance of LightGBM in terms of its precision, model stability, and computing efficiency [32].

Once the classification model is applied in Software Defect Prediction (SDP), the next step is to evaluate its performance using appropriate metrics. Two commonly used metrics in assessing prediction quality are Accuracy and Area Under the Curve (AUC-ROC) [33]. Accuracy measures the proportion of correct predictions to the overall data [34], making it a simple but less informative metric if the dataset suffers from class imbalance. Meanwhile, AUC-ROC provides a more comprehensive picture by assessing the model's ability to distinguish between defect and non-defect classes at various thresholds [35], making it more relevant for scenarios with uneven data distribution. In the context of SDP, the combination of these two metrics is often used to understand the extent to which the model can identify software defects with accuracy and balance.

While metrics such as Accuracy and AUC-ROC provide an overview of model performance in Software Defect Prediction (SDP), the main challenge in this research lies in how the comparison between Within-Project Software Defect Prediction (WP-SDP) and Cross-Project Software Defect Prediction (CP-SDP) is done fairly. Many previous studies have compared these two approaches without ensuring that the classification methods and resampling techniques used are in an apple-to-apple condition, so the results obtained cannot always be used as a clear reference. As a result, researchers often face difficulties in determining the most effective strategy to improve defect prediction accuracy, both in the context of the same project and across projects. This research gap suggests the need for a more systematic evaluation of the combination of classification methods and resampling techniques to provide more comprehensive guidance in the selection of the optimal approach for SDP.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

Due to the need for systematic evaluation in Software Defect Prediction, there are still fundamental weaknesses in the comparison between WP-SDP and CP-SDP that exist in the current literature. The study by Bhat and Farooq (2023) used varied pre-processing, evaluation metrics, and algorithms making it difficult to obtain a consistent comparison between the two methods [36], while the study by Zhu et al. (2020) relied too much on specific algorithms that could lead to bias [37], and the study by Li et al. (2025) has not deeply examined the influence of different dataset distributions and preprocessing techniques on prediction results [38]. Therefore, there is a research gap that needs to be filled through a comprehensive study that conducts an apple-to-apple comparison between WP-SDP and CP-SDP with consistent control of experimental variables to obtain a more fair and objective evaluation.

The method proposed in this study is designed to ensure a fair comparison between within-project and cross-project based software defect prediction. The technique utilizes the NASA MDP data set processed in a computational framework using Google Colab (runtime: Python 3.10, CPU: Intel Xeon 2.30 GHz, RAM: 12 GB, GPU: NVIDIA Tesla T4) and Python programming language, as shown in FIGURE 1 which presents the research flowchart. For the within-project approach (WP-SDP), data is taken from one project and then divided into 70% training data and 30% test data. In the cross-project approach (CP-SDP), training data is obtained from all projects except the project used in WP-SDP, with the same 30% of test data used as evaluation points. Furthermore, both approaches were applied to three types of sampling techniques sampling, oversampling, and undersampling which were then evaluated using five classification algorithms, namely Support Vector Machine, Random Forest, XGBoost, LightGBM, and Gradient Boosting, to obtain a comprehensive assessment of the effectiveness of software defect prediction. Model performance evaluation uses the AUC and the Accuracy metric to measure the model's ability to distinguish between defect and non-defect classes. This research framework is designed to provide a fair comparison between software defect prediction based on within-project and cross project. This study makes four main contributions: (1) objective comparison of WP-SDP and CP-SDP with identical pre-processing and test data; (2) holistic resampling impact analysis for both schemes; (3) performance evaluation of modern algorithms such as Support Vector Machine, Random Forest, XGBoost, LightGBM, and Gradient Boosting in various scenarios; and (4) practical data-driven guidance for SDP scheme selection according to historical data availability, project size, and degree of class imbalance.

2. Materials and Method

In this study, five machine learning models-Support Vector Machine, Random Forest, XGBoost, Gradient Boosting, and LightGBM-were used to evaluate the performance of three resampling techniques: no sampling, oversampling, and undersampling. Figure 1 shows the research workflow.

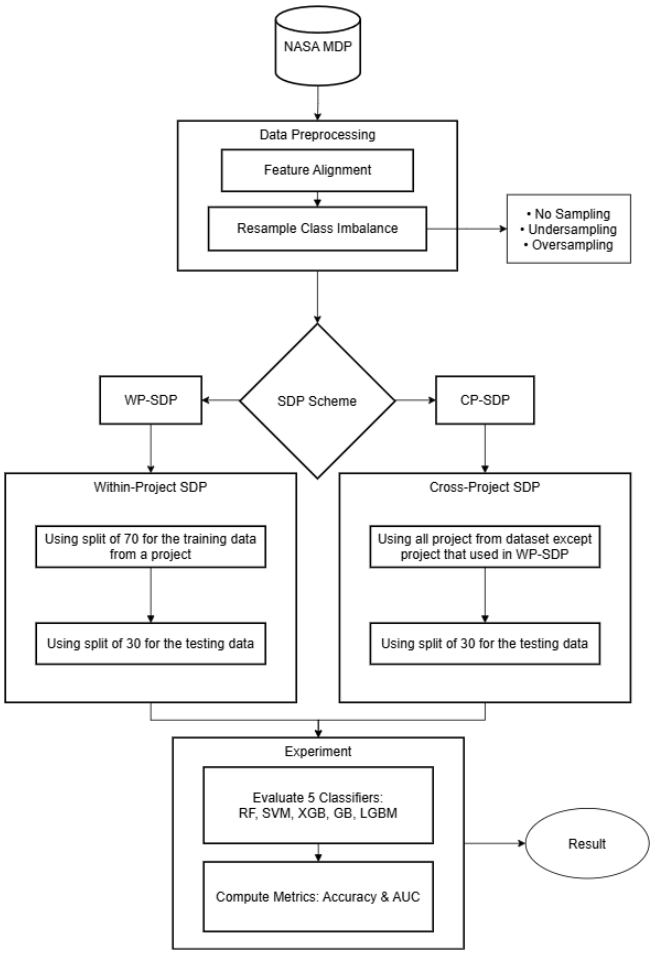

Fig. 1. Research Flowchart

A. Dataset

The NASA MDP (Metric Data Program) dataset is a collection of data provided by NASA and is widely used in software engineering research, particularly in the field of software defect prediction. This dataset contains software metrics collected from various NASA projects that shows in Table 1, including code size, complexity, and the number of defects found in the source code. The data is obtained through a data collection process that involves extracting metrics from source code and recording information about software defects based on development history. With the NASA MDP dataset, researchers can compare various software defect prediction methods, such as WPDP and CPDP, under apple-to-apple conditions to evaluate the accuracy of the models used. The NASA MDP dataset is available through the following link: https://github.com/klainfo/NASADefectDataset [39].

