Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871

Vol. 6, No. 6, December 2025, Page. 5821-5836
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5393

Comparative Performance Evaluation of Linear, Bagging, and Boosting Models Using BorutaSHAP for Software Defect Prediction on NASA MDP Datasets

Najla Putri Kartika[1], Rudy Herteno*[2], Irwan Budiman[3], Dodon Turianto Nugrahadi[4], Friska Abadi[5], Umar Ali Ahmad[6], Mohammad Reza Faisal[7]

[1,2,3,4,5,7]Computer Science, Lambung Mangkurat University, Banjarbaru, Indonesia
[6]School of Electrical Engineering, Telkom University, Bandung, Indonesia
[6]Collaborative Researcher, Kanazawa University, Kanazawa, Ishikawa, Japan

Email: [2]rudy.herteno@ulm.ac.id

Received : Oct 27, 2025; Revised : Nov 13, 2025; Accepted : Nov 17, 2025; Published : Dec 23, 2025

Abstract

Software defect prediction aims to identify potentially defective modules early on in order to improve software reliability and reduce maintenance costs. However, challenges such as high feature dimensions, irrelevant metrics, and class imbalance often reduce the performance of prediction models. This research aims to compare the performance of three classification model groups—linear, bagging, and boosting—combined with the BorutaSHAP feature selection method to improve prediction stability and interpretability. A total of twelve datasets from the NASA Metrics Data Program (MDP) were used as test references. The research stages included data preprocessing, class balancing using the Synthetic Minority Oversampling Technique (SMOTE), feature selection with BorutaSHAP, and model training using five algorithms, namely Logistic Regression, Linear SVC, Random Forest, Extra Trees, and XGBoost. The evaluation was conducted with Stratified 5-Fold Cross-Validation using the F1-score and Area Under the Curve (AUC) metrics. The experimental results showed that tree-based ensemble models provided the most consistent performance, with Extra Trees recording the highest average AUC of 0.794 ± 0.05, followed by Random Forest (0.783 ± 0.06). The XGBoost model provided the best results on the PC4 dataset (AUC = 0.937 ± 0.008), demonstrating its ability to handle complex data patterns. These findings prove that BorutaSHAP is effective in filtering relevant features, improving classification reliability, and strengthening transparency and interpretability in the Explainable Artificial Intelligence (XAI) framework for software quality improvement.

Keywords : *BorutaSHAP, Feature Selection, Machine Learning Ensembles, SMOTE, Software Defect Prediction, XAI.*

1. INTRODUCTION

Software Defect Prediction (SDP) is an important field in software engineering that aims to identify software modules that potentially contain defects before the testing or implementation stage [1], [2], [3]. With the increasing complexity of systems and volume of data, the ability to accurately predict software defects not only impacts quality improvement, but also cost and development time efficiency [4], [5], [6], [7]. Recent research emphasizes that the main challenges in SDP are data imbalance and feature redundancy, which can reduce the performance of prediction models [6], [8], [9].

The most commonly used dataset for testing SDP models is the NASA Metrics Data Program (NASA-MDP) because it has a variety of projects with different code metric characteristics, sizes, and complexities [1], [10]. However, this dataset has an unbalanced class distribution—the number of non-defect modules is much greater than the number of defect modules—which causes the model to be biased towards the majority class [11], [12], [13]. To overcome this, several resampling approaches such as SMOTE (Synthetic Minority Oversampling Technique) are widely used to improve the model's sensitivity to the minority class [9], [11], [12], [14].

In addition to data balancing, the feature selection process plays an important role in improving the performance of prediction models. Irrelevant or excessive features can cause overfitting and reduce the model's generalization ability [15], [16], [17]. One of the most promising feature selection methods is BorutaSHAP, which combines the Boruta wrapper approach with the explainability of SHAP (Shapley Additive Explanations) [16], [18], [19]. This approach enables the identification of features that truly contribute to predictions with transparent interpretations, thereby supporting the application of Explainable Artificial Intelligence (XAI) in the domain of software engineering [20], [21], [22].

Previous studies have applied various machine learning algorithms such as Logistic Regression, Random Forest, Extra Trees, and XGBoost to SDP. Ensemble models (bagging and boosting) often show better performance than linear models in handling non-linear data and diverse distributions [22], [23], [24], [25]. However, most studies have not yet systematically explored the application of BorutaSHAP in combination with balancing methods such as SMOTE and comparisons between model categories (linear, bagging, boosting) using the NASA-MDP dataset. This gap is the main research gap in this research. To clarify the position of this research in the context of previous studies, Table 1 summarizes several relevant studies related to SDP, XAI, and BorutaSHAP.