Table 1. Nasa MDP Dataset

Project	Programming Language	Number of Instance	Defect Ratio (%)
CM1	C	327	12.8
JM1	Java	7720	20.8
KC1	Java	1162	25.3

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

KC3	Java	194	18.6
MC1	C and C++	1988	2.3
MC2	C	125	35.2
MW1	C	253	10.7
PC1	C	705	8.7
PC2	C	745	2.1
PC3	C	1077	12.4
PC4	C	1287	13.8
PC5	C++	1711	27.5

B. Data Processing and Partitioning

When working with a biased dataset, it is important to fix imbalances in the training data. One effective method is resampling, which balances the number of instances from majority and minority classes. Additionally, we perform feature alignment in preprocessing to ensure that the same set of attributes is used across all projects before sampling, creating a consistent input space for both WP-SDP and CP-SDP. Previous studies have shown that this approach helps create a more representative dataset for training [40].

Data sampling is generally divided into two, oversampling, and undersampling [41]. We chose random resampling specifically for its simplicity and reproducibility: it allows us to adjust class distributions without introducing additional heuristics, and by fixing a seed value we ensure identical behavior across runs. Furthermore, no class weighting or hyperparameter tuning was applied, so that all observed performance differences can be attributed solely to the sampling scheme and classifier choice.

Random resampling is a basic method provided by Python libraries such as imbalanced-learn to perform data balancing by randomizing and selecting random samples from a dataset. This method serves as the foundation for applying both oversampling and undersampling techniques. By using the functions provided by the library, the class distribution in the dataset can be adjusted so that the prediction model can be trained with balanced data.

1) Random oversampling is a technique used to enhance the representation of the minority class by duplicating a selected number of its samples, sometimes with slight modifications, thereby addressing class imbalance [42]. This method calculates the total samples after oversampling as $N_0 + k \times N_1$, where N_0 represents the number of majority class samples, N_1 the number of minority samples, and k is typically set to 1 or 2. Additionally, by randomly selecting minority class examples with replacement, the technique effectively increases the dataset size and contributes to a more balanced training process [43]. However, the duplicated data from the minority class can have the same value (redundant), increasing the chance of overfitting [44].
2) Random undersampling (RUS) is a non-heuristic technique used to balance imbalanced datasets by reducing the number of samples in the majority class rather than replicating minority class instances [45]. Instead of adding more data to the minority class, this method randomly removes samples from the majority class to achieve a balanced distribution, thereby lowering the dataset's size. While this reduction can significantly decrease training time and computational cost, it runs the risk of eliminating critical data points that might be valuable for accurate model training [46]. Consequently, the efficiency gains from RUS must be carefully weighed against the potential degradation in classification performance that may result from losing important information.

A common approach for validating models is to split the data into two parts: one for training and one for testing. In this approach, the dataset is divided such that the training subset is used to build the model, while the testing subset is reserved for evaluating its performance. By separating the data in this way, an unbiased assessment of the model's predictive capabilities is ensured, and concerns about overfitting on the training data are minimized [47]. In this study, we used a 70:30 split where 70 is used for model training data and 30 is used as test data. This ratio provides a balance between providing enough samples for the model to learn complex patterns and retaining unseen data to reliably measure its generalization performance; it is a generally accepted convention that offers stable estimates without overly restricting training or evaluation.

C. Classification Algorithms

Classification algorithms are an important component in machine learning, especially in software defect prediction. Various algorithms are developed to classify data into defect and non-defect categories based on features. In this research, we use five popular classification algorithms, namely Support Vector Machine (SVM), Random Forest (RF), XGBoost, Gradient Boosting, and LightGBM. Each algorithm has its advantages in processing data, overcoming class imbalance, and achieving high prediction accuracy.

1) Support Vector Machine (SVM)

Support Vector Machine (SVM) is an effective supervised learning model for binary classification, particularly suitable for datasets with limited sample sizes [48]. In Software Defect Prediction (SDP) research, SVM is employed to predict software defect class labels by constructing an optimal hyperplane that maximizes the separation margin between instances of the two classes (Figure 2). This hyperplane, visualized in a two-dimensional feature space, is defined by the maximum distance to the nearest data points (support vectors) from both classes, thereby enhancing prediction accuracy [49].

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

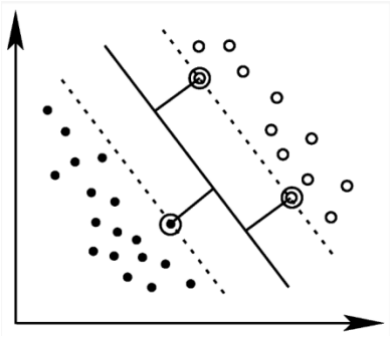

Fig. 2. The implementation of SVM using an optimal hyperplane and maximum margin for data classification.

The strength of SVM lies in its flexibility to process both linear and nonlinear data through kernel functions such as Radial Basis Function (RBF) or polynomials, unlike association rule methods that are confined to linear data [50]. The selection of kernel type, size, and parameters like gamma significantly influences the smoothness of class separation and prevents overfitting [51]. This combination of high computational efficiency and adaptability positions SVM as an efficient solution for SDP, especially in detecting hidden defect patterns within program code.

2) Random Forest (RF)

Random Forest (RF) is an ensemble algorithm designed to enhance prediction accuracy and stability by aggregating outputs from multiple decision trees [52]. Each tree is trained on randomly selected subsets of data and features, which mitigates overfitting risks while ensuring robustness in handling datasets with complex, high-dimensional features [53]. The final prediction is determined through majority voting, where the most frequent class label across all trees is selected [54]. This hierarchical approach recursively partitions data into subgroups based on splitting criteria until termination conditions are met, with terminal segments (leaf nodes) representing definitive classifications.