Table 1. Prior studies on SDP

Researcher	Title	Feature Selection	Classification	Main Results
Yue (2024)	Screening of lung cancer serum biomarkers based on Boruta-SHAP and RFC-RFECV algorithms (*J Proteomics*)	BorutaSHAP	NB, SVC	Mean AUC ≈ 0.88; valid AUC NB = 0.93, SVC = 0.94 [18]
Al-Smadi et al. (2023)	Reliable prediction of software defects using Shapley interpretable machine learning models (*Egyptian Informatics Journal*)	SHAP	Ensembles (11 algorithms)	ROC-AUC > 0.90 (multi-model, interpretability improved) [12]
Albattah & Alzahrani (2024)	Software Defect Prediction Based on ML and DL Techniques (*AI Switzerland*)	–	Linear, Bagging, Boosting, DL	Accuracy ≈ 0.87 (AUC not reported) [23]
Mustaqeem et al. (2024)	A trustworthy hybrid model for transparent software defect prediction: SPAM-XAI (*PLOS ONE*)	Boruta + SHAP	RF, Bagging	AU-ROC = 0.91 (CM1), 0.79 (PC1); F1 ≈ 0.82 [8]
Živković et al. (2023)	Software defects prediction by metaheuristics tuned XGBoost and analysis based on SHAP (*Appl Soft Comput*)	SHAP	XGBoost	Accuracy > 90%; AUC ≈ 0.95; F1 ≈ 0.90 [20]

The table shows that although a number of studies have applied explainable approaches such as SHAP and ensemble methods, no study has comprehensively combined BorutaSHAP with SMOTE and compared the performance of various model categories (Linear–Bagging–Boosting) across the entire NASA-MDP dataset.

Therefore, this research aims to apply BorutaSHAP as a feature selection method on the NASA-MDP dataset, combined with the SMOTE technique to overcome data imbalance. Furthermore, this research compares the performance of three categories of models, namely linear models (Logistic Regression and Linear SVC), bagging models (Random Forest and Extra Trees), and boosting models (XGBoost), using the F1-score and AUC evaluation metrics. Through this approach, this research is expected to make a real contribution to the development of a more accurate, stable, and explainable software defect prediction framework (XAI-based SDP framework) in accordance with the latest research standards in the field of Informatics and Software Engineering [2], [12], [20], [21].

2. METHOD

This research uses a computational experimental approach utilizing the NASA Metrics Data Program (MDP) dataset, which is widely used in software defect prediction (SDP) studies. This dataset was chosen because it contains various complex and varied software metrics, but presents challenges in the form of high feature dimensions, irrelevant features, and class imbalance between defective and non-defective modules [5], [20], [24]. The research stages include six main steps, covering data collection, data preprocessing, data balancing, feature selection using BorutaSHAP, model classification, and performance evaluation. The entire process is visualized in Figure 1, which shows the research pipeline flow from the data processing stage to the final model evaluation.

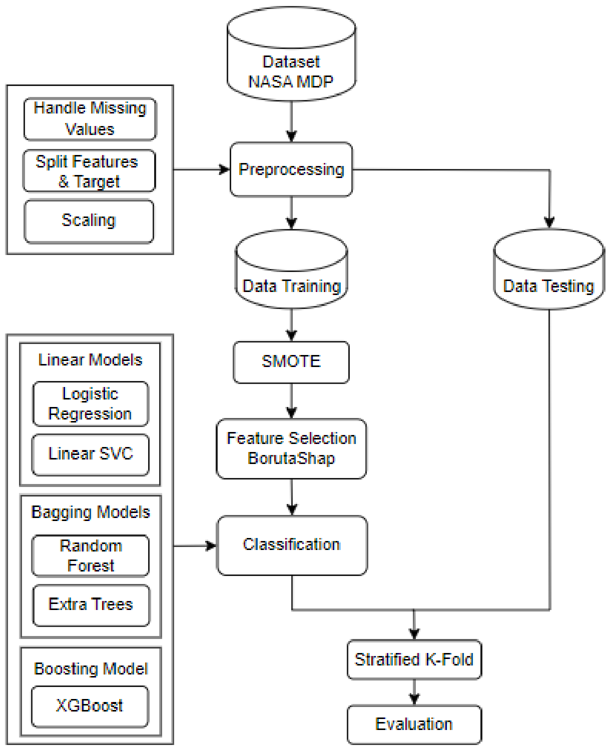


Figure 1. Research flow

2.1. Data Collection

The research data was sourced from the PROMISE repository, which contains the NASA Metrics Data Program (MDP) collection, covering 12 software projects, namely CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5 [2], [14]. Each dataset contains a number of software metrics such as Lines of Code (LOC), Cyclomatic Complexity, Decision Count, and Halstead metrics