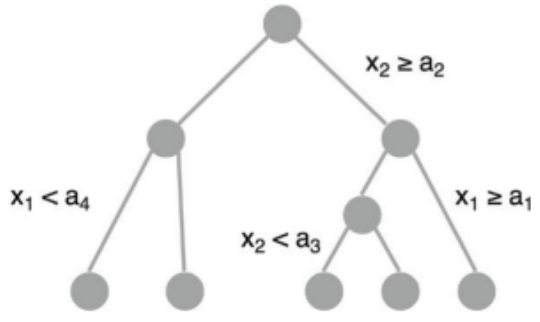

Fig. 3. A graphical representation of the decision tree

The input space is recursively divided using axis-aligned boundaries, where splits occur along coordinates parallel to feature axes. For instance, an initial partition is executed at the threshold $x_2 \geq \alpha_2$, followed by subsequent subdivisions: the left subspace is split at $x_1 \geq \alpha_4$, while the right subspace undergoes splits at $x_1 \geq \alpha_1$ and $x_2 \geq \alpha_3$. This hierarchical division generates distinct regions, as systematically illustrated in prior studies (Figure 3) [55].

3) XGBoost

XGBoost is a powerful ensemble learning method based on gradient boosting, designed for scalability and high performance on complex datasets [56]. It builds an additive model by sequentially minimizing a loss function, which combines the error term and a regularization component to control model complexity. This overall objective can be formulated as in Eq. (1) and Eq. (2):

$$L_{xgb} = \sum_{i=1}^{N} L\left(y_i, F(x_i)\right) + \sum_{m=1}^{M} \Omega\left(h_m\right) \quad (1)$$

where the regularization term is defined as

$$\Omega(h) = \gamma T + \frac{1}{2}\|w\|^2 \quad (2)$$

In addition to its elegant formulation, XGBoost incorporates advanced techniques such as random subsampling and column sampling to reduce overfitting and enhance training speed [57]. Each new decision tree is added to the ensemble with the specific goal of correcting the errors made by the previous trees, resulting in an iterative refinement process that boosts overall performance. These methodological choices make XGBoost not only efficient but also highly effective in handling large-scale and complex machine learning problems [58].

4) Gradient boosting

Gradient boosting is a powerful ensemble learning method that iteratively combines multiple weak learners into a strong predictive model [59]. Initially introduced by Friedman in 2001, this technique is widely applied in both regression and classification tasks, assigning specific weights to data points to guide the learning process. The method builds an additive model where each successive learner focuses on the residuals the errors made by the combined predictions of its predecessors. When decision trees serve as the base classifiers, the overall model can be expressed by the summation of regression trees as follows in Eq. (3):

$$F_n(x_t) = \sum_{i=1}^{n} f_i\left(x_t\right) \quad (3)$$

where each $f_i(x_t)$ represents an individual regression tree. In every iteration, gradient boosting fits a new learner to the residual errors derived from the previous iterations, utilizing gradient-based optimization instead of relying solely on misclassification weights like those in AdaBoost [60]. This process, while effective in improving accuracy, requires careful regularization such as applying shrinkage (reducing the gradient descent step size) and restricting the complexity of the decision trees (for example, by limiting their depth) to prevent overfitting [61]. Moreover,

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

incorporating randomization techniques, such as random sub sampling without replacement, further enhances the model's generalization capabilities by introducing variability during training. These combined strategies enable gradient boosting to deliver highly accurate models even in complex and noisy data environments.

5) LightGBM

LightGBM is an innovative gradient boosting framework that leverages decision tree-based learning to achieve both speed and scalability [62]. Developed to efficiently handle large-scale and complex datasets, it incorporates advanced techniques such as gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB) to reduce the volume of data and the number of features without sacrificing essential information. Moreover, it employs a histogram-based learning method combined with a leaf-wise tree growth strategy under a depth limit, which further enhances training speed and model accuracy [63].

In addition to its robust architectural design, LightGBM is widely recognized for its high-performance computing capabilities in distributed environments [64]. It supports GPU acceleration and parallel learning, which significantly lowers memory consumption and speeds up the training process across various machine learning tasks such as regression, ranking, and classification. Furthermore, by intensifying the focus on misclassified instances during successive iterations, LightGBM refines weak learners into a strong ensemble model that has been shown to outperform many traditional boosting algorithms. A unique feature that distinguishes the LightGBM algorithm from other gradient boosting tree methods is its splitting tree, as depicted in the Figure 4.

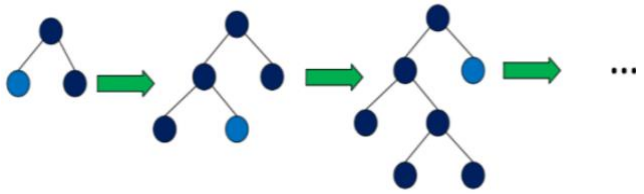

Fig. 4. Leaf-wise tree growth in LightGBM

The five classifiers SVM, Random Forest, XGBoost, Gradient Boosting, and LightGBM were chosen for their complementary strengths in addressing key challenges in defect prediction. SVM handles high-dimensional data and nonlinear patterns via kernels, while Random Forest reduces overfitting and excels with imbalanced data. XGBoost and Gradient Boosting optimize complex decision boundaries through sequential error correction, ideal for generalization. LightGBM prioritizes efficiency and scalability for large datasets. Together, they enable a balanced comparison of linear, tree-based, and gradient-boosted approaches across WP-SDP and CP-SDP scenarios.

D. Dataset

Performance metrics offer quantifiable measures to evaluate how well software defect prediction models perform in classification tasks. To achieve an objective assessment, it is advisable to use a combination of widely accepted metrics, such as accuracy, rather than relying on a single indicator. Accuracy is one of the most commonly understood evaluation metrics, indicating the ratio of correctly predicted instances to the overall number of instances in the dataset [65]. The formula used is in Eq. (4).