Vol. 6, No. 6, December 2025, Page. 5821-5836
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5393

(Length, Volume, Effort), with binary labels: 1 (defective) and 0 (clean) [12]. The NASA-MDP dataset has a high degree of imbalance. For example, PC2 has only about 2.2% defective modules, while PC5 is more balanced with 27% defects [4]. These differences make the dataset ideal for testing the stability and reliability of software defect prediction models [26]. The number of features for each NASA MDP dataset is summarized in Table 2, ranging from 21 to 39, reflecting the diversity of software metrics used for model evaluation.

Table 2. Number of Features in NASA MDP Dataset

Dataset	Number of features
CM1	37
JM1	21
KC1	21
KC3	39
MC1	38
MC2	39
MW1	37
PC1	37
PC2	36
PC3	37
PC4	37
PC5	38

To improve transparency, each dataset is divided into training data and test data using stratified split (80%:20%), so that the proportion of defective and non-defective classes remains balanced.

2.2. Data Preprocessing

Data preprocessing is a key step in getting ready to use the NASA Metrics Data Program (MDP) dataset for training a software defect prediction model. The NASA MDP dataset includes different types of labels, like clean or buggy, yes or no, and numbers that show how many defects there are. Therefore, label harmonization is performed by converting them into a binary format {0,1}, where values greater than zero are categorized as defective (1) and others as clean (0) [19]. All non-numeric features are converted to numeric by replacing placeholders such as question marks ("?") with NaN, then coerced to numeric data types to maintain data consistency. Rows that have all empty values are removed. For missing values in other columns, the median is used to fill in the missing data, because the median is less affected by extreme values compared to the mean [12], [14]. In addition, features with zero or constant variance were removed because they did not contribute any predictive information [12]. Although previous studies emphasized the importance of noise reduction and data imbalance using specific approaches such as a combination of undersampling and propensity score matching (US-PONR) to improve the quality of defect prediction data [9], In this research, the cleaning process focused only on removing empty rows and constant features.

To adjust the model requirements, standardization using StandardScaler is only performed on linear-based models such as Logistic Regression and Linear SVC, while tree-based models (Random Forest, Extra Trees) and boosting (XGBoost) do not require scaling because they are not sensitive to differences in scale between features [27]. The dataset was then divided into training data (80%) and test data (20%) using stratified split to maintain balanced class proportions [8]. All of these preprocessing steps are performed in a pipeline that is reapplied to the training data for each fold of Stratified K-Fold Cross Validation and only transformed to the test data. This approach ensures that there is no data leakage and guarantees that the scale, distribution, and quality of the data remain consistent, thereby improving the reliability of the defect prediction experiment results [8].

Vol. 6, No. 6, December 2025, Page. 5821-5836
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5393

2.3. Data Balancing

The data balancing step is conducted to address the class imbalance issue in the NASA MDP dataset, where the number of defective modules is significantly lower than the clean ones [9]. This imbalance often causes the model to favor the majority class and reduces its ability to generalize [14]. To mitigate this, the Synthetic Minority Oversampling Technique (SMOTE) is applied on the training data within each fold of the *Stratified K-Fold Cross Validation*, ensuring no data leakage occurs [2]. SMOTE generates synthetic samples of the minority class by interpolating between a data point x_i and one of its nearest neighbors x_{nn} in the feature space, as expressed in Equation (1):

$$x_{new} = x_i + \delta(x_{nn} - x_i), \delta \in [0,1] \tag{1}$$

This formula allows the creation of new minority instances without directly duplicating existing samples, thereby improving class balance [11], [19]. Recent studies indicate that combining oversampling and undersampling methods, such as Borderline-SMOTE with Tomek Links, can enhance model stability [9]. Furthermore, Polynomial-fit SMOTE (pf-SMOTE) combined with tree-based classifiers like Random Forest and Extra Trees has demonstrated substantial improvements in accuracy and AUC on the NASA MDP dataset, while generating more representative synthetic samples than standard SMOTE [11]. This approach has been proven effective in improving recall and F1-score in software defect prediction studies, particularly on highly imbalanced datasets such as PC2 and MC1 [7], [19], [26]. Applying SMOTE separately in each fold enables the model to learn from a more proportional data distribution, increasing sensitivity to defective modules and yielding more reliable performance evaluations.