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN} \tag{4}$$

Among these metrics, AUC (Area Under the Curve) stands out as a robust measure of a model's ability to distinguish between defect and non-defect classes independent of any threshold. AUC reflects the probability that a defective module will be scored higher than a non-defective one, providing a nuanced view of the model's classification performance even in the presence of imbalanced or noisy datasets. Recent research [66] has demonstrated that AUC is a reliable indicator in evaluating the effectiveness of software defect prediction models, reinforcing its role in ensuring a comprehensive and objective evaluation. The formula used is shown in Eq. (5).

$$AUC = \frac{\left(\frac{TP}{TP + FN}\right) + \left(\frac{TN}{TN + FP}\right)}{2} \tag{5}$$

3. Results

This study evaluates the performance of five classification algorithms Random Forest (RF), Support Vector Machine (SVM), XGBoost (XGB), Gradient Boosting (GB), and LightGBM (LGBM) on two software defect prediction (SDP) schemes: Within Project (WP-SDP) and Cross Project (CP-SDP). To address the class imbalance, three resampling strategies (no sampling, undersampling, and oversampling) are applied. The dataset is divided into a training-testing ratio of 70:30 to ensure a robust evaluation of generalization ability. Performance metrics, including Accuracy and AUC, are measured to compare the effectiveness of each configuration. Notably, the study produced up to 360 distinct sets of AUC and accuracy data, reflecting a variety of outcomes across different experimental settings.

Experiments on 12 NASA MDP projects show performance variations between Cross-Project Software Defect Prediction (CP-SDP) and Within-Project Software Defect Prediction (WP-SDP) schemes as shown in Table 2. Overall, WP-SDP dominated by excelling in 8 out of 12 projects, while CP-SDP showed effectiveness in 4 projects, especially in small-sized projects such as MW1 and PC1. The combination of resampling and tree-based classification techniques (such as Random Forest) proved to be the most consistent in delivering high performance.

The absolute best configuration was achieved by WP-SDP on project PC4 with oversampling and Random Forest classification, resulting in an Accuracy of 89.92% and an AUC of 0.931. The scheme also excelled on projects PC2 (AUC 0.917) and MC1 (AUC 0.908),

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

demonstrating its ability to handle class imbalance when project historical data is sufficient. On the other hand, CPDP recorded the highest result in MW1 (AUC 0.929 with no sampling and Random Forest), indicating that cross-project prediction can be competitive if the characteristics of the source and target projects are homogeneous.

Table 2. Comparison of the best performance between the WP-SDP and CP-SDP methods per class on the NASA MDP dataset highlighting the performance advantages of each technique in handling each class of data.

Project	Method	Sampling	Classifier	ACC (%)	AUC
CM1	CP-SDP	No	RF	84.85	0.801
JM1	WP-SDP	Over	GB	70.11	0.691
KC1	WP-SDP	Under	GB	67.89	0.719
KC3	CP-SDP	Over	LGBM	76.27	0.762
MC1	WP-SDP	Under	GB	78.22	0.908
MC2	WP-SDP	Under	GB	68.42	0.833
MW1	CP-SDP	No	RF	89.47	0.929
PC1	CP-SDP	Over	LGBM	80.19	0.862
PC2	WP-SDP	Under	RF	78.13	0.917
PC3	WP-SDP	Under	LGBM	75	0.795
PC4	WP-SDP	Over	RF	89.92	0.931
PC5	WP-SDP	Over	RF	74.32	0.754

The oversampling technique significantly improves SDP performance, especially on large projects such as PC4 (+0.15 AUC compared to no sampling). However, undersampling is more effective for projects with moderate class imbalance such as MC1 (AUC 0.908) and PC2 (AUC 0.917). Meanwhile, CPDP achieved optimal performance without resampling techniques (no sampling) in MW1 and CM1, suggesting that data reduction or augmentation may remove important patterns across projects.

Random Forest was the most reliable algorithm, topping the rankings in 6 projects (e.g. PC4, MW1, CM1) with an average AUC of 0.893. Boosting-based classifiers such as LightGBM and Gradient Boosting showed mixed results: LightGBM excelled in PC1 (0.862 AUC with CPDP), while Gradient Boosting had the lowest performance in JM1 (0.691 AUC) due to its sensitivity to data noise.

Project JM1 recorded the worst result (AUC 0.691 with SDP + oversampling + Gradient Boosting), presumably due to the complexity of the code and the poorly represented feature distribution in the training data. On the other hand, KC1 and KC3 (AUC 0.719 and 0.762) show that the combination of SDP/CPDP with undersampling or oversampling can moderately overcome class imbalance.

We present paired t-test results for Random Forest on just two projects PC4 under WP-SDP (Table 3) and MW1 under CP-SDP (Table 4) selected because they achieved the highest performance in their respective schemes and thus serve as clear, representative examples. In each project, the comparison between no sampling and oversampling shows no significant difference in Accuracy or AUC, confirming that duplicating minority instances does not materially alter RF's performance when ample data is available (PC4) or when source and target datasets share strong homogeneity (MW1). By contrast, both no sampling versus undersampling and oversampling versus undersampling exhibit significant declines in both metrics ($p < 0.05$), underscoring that removing majority-class examples impairs the model's ability to capture defect patterns. These results reinforce our recommendation to favor oversampling over undersampling for Random Forest in high-data scenarios and to rely on unsampled CP-SDP when project characteristics are closely aligned, demonstrating that our findings remain consistent across multiple runs.

Table 3. Paired t-test results of project PC4 representing WP-SDP

Comparison	t-stat (ACC)	p-value (ACC)	Sig. ACC	t-stat (AUC)	p-value (AUC)	Sig. AUC
No vs Over	1.24	0.249	No	-0.15	0.887	No
No vs Under	2.84	0.02	Yes	3.21	0.012	Yes
Over vs Under	3.55	0.006	Yes	3.87	0.004	Yes

Table 4. Paired t-test results of project MW1 representing CP-SDP

Comparison	t-stat (ACC)	p-value (ACC)	Sig. ACC	t-stat (AUC)	p-value (AUC)	Sig. AUC
No vs Over	1.24	0.249	No	−0.15	0.887	No
No vs Under	2.84	0.02	Yes	3.21	0.012	Yes

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

Over vs Under	3.55	0.006	Yes	3.87	0.004	Yes

4. Discussion

The results show that the performance of software defect prediction is highly dependent on the selection of schemes, resampling strategies, and classification algorithms. WP-SDP tends to excel in most cases, especially when oversampling is combined with an ensemble algorithm such as Random Forest, resulting in an AUC of 0.931 on one major project. On the other hand, CP-SDP also showed superiority on certain projects, especially on a small project such as MW1 which obtained an AUC of 0.929 without the application of resampling. This finding confirms that local data patterns are a key factor affecting model accuracy and allows WP-SDP to show consistent results on projects with adequate data representation. In general, WP-SDP benefits from richer, project-specific histories its models learn nuanced defect patterns when ample examples are available whereas CP-SDP shines on smaller datasets by leveraging broader experience from other projects to compensate for limited local observations. Overall, an in-depth analysis of this experiment revealed that the method configuration and dataset characteristics must be adjusted to achieve optimal prediction results.