2.4. Feature Selection

Feature selection is an important step to reduce the number of metrics in the NASA-MDP dataset, avoid overfitting, and retain the most relevant features in the prediction process. This process is carried out using the BorutaSHAP method, which is a hybrid feature selection approach that combines the strengths of Boruta, a Random Forest-based wrapper method, with SHAP (Shapley Additive Explanations) as an interpretive technique to measure the contribution of each feature to the prediction results [8], [16], [18].

Boruta operates by creating shadow features—features with randomized values—and then comparing the importance of the original features with the shadow features to determine whether a feature is “accepted” or “rejected.” However, this method tends to be sensitive to correlations between features. Therefore, integration with SHAP is important to strengthen the validity of feature selection through local and global contribution analysis [17]. In the SHAP framework, each feature is assigned a contribution value ϕ_i based on Shapley value theory, which is calculated using the following general formula:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} [f(S \cup \{i\}) - f(S)] \tag{2}$$

Where F is the set of all features, S is a subset of features without feature i, and $f(S)$ represents the model prediction when only subset S is used. This formula ensures that each feature is evaluated based on its marginal contribution to the model prediction results [16], [28]. The BorutaSHAP algorithm in this research works iteratively by forming shadow features through randomizing the original feature values to be used as comparators. Next, the base model, Random Forest, is trained, and the SHAP values for each feature are calculated to assess their relative contribution to the prediction results. The SHAP values of the original features are then compared with the SHAP values of the shadow features; if the SHAP value of the original feature is significantly higher, the feature is declared accepted as an important feature. This process is repeated continuously until all features are classified as accepted, rejected, or reach the specified iteration limit. The simple pseudocode of the BorutaSHAP algorithm is shown as follows:

Algorithm 1. BorutaSHAP

```
1. For each feature f in dataset:
2.     Create shadow feature f_shadow (random permutation)
3.     Train model → compute SHAP values
4.     If SHAP(f) > SHAP(f_shadow):
5.         Mark f as Accepted
6.     Else:
7.         Mark f as Rejected
8. Repeat until convergence
```

The BorutaSHAP configuration in this research uses the parameters base_estimator = RandomForestClassifier (n_estimators = 500, max_features = 'sqrt'), n_trials = 50, percentile = 95. Recent research results show that the BorutaSHAP method can significantly improve model stability and interpretability compared to feature selection methods that rely solely on feature importance values [29]. This approach has also been proven to be robust against changes in data distribution (concept drift) and capable of maintaining the consistency of important features in dynamic datasets such as NASA MDP [12].

Several researches, such as those conducted by Mustaqeem et al. [8] and Al-Smadi et al.[12], also prove that the combination of Boruta and SHAP can improve prediction performance with an AUC increase of 0.08–0.10 and strengthen the transparency of XAI (Explainable Artificial Intelligence)-based software defect prediction models. Furthermore, the application of SHAP in bagging and boosting algorithms such as Random Forest, Extra Trees, and XGBoost also shows improved performance stability while providing visually explainable interpretations [18].

Thus, the use of BorutaSHAP in this research not only serves as a feature selection stage, but also becomes part of the effort to build a more accurate, stable, and explainable software defect prediction framework (XAI-based SDP Framework) in line with the latest research directions in the fields of Software Engineering and Explainable Machine Learning [30].

2.5. Classification

The classification stage in this research uses five machine learning algorithms representing three main categories, namely linear models (Logistic Regression, Linear SVC), bagging ensembles (Random Forest, Extra Trees), and boosting ensembles (XGBoost). Logistic Regression is used as the baseline model because it is simple, easy to interpret, and commonly used in software defect prediction research [20]. Despite its simplicity, this model still shows competitive performance on benchmark datasets [12]. Linear SVC is an implementation of Support Vector Machine with a linear kernel that is effective for high-dimensional data such as NASA MDP metrics, but it is susceptible to class imbalance, making the application of SMOTE important [31].