In the study by N. A. Bhat and S. U. Farooq (2023), they found that local data has an important role in maintaining defect patterns, which aligns with our WP-SDP superiority. Meanwhile, in the study by Zhu et al. (2020), the transfer learning method with feature weighting improves the effectiveness of CP-SDP on projects with homogeneous characteristics, which is in line with the results that show CP-SDP excels in MW1. In addition, research by T. Li, Z. Wang, and P. Shi (2025) proposed model fusion to optimize prediction, which supports our findings regarding the reliability of the Random Forest algorithm in reducing noise and handling non-linear features. A comparison of these three studies shows that although the approaches used are different, they all emphasize the importance of tailoring the method to the characteristics of the available data. This indicates that collaboration between local techniques and across projects can open up opportunities to combine the strengths of each approach in creating more adaptive defect prediction systems.

This research has several limitations that need to be considered for the interpretation of the results and further development of the method. First, the number of projects used is limited to the NASA MDP dataset, so the results may not be generalizable to all types of software projects. To address this, future work should include diverse repositories such as GitHub or JIRA and assess transferability across domains. Second, the hyperparameter configuration for the classification algorithm has not been optimized in depth, which could potentially affect performance, especially in models such as Gradient Boosting that show sensitivity to noise. A systematic tuning campaign or automated search could mitigate this gap. Third, the application of oversampling and undersampling techniques in CP-SDP schemes sometimes results in bias, as not all projects require data manipulation to achieve prediction stability. Adaptive or hybrid sampling approaches that respond to each project's imbalance level may overcome this issue. Fourth, the comparison between WP-SDP and CP-SDP has not been accompanied by an in-depth quantitative analysis of the similarity of characteristics between projects so that adjustments to the method can be made more systematically. Incorporating metrics of project similarity or domain distance could guide better method selection.

The research has broad implications for practitioners and researchers in the field of software engineering. The results show that the selection of prediction methods must carefully consider local data characteristics and uniformity between projects so that WP-SDP can be prioritized on projects with data that represents the actual conditions of development. For developers, the use of oversampling and ensemble algorithms such as Random Forest is an effective solution to overcome the problem of class imbalance, especially in large-scale projects. The findings also suggest that different resampling strategies can be applied based on the level of data imbalance, such as undersampling for cases with moderate imbalance. Furthermore, the findings from this study open up opportunities for the integration of transfer learning and model averaging approaches to improve prediction efficiency and accuracy in more diverse industrial applications. In a real-world setting, teams with extensive historical logs could default to WP-SDP, while those launching new or smaller codebases might adopt CP-SDP to bootstrap their prediction pipelines; our framework thus serves as a practical guide for method selection in live projects.

5. Conclusion

This study aims to compare the effectiveness of Cross-Project (CP-SDP) and Within-Project (WP-SDP) defect prediction schemes using the NASA MDP dataset, considering resampling techniques (undersampling, oversampling, no sampling) and the performance of five classifiers. Results show that WP-SDP excels in 8 out of 12 projects, with the highest performance in project PC4 (AUC 0.931, Accuracy 89.92%) using oversampling and Random Forest. Meanwhile, CP-SDP was effective in homogeneous projects such as MW1 (AUC 0.929, Accuracy 89.47%) with no sampling and Random Forest. The oversampling technique improves the performance of WP-SDP by 15%, while undersampling is optimal for projects with moderate class imbalance. Random Forest was the best classifier (excelling in 6 projects), while Gradient Boosting was susceptible to data noise (lowest AUC 0.691 in JM1).

Further research is recommended to test the generalizability of these findings to non-NASA datasets (e.g. GitHub/JIRA) and integrate cost-sensitive metrics to reflect the risk of false negatives in the real world. Building

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

on our observation that CP-SDP often struggles with distribution shifts while WP-SDP falters when data is scarce, future work should explore transfer learning strategies that can adapt models trained on mature projects to emerging ones, effectively bridging the gap between inter and intra project data. Likewise, hybrid sampling techniques which intelligently combine oversampling of minority cases with targeted undersampling of noisy majority instances could mitigate class imbalance more robustly than single-method schemes, as our results showed oversampling increased recall but sometimes inflated false positives, and undersampling reduced training cost at the expense of lost information. Incorporating model interpretability analyses such as SHAP values will help reveal which features drive predictions across both WP and CP settings, guiding more transparent defect prediction pipelines. The findings provide practical guidance for developers in selecting defect prediction schemes according to historical data availability, project characteristics, and algorithm complexity, so that software quality optimization can be performed more efficiently.

References

[1] Li F, Yang P, Keung J W, et al. Revisiting ‘revisiting supervised methods for effort-aware cross-project defect prediction’[J]. IET Software, 2023,17(4): 472-495.

[2] Zou Q, Lu L, Qiu S, et al. Correlation feature and instance weights transfer learning for cross project software defect prediction[J]. IETSoftware, 2021, 15(1): 55-74.

[3] A. Perera, A. Aleti, M. Böhme, and B. Turhan, "Defect prediction guided search-based software testing," in Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), Virtual Event, Australia, 2020, pp. 448–460, doi: 10.1145/3324884.3416612.

[4] S. Stradowski and L. Madeyski, "Costs and benefits of machine learning software defect prediction: Industrial case study," in Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE 2024), New York, NY, USA: Association for Computing Machinery, 2024, pp. 92–103. doi: 10.1145/3663529.3663831.

[5] S. Haldar and L. F. Capretz, "Interpretable software defect prediction from project effort and static code metrics," Computers, vol. 13, no. 2, p. 52, 2024. doi: 10.3390/computers13020052.

[6] S. Haldar and L. F. Capretz, "Explainable software defect prediction from cross company project metrics using machine learning," in Proc. 7th Int. Conf. Intell. Comput. Control Syst. (ICICCS), May 2023, pp. 150–157, doi: 10.1109/ICICCS56967.2023.10142534.

[7] M. Nevendra and P. Singh, "Cross-Project Defect Prediction with Metrics Selection and Balancing Approach," Appl. Comput. Syst., vol. 27, no. 2, pp. 137–148, 2022, doi: 10.2478/acss-2022-0015.