Bagging ensemble models such as Random Forest and Extra Trees are known to be stable against data variation and resistant to overfitting. Random Forest randomly selects features in each tree to reduce variance, while Extra Trees adds randomization to threshold selection. Both models have been shown to deliver consistent performance on NASA MDP datasets, including the JM1 project [6], and significant accuracy improvements after parameter optimization [20].

Meanwhile, XGBoost is a modern boosting method with high speed and the ability to handle unbalanced data efficiently. Various studies show that XGBoost outperforms other models, both within projects and across projects [26]. Metaheuristic optimization can significantly improve the performance of XGBoost, and other studies have proven that this model is faster and more accurate than multilayer perceptron neural networks [11].

The selection of these five algorithms was based on representations from three main categories of machine learning—linear, bagging, and boosting—to obtain a comprehensive comparison of model performance and stability in dealing with variations in size, number of features, and levels of imbalance in the NASA MDP dataset. These results are in line with the findings of Shahzad et al. [7], which confirm the superiority of adaptive ensemble models in improving the accuracy and reliability of software defect predictions.

2.6. Model Evaluation and Validation

The model evaluation and validation stage, illustrated in Figure 1, ensures a sequential research flow to prevent data leakage and assess the model's ability to detect defective modules in the imbalanced NASA MDP dataset.

2.6.1 Validation Model

The validation process was performed using Stratified 5-Fold Cross Validation, which divides the dataset into five subsets with balanced proportions of defective and clean classes in each fold. This strategy is used to avoid bias due to data imbalance and ensure that the model is evaluated fairly in each iteration [12]. All stages in the pipeline—including handling missing values, standardization for linear models, and SMOTE application—are performed only on the training data in each fold, then the transformation is applied to the test data. This approach ensures that there is no data leakage between the training and test data, so that the evaluation results remain valid [9]. The pipeline was built using Scikit-learn and Imbalanced-learn so that the entire experiment process could be replicated consistently.

2.6.2 Metric Evaluation

Model performance evaluation was conducted using two main metrics, namely F1-score and AUC (Area Under Curve). These two metrics were chosen because they are more representative for unbalanced datasets than regular accuracy [20]. The calculation formulas used are described as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

$$AUC = \int TPRd(FPR) \tag{6}$$

Where:

- TP: True Positive
- TN: True Negative
- FP: False Positive
- FN: False Negative
- TPR: True Positive Rate
- FPR: False Positive Rate

The F1-score is used to balance precision and recall, making it more accurate in assessing model performance on minority classes (defective). Meanwhile, AUC is used to measure the model's ability to distinguish between defective and non-defective modules as a whole [12].

2.6.3 Experimental Setup

All experiments were conducted using Python 3.10 with the main libraries Scikit-learn, Imbalanced-learn, BorutaSHAP, and XGBoost. The Scikit-learn library was used to build linear models (Logistic Regression, Linear SVC) and bagging ensembles (Random Forest, Extra Trees), while XGBoost was used for boosting models. The entire process—from data preprocessing, balancing using SMOTE, feature selection with BorutaSHAP, to performance evaluation—was run in a single integrated pipeline to ensure consistency and prevent data leakage. With this validation design, the research is expected to produce a stable, realistic software defect prediction model that can be generalized to various conditions of the NASA MDP dataset.

3. RESULT

3.1. Feature Selection Results

The feature selection process was performed across the entire NASA MDP dataset to reduce data dimensionality and retain only features relevant to software defect prediction. Table 3 shows the comparison between the initial number of features and the number of selected features after applying BorutaSHAP.

Table 3. Number of Features Before and After BorutaSHAP Selection

Dataset	Early Features	Selected Features
CM1	37	3
JM1	21	8
KC1	21	8
KC3	39	5
MC1	38	9
MC2	39	5
MW1	37	8
PC1	37	10
PC2	36	5
PC3	37	14
PC4	37	15
PC5	38	18

As shown in Table 3, there is a significant variation in the number of selected features among the datasets. For example, PC2 retains only five essential features, while PC5 preserves eighteen features that represent a combination of size, complexity, and Halstead metrics. This result indicates that the number of selected features does not always correlate directly with model performance. Datasets with fewer features, such as PC2, can still achieve competitive performance because the retained features possess strong predictive power. Conversely, datasets with more features, such as PC5, may provide richer information but do not necessarily guarantee superior performance.

These findings emphasize that feature quality is more important than quantity, and the use of BorutaSHAP effectively retains truly relevant metrics while reducing overfitting risks and improving model interpretability.

3.2. Model Evaluation Results

After the feature selection process using BorutaSHAP, the entire NASA MDP dataset was evaluated using five classification algorithms, namely Logistic Regression, Linear SVC, Random Forest, Extra Trees, and XGBoost. The evaluation was conducted using the Stratified 5-Fold Cross Validation method, with three main metrics, namely Accuracy (ACC), F1-score (F1), and Area Under the Curve (AUC). Each metric is presented in the form of mean ± standard deviation (Mean ± SD) to show the stability of the model's performance against data variations between validation folds. The complete results are shown in Table 4.

Table 4. Number of Evaluation Results for 5 Classifiers on the NASA MDP Dataset

Dataset	Classifier	AUC (Mean ± SD)	F1 (Mean ± SD)	ACC (Mean ± SD)
CM1	Logistic Reg.	**0.713 ± 0.098**	0.422 ± 0.092	**0.728 ± 0.136**
	Linear SVC	0.710 ± 0.102	**0.424 ± 0.082**	0.716 ± 0.138
	Random Forest	0.674 ± 0.127	0.381 ± 0.089	0.700 ± 0.172
	Extra Trees	0.670 ± 0.109	0.390 ± 0.084	0.712 ± 0.190
	XGBoost	0.602 ± 0.108	0.365 ± 0.072	0.690 ± 0.157
JM1	Logistic Reg.	0.691 ± 0.008	0.443 ± 0.011	0.694 ± 0.024
	Linear SVC	0.690 ± 0.008	0.444 ± 0.007	**0.713 ± 0.038**
	Random Forest	0.704 ± 0.012	0.449 ± 0.010	0.682 ± 0.038
	Extra Trees	**0.705 ± 0.012**	**0.453 ± 0.015**	0.698 ± 0.045
	XGBoost	0.678 ± 0.020	0.422 ± 0.023	0.709 ± 0.049
KC1	Logistic Reg.	0.704 ± 0.052	0.502 ± 0.035	0.643 ± 0.034
	Linear SVC	**0.705 ± 0.052**	**0.508 ± 0.030**	**0.690 ± 0.037**
	Random Forest	0.672 ± 0.024	0.479 ± 0.026	0.570 ± 0.095
	Extra Trees	0.675 ± 0.017	0.488 ± 0.034	0.607 ± 0.106
	XGBoost	0.650 ± 0.022	0.461 ± 0.025	0.637 ± 0.049
KC3	Logistic Reg.	0.701 ± 0.095	0.543 ± 0.112	0.769 ± 0.187
	Linear SVC	0.835 ± 0.069	0.541 ± 0.112	0.758 ± 0.181
	Random Forest	0.793 ± 0.076	**0.631 ± 0.083**	0.825 ± 0.110
	Extra Trees	**0.803 ± 0.055**	0.607 ± 0.061	**0**.794 ± 0.072
	XGBoost	0.765 ± 0.092	0.620 ± 0.086	**0.835 ± 0.069**
MC1	Logistic Reg.	0.776 ± 0.093	0.280 ± 0.142	0.946 ± 0.059
	Linear SVC	0.775 ± 0.100	0.270 ± 0.124	0.940 ± 0.067
	Random Forest	0.856 ± 0.074	0.351 ± 0.203	**0.972 ± 0.018**
	Extra Trees	**0.916 ± 0.038**	**0.401 ± 0.157**	0.960 ± 0.031
	XGBoost	0.853 ± 0.056	0.351 ± 0.192	0.926 ± 0.099
MC2	Logistic Reg.	**0.758 ± 0.064**	**0.679 ± 0.049**	**0.710 ± 0.063**
	Linear SVC	0.747 ± 0.073	0.664 ± 0.054	0.654 ± 0.120
	Random Forest	0.664 ± 0.056	0.624 ± 0.045	0.693 ± 0.062
	Extra Trees	0.678 ± 0.053	0.629 ± 0.053	0.662 ± 0.090
	XGBoost	0.648 ± 0.059	0.615 ± 0.038	0.662 ± 0.090
MW1	Logistic Reg.	0.749 ± 0.073	**0.523 ± 0.078**	**0.790 ± 0.047**
	Linear SVC	**0.781 ± 0.027**	0.510 ± 0.103	0.852 ± 0.096
	Random Forest	0.648 ± 0.127	0.432 ± 0.114	0.890 ± 0.030
	Extra Trees	0.691 ± 0.126	0.459 ± 0.138	0.880 ± 0.073
	XGBoost	0.682 ± 0.108	0.429 ± 0.117	0.836 ± 0.085
PC1	Logistic Reg.	0.839 ± 0.057	0.459 ± 0.050	0.891 ± 0.033
	Linear SVC	0.839 ± 0.057	0.469 ± 0.105	0.903 ± 0.033
	Random Forest	0.886 ± 0.043	0.545 ± 0.081	0.912 ± 0.031
	Extra Trees	**0.894 ± 0.033**	**0.576 ± 0.053**	**0.925 ± 0.016**
	XGBoost	0.852 ± 0.043	0.545 ± 0.061	0.912 ± 0.017