[8] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software defect prediction analysis using machine learning techniques," Sustainability, vol. 15, no. 6, p. 5517, 2023, doi: 10.3390/su15065517.

[9] Balogun, A.O.; Basri, S.; Mahamad, S.; Abdulkadir, S.J.; Capretz, L.F.; Imam, A.A.; Almomani, M.A.; Adeyemo, V.E.; Kumar, G. Empirical analysis of rank aggregation-based multi-filter feature selection methods in software defect prediction. Electronics 2021, 10, 179.

[10] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Y. Ghadi, and M. Amir Khan, "Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning," PeerJ Computer Science, vol. 10, p. e1860, 2024, doi: 10.7717/peerj-cs.1860.

[11] M. Mukelabai, M. Mukelabai, S. Strüder, D. Strüber, and T. Berger, "Feature-oriented defect prediction: Scenarios, metrics, and classifiers," arXiv preprint arXiv:2104.06161, 2021. doi: 10.48550/arXiv.2104.06161.

[12] D. Chen, X. Chen, H. Li, J. Xie, and Y. Mu, "DeepCPDP: Deep learning based cross-project defect prediction," IEEE Access, vol. 7, pp. 184832–184848, 2019. doi: 10.1109/ACCESS.2019.2961129.

[13] A. Abdu, Z. Zhai, H. A. Abdo, S. Lee, M. A. Al-masni, Y. H. Gu, and R. Algabri, "Cross-project software defect prediction based on the reduction and hybridization of software metrics," *Alexandria Eng. J.*, vol. 112, pp. 161–176, 2025, doi: 10.1016/j.aej.2024.10.034.

[14] J. Xian, J. Li, Q. Zou, and Y. Xian, "LPDA: Cross-project software defect prediction approach via locality preserving and distribution alignment," International Journal of Advanced Computer Science and Applications, vol. 14, no. 12, 2023. doi: 10.14569/IJACSA.2023.0141290.

[15] R. Vashisht and S. A. Rizvi, "Addressing noise and class imbalance problems in heterogeneous cross-project defect prediction: An empirical study," International Journal of e-Collaboration (IJeC), vol. 19, no. 1, pp. 1-27, 2023. doi: 10.4018/IJeC.315777.

[16] K. E. Bennin, A. Tahir, S. G. MacDonell, and J. Börstler, "An empirical study on the effectiveness of data resampling approaches for cross-project software defect prediction," IET Software, vol. 16, no. 2, pp. 185–199, Mar. 2022. doi: 10.1049/sfw2.12052.

[17] R. M. Pereira, Y. M. G. Costa, and C. N. Silla Jr., "Toward hierarchical classification of imbalanced data using random resampling algorithms," Information Sciences, vol. 578, pp. 344-363, 2021. doi: 10.1016/j.ins.2021.07.033.

[18] A. Tripathi, R. Chakraborty, and S. K. Kopparapu, "A novel adaptive minority oversampling technique

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

for improved classification in data imbalanced scenarios," arXiv preprint, arXiv:2103.13823, 2021.

[19] A. B. Hassanat, A. S. Tarawneh, G. A. Altarawneh, and A. Almuhaimeed, "Stop oversampling for class imbalance learning: A critical review," arXiv preprint, arXiv:2202.03579, 2022.

[20] P. Sadhukhan, A. Pakrashi, and B. Mac Namee, "Random Walk-steered Majority Undersampling," in 2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 530-537, Oct. 2022. doi: 10.1109/SMC53654.2022.9945351.

[21] W. Zhou, C. Liu, P. Yuan, and L. Jiang, "An undersampling method approaching the ideal classification boundary for imbalance problems," Applied Sciences, vol. 14, no. 13, p. 5421, 2024. doi: 10.3390/app14135421.

[22] D. Hsu, V. Muthukumar, and J. Xu, "On the proliferation of support vectors in high dimensions," in International Conference on Artificial Intelligence and Statistics, pp. 91-99, Mar. 2021. PMLR.

[23] R. Guido, S. Ferrisi, D. Lofaro, and D. Conforti, "An overview on the advancements of support vector machine models in healthcare applications: A review," Information, vol. 15, no. 4, p. 235, 2024. doi: 10.3390/info15040235.

[24] H. Dabiri, V. Farhangi, M. J. Moradi, M. Zadehmohamad, and M. Karakouzian, "Applications of decision tree and random forest as tree-based machine learning techniques for analyzing the ultimate strain of spliced and non-spliced reinforcement bars," Applied Sciences, vol. 12, no. 10, p. 4851, 2022. doi: 10.3390/app12104851.

[25] L. Barreñada, P. Dhiman, D. Timmerman, et al., "Understanding overfitting in random forest for probability estimation: a visualization and simulation study," Diagn Progn Res, vol. 8, p. 14, 2024. doi: 10.1186/s41512-024-00177-1.

[26] N. S. Thomas and S. Kaliraj, "An improved and optimized random forest based approach to predict the software faults," SN Computer Science, vol. 5, p. 530, 2024. doi: 10.1007/s42979-024-02764-x.

[27] X. Ju and M. Salibián-Barrera, "Robust boosting for regression problems," Computational Statistics & Data Analysis, vol. 153, p. 107065, 2021. doi: 10.1016/j.csda.2020.107065.

[28] Z. Zhang, Y. Zhao, A. Canes, D. Steinberg, and O. Lyashevska, "Predictive analytics with gradient boosting in clinical medicine," Annals of Translational Medicine, vol. 7, no. 7, 2019. doi: 10.21037/atm.2019.03.29.

[29] M. Nasir, N. S. Summerfield, S. Simsek, and A. Oztekin, "An interpretable machine learning methodology to generate interaction effect hypotheses from complex datasets," Decision Sciences, vol. 55, pp. 549–576, 2024. doi: 10.1111/deci.12642.

[30] S. Fatima, A. Hussain, S. B. Amir, S. H. Ahmed, and S. M. H. Aslam, "XGBoost and random forest algorithms: An in-depth analysis," Pakistan Journal of Scientific Research, vol. 3, no. 1, pp. 26–31, 2023. doi: 10.57041/vol3iss1pp26-31.