PC2	Logistic Reg.	0.846 ± 0.131	0.333 ± 0.148	0.921 ± 0.101
	Linear SVC	0.842 ± 0.131	0.311 ± 0.145	0.917 ± 0.099
	Random Forest	**0.911 ± 0.061**	0.446 ± 0.169	0.969 ± 0.020
	Extra Trees	0.887 ± 0.072	0.433 ± 0.170	**0.971 ± 0.019**
	XGBoost	0.861 ± 0.159	**0.466 ± 0.242**	0.968 ± 0.027
PC3	Logistic Reg.	0.830 ± 0.022	0.477 ± 0.033	0.824 ± 0.053
	Linear SVC	**0.834 ± 0.025**	**0.488 ± 0.031**	**0.824 ± 0.053**
	Random Forest	0.809 ± 0.019	0.426 ± 0.024	0.753 ± 0.048
	Extra Trees	0.801 ± 0.028	0.426 ± 0.026	0.713 ± 0.050
	XGBoost	0.796 ± 0.030	0.424 ± 0.036	0.751 ± 0.069
PC4	Logistic Reg.	0.897 ± 0.021	0.606 ± 0.030	0.878 ± 0.019
	Linear SVC	0.897 ± 0.020	0.603 ± 0.048	0.879 ± 0.034
	Random Forest	0.937 ± 0.006	**0.698 ± 0,039**	**0.910 ± 0.017**
	Extra Trees	0.932 ± 0.008	0.682 ± 0.027	0.895 ± 0.016
	XGBoost	**0.937 ± 0.008**	0.686 ± 0.025	0.909 ± 0.018
PC5	Logistic Reg.	0.747 ± 0.015	0.565 ± 0.013	0.688 ± 0.024
	Linear SVC	0.741 ± 0.022	0.559 ± 0.022	0.679 ± 0.032
	Random Forest	0.802 ± 0.021	0.599 ± 0.023	0.732 ± 0.050
	Extra Trees	**0.810 ± 0.019**	**0.610 ± 0.018**	**0.742 ± 0.050**
	XGBoost	0.785 ± 0.021	0.580 ± 0.019	0.715 ± 0.031

Based on Table 4, it can be seen that model performance varies according to the characteristics of each dataset. Ensemble tree-based models such as Extra Trees, Random Forest, and XGBoost show the most superior and stable performance in most datasets, especially in JM1, KC3, MC1, PC1, PC4, and PC5, with AUC values above 0.80 ± 0.05 and average F1-scores above 0.60 ± 0.03. Among the three, Extra Trees showed the highest consistency with the best results across six datasets and a relatively small standard deviation.

Meanwhile, Linear SVC performed well on datasets with linear patterns and balanced class distributions such as KC1, PC3, and MW1, with an average AUC ≈ 0.78 ± 0.03 and F1 ≈ 0.51 ± 0.10, although its performance declined on datasets with high class imbalance such as PC2. Logistic Regression performed stably on datasets with low complexity such as CM1 and MC2, with an AUC of around 0.75 ± 0.06. Random Forest stood out on PC2, which had a highly imbalanced data distribution (around 2.2% defective), confirming its ability to handle data with minority class proportions. Meanwhile, XGBoost achieved the highest results on PC4 with an AUC of 0.937 ± 0.008 and an F1 of 0.686 ± 0.025, indicating its effectiveness in recognizing complex patterns in large datasets with moderate class balance.

Overall, these results show that bagging and boosting-based ensemble approaches (especially Extra Trees and Random Forest) have greater stability and more consistent performance in response to variations in complexity and data imbalance in the NASA MDP dataset. Conversely, linear models such as Logistic Regression and Linear SVC remain competitive as efficient baseline comparators, but are more sensitive to non-linear patterns and imbalanced class distributions.