[31] B. Kumar and T. Kumar, "Comparative analysis of ML-based gradient boosting algorithms: XGBoost, CatBoost, and LightGBM," Journal of Scientific and Engineering Research, vol. 7, no. 8, pp. 235–239, 2020. doi: 10.5281/zenodo.12666689.

[32] J. Yan, Y. Xu, Q. Cheng, et al., "LightGBM: accelerated genomically designed crop breeding through ensemble learning," Genome Biology, vol. 22, p. 271, 2021. doi: 10.1186/s13059-021-02492-y.

[33] G. Varoquaux and O. Colliot, "Evaluating machine learning models and their diagnostic value," in Machine Learning for Brain Disorders, O. Colliot, Ed. Neuromethods, vol. 197, New York, NY: Humana, 2023. doi: 10.1007/978-1-0716-3195-9_20.

[34] S. Adhikari, S.-L. Normand, J. Bloom, D. Shahian, and S. Rose, "Revisiting performance metrics for prediction with rare outcomes," Statistical Methods in Medical Research, vol. 30, no. 10, pp. 2352-2366, 2021. doi: 10.1177/09622802211038754.

[35] P. Sasankar and G. Sakarkar, "An empirical study of classification models using AUC-ROC curve for software fault predictions," International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), vol. 9, no. 1, pp. 250-260, Jan.-Feb. 2023. doi: 10.32628/CSEIT2390143.

[36] N. A. Bhat and S. U. Farooq, "An empirical evaluation of defect prediction approaches in within-project and cross-project context," Software Quality Journal, vol. 31, pp. 917–946, 2023. doi: 10.1007/s11219-023-09615-7.

[37] K. Zhu, N. Zhang, S. Ying, and X. Wang, "Within-Project and Cross-Project Software Defect Prediction Based on Improved Transfer Naive Bayes Algorithm," Comput. Mater. Contin., vol. 63, no. 2, pp. 891–910, 2020. doi: 10.32604/cmc.2020.08096.

[38] T. Li, Z. Wang, and P. Shi, "Within-project and cross-project defect prediction based on model averaging," Scientific Reports, vol. 15, p. 6390, 2025. doi: 10.1038/s41598-025-90832-4.

[39] "NASADefectDataset," Github. Accessed: Jan. 6, 2025. [Online]. Available: https://github.com/klainfo/NASADefectDataset

[40] T. Sasada, Z. Liu, T. Baba, K. Hatano, and Y. Kimura, "A resampling method for imbalanced datasets considering noise and overlap," Procedia Computer Science, vol. 176, pp. 420–429, 2020. doi: 10.1016/j.procs.2020.08.043.

[41] B. J. Odejide et al., "An empirical study on data sampling methods in addressing class imbalance problem in software defect prediction," in Software Engineering Perspectives in Systems. CSOC 2022,

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

R. Silhavy, Ed., Lecture Notes in Networks and Systems, vol. 501, 2022, doi: 10.1007/978-3-031-09070-7_49.

[42] R. Wang, F. Liu, and Y. Bai, "A software defect prediction method that simultaneously addresses class overlap and noise issues after oversampling," Electronics, vol. 13, no. 20, p. 3976, 2024. doi: 10.3390/electronics13203976.

[43] N. A. A. Khleel and K. Nehéz, "Software defect prediction using a bidirectional LSTM network combined with oversampling techniques," Cluster Computing, vol. 27, pp. 3615–3638, 2024. doi: 10.1007/s10586-023-04170-z.

[44] A. O. Balogun et al., "Empirical analysis of data sampling-based ensemble methods in software defect prediction," in Computational Science and Its Applications – ICCSA 2022 Workshops. ICCSA 2022, O. Gervasi, B. Murgante, S. Misra, A. M. A. C. Rocha, and C. Garau, Eds., Lecture Notes in Computer Science, vol. 13381, Cham: Springer, 2022, doi: 10.1007/978-3-031-10548-7_27.

[45] R. Malhotra and J. Jain, "Predicting defects in imbalanced data using resampling methods: an empirical investigation," PeerJ Computer Science, vol. 8, e573, 2022. doi: 10.7717/peerj-cs.573.

[46] S. Bagui and K. Li, "Resampling imbalanced data for network intrusion detection datasets," Journal of Big Data, vol. 8, p. 6, 2021. doi: 10.1186/s40537-020-00390-x.

[47] V. R. Joseph, "Optimal ratio for data splitting," Statistical Analysis and Data Mining: The ASA Data Science Journal, vol. 15, pp. 531–538, 2022. doi: 10.1002/sam.11583.

[48] S. Steinert, V. Ruf, D. Dzsotjan, N. Großmann, A. Schmidt, J. Kuhn, and S. Küchemann, "A refined approach for evaluating small datasets via binary classification using machine learning," PLoS One, vol. 19, no. 5, p. e0301276, May 2024, doi: 10.1371/journal.pone.0301276.

[49] K.-L. Du, B. Jiang, J. Lu, J. Hua, and M. N. S. Swamy, "Exploring kernel machines and support vector machines: Principles, techniques, and future directions," Mathematics, vol. 12, no. 24, p. 3935, 2024, doi: 10.3390/math12243935.

[50] M. Mustaqeem and M. Saqib, "Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection," Cluster Computing, vol. 24, pp. 2581–2595, 2021. doi: 10.1007/s10586-021-03282-8.

[51] H. Abu Alhija, M. Azzeh, and F. Almasalha, "Software defect prediction using support vector machine," arXiv preprint arXiv:2209.14299, 2022, doi: 10.48550/arXiv.2209.14299.

[52] N. S. Thomas and S. Kaliraj, "An improved and optimized random forest based approach to predict the software faults," SN Computing Science, vol. 5, p. 530, 2024, doi: 10.1007/s42979-024-02764-x.

[53] B. Khan, R. Naseem, M. A. Shah, K. Wakil, A. Khan, M. I. Uddin, and M. Mahmoud, "Software defect prediction for healthcare big data: An empirical evaluation of machine learning techniques," Journal of Healthcare Engineering, vol. 2021, Article ID 8899263, 2021, doi: 10.1155/2021/8899263.

[54] D. C. E. Saputra, Y. Maulana, T. A. Win, R. Phann, and W. Caesarendra, "Implementation of machine learning and deep learning models based on structural MRI for identification autism spectrum disorder," Jurnal Ilmiah Teknik Elektro, Komputer, dan Informatika (JITEKI), vol. 9, no. 2, pp. 307–318, May 2023. doi: 10.26555/jiteki.v9i2.26094.

[55] M. Schonlau and R. Y. Zou, "The random forest algorithm for statistical learning," tata Journal, vol. 20, no. 1, pp. 3–29, 2020. doi: 10.1177/1536867X20909688.

[56] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of XGBoost," arXiv, 2019. doi: 10.48550/arXiv.1911.01914.

[57] Tariq, N., & Hasoon, S. (2024). Software defect prediction using extreme gradient boosting (XGBoost) with optimization hyperparameter. Computer Science and Mathematical Journal, 18, 22–29. doi: 10.33899/CSMJ.2023.142739.108

[58] Y. Al-Smadi, M. Eshtay, A. Al-Qerem, S. Nashwan, O. Ouda, and A. A. Abd El-Aziz, "Reliable prediction of software defects using Shapley interpretable machine learning models," Egyptian Informatics Journal, vol. 24, no. 3, Article ID 100386, 2023. doi: 10.1016/j.eij.2023.05.011.

[59] R. U. Aflaha, R. Herteno, M. R. Faisal, F. Abadi, and S. W. Saputro, "Effect of SMOTE variants on software defect prediction classification based on boosting algorithm," Jurnal Ilmiah Teknik Elektro, Komputer, dan Informatika (JITEKI), vol. 10, no. 2, pp. 201–216, 2024. doi: 10.26555/jiteki.v10i2.2852.

[60] A. Alazba and H. Aljamaan, "Software defect prediction using stacking generalization of optimized tree-based ensembles," Applied Sciences, vol. 12, no. 9, p. 4577, 2022. doi: 10.3390/app12094577.

[61] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," Artificial Intelligence Review, vol. 54, pp. 1937–1967, 2021. doi: 10.1007/s10462-020-09896-5.

[62] A. M. Akbar, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "Optimizing software defect prediction models: Integrating hybrid grey wolf and particle swarm optimization for enhanced feature selection with popular gradient boosting algorithm," Journal of Electronic & Electromedical Engineering & Medical Informatics, vol. 6, no. 2, pp. 169–181, Apr. 2024. doi: 10.35882/jeeemi.v6i2.388.

[63] S. Li, X. Dong, D. Ma, B. Dang, H. Zang, and Y. Gong, "Utilizing the LightGBM algorithm for operator user credit assessment research," arXiv preprint arXiv:2403.14483, 2024.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

[64] S. Li, N. Jin, A. Dogani, Y. Yang, M. Zhang, and X. Gu, "Enhancing LightGBM for industrial fault warning: An innovative hybrid algorithm," Processes, vol. 12, no. 1, p. 221, 2024. doi: 10.3390/pr12010221.

[65] Y. Zhao, Z. Huang, L. Gong, Y. Zhu, Q. Yu, and Y. Gao, "Evaluating the impact of data transformation techniques on the performance and interpretability of software defect prediction models," IET Software, Article ID 6293074, 30 pages, 2023. doi: 10.1049/2023/6293074.

[66] H. Shi, J. Ai, J. Liu, and J. Xu, "Improving software defect prediction in noisy imbalanced datasets," Applied Sciences, vol. 13, no. 18, p. 10466, 2023, doi: 10.3390/app131810466.

AUTHOR BIOGRAPHY

Raidra Zenianantо, a student at Lambung Mangkurat University with Computer Science study program, faculty of Mathematics and Natural Sciences. he has a strong interest in software engineering, particularly mobile-based application development. With his perseverance and passion for learning, Raidra strives to hone his technical skills through various projects and research. Through his dedication and innovative mindset, he seeks not only to excel in his studies but also to make a significant impact in the field of mobile technology and software development. Additionally, he devotes himself to exploring emerging technologies and refining his practical abilities by working on challenging assignments that bridge theoretical insights with real-world applications. Email: raidrazeni@gmail.com.

Rudy Herteno received his bachelor's degree in Computer Science from Lambung Mangkurat University in 2011. After completing his studies, he worked as a software developer for several years to gain more experience in the field. During this period, he developed various software applications, particularly to support the needs of local governments. In 2017, he obtained a master's degree in Informatics from STMIK Amikom University. Currently, he is a lecturer in the Computer Science program at Lambung Mangkurat University. His research interests include software engineering, software defect prediction, and deep learning, aiming to improve software quality, optimize error detection in systems, and develop artificial intelligence-based solutions. He can be contacted at email: rudy.herteno@ulm.ac.id.

Radityo Adi Nugroho received his bachelor's degree in Informatics from the Islamic University of Indonesia and a master's degree in Computer Science from Gadjah Mada University. Currently, he is an assistant professor in the Department of Computer Science at Lambung Mangkurat University. His research interests include software defect prediction and computer vision. He can be contacted at email: radityo.adi@ulm.ac.id.

Andi Farmadi, a senior lecturer in the Computern Science program at Lambung Mangkurat University. He has been teaching since 2008 and currently serves as the Head of the Data Science Lab since 2018. He completed his undergraduate studies at Hasanuddin University and his graduate studies at Bandung Institute of Technology. His research area, up to the present, focuses on Data Science. One of his research projects, along with other researchers, published in the International Conference of Computer and Informatics Engineering (IC2IE), is titled "Hyperparameter tuning using GridsearchCV on the comparison of the activation function of the ELM method to the classification of pneumonia in toddlers," and this research was published in 2021. Email: andifarmadi@ulm.ac.id.

Setyo Wahyu Saputro, is a lecturer in Computer Science Department, Faculty of Mathematics and Natural Science, Lambung Mangkurat University in Banjarbaru. He received bachelor's degree also in Computer Science from Lambung Mangkurat University in 2011, and received his master's degree in Informatics from STMIK Amikom University in 2016. He is active as an information technology practitioner and consultant, being a project manager or systems analyst working on several projects in government and private agencies in South Kalimantan province since 2017. His research interests include software engineering, human computer interaction, and artificial intelligence applications. He can be contacted at email: setyo.saputro@ulm.ac.id.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
Digital Object Identifier (DOI): https://doi.org/10.35882/ijeeemi.v7i3.95
Copyright © 2025 by the authors. Published by Jurusan Teknik Elektromedik, Politeknik Kesehatan Kemenkes Surabaya Indonesia. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).