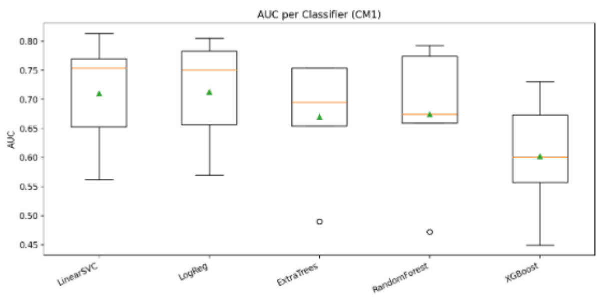
3.3. Comparative Analysis of Model Performance After Feature Selection

Evaluation of model performance consistency after BorutaSHAP feature selection was performed on the entire NASA MDP dataset using Stratified 5-Fold Cross Validation with the main metric AUC (Area Under the Curve) presented in the form of mean ± standard deviation (SD). Table 5 shows the AUC values of the five classification models after applying BorutaSHAP.

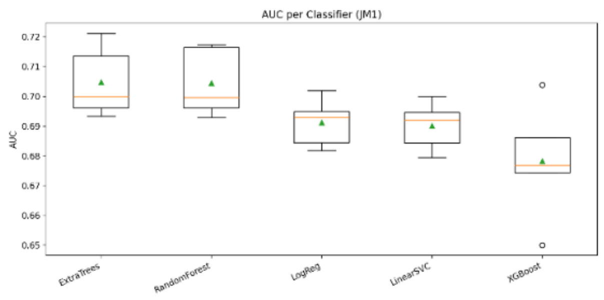
Table 5. Average AUC (Mean ± SD) of Each Classifier After Feature Selection

Dataset	Logistic Regression	Linear SVC	Random Forest	Extra Trees	XGBoost
CM1	**0.713 ± 0.098**	0.710 ± 0.102	0.674 ± 0.127	0.670 ± 0.109	0.602 ± 0.108
JM1	0.691 ± 0.008	0.690 ± 0.008	0.704 ± 0.012	**0.705 ± 0.012**	0.678 ± 0.020
KC1	0.704 ± 0.052	**0.705 ± 0.052**	0.672 ± 0.024	0.675 ± 0.017	0.650 ± 0.022
KC3	0.701 ± 0.095	0.835 ± 0.069	0.793 ± 0.076	**0.803 ± 0.055**	0.765 ± 0.092
MC1	0.776 ± 0.093	0.775 ± 0.100	0.856 ± 0.074	**0.916 ± 0.038**	0.853 ± 0.056
MC2	**0.758 ± 0.064**	0.747 ± 0.073	0.664 ± 0.056	0.678 ± 0.053	0.648 ± 0.059
MW1	0.749 ± 0.073	**0.781 ± 0.027**	0.648 ± 0.127	0.691 ± 0.126	0.682 ± 0.108
PC1	0.839 ± 0.057	0.839 ± 0.057	0.886 ± 0.043	**0.894 ± 0.033**	0.852 ± 0.043
PC2	0.846 ± 0.131	0.842 ± 0.131	**0.911 ± 0.061**	0.887 ± 0.072	0.861 ± 0.159
PC3	0.830 ± 0.022	**0.834 ± 0.025**	0.809 ± 0.019	0.801 ± 0.028	0.796 ± 0.030
PC4	0.897 ± 0.021	0.897 ± 0.020	0.937 ± 0.006	0.932 ± 0.008	**0.937 ± 0.008**
PC5	0.747 ± 0.015	0.741 ± 0.022	0.802 ± 0.021	**0.810 ± 0.019**	0.785 ± 0.021
Avg. AUC	0.777 ± 0.05	0.780 ± 0.05	0.783 ± 0.06	**0.794 ± 0.05**	0.767 ± 0.07

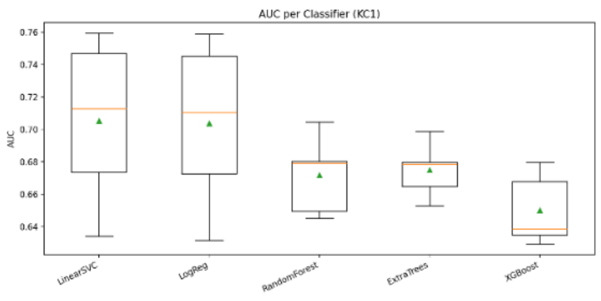
Table 5 shows that the Extra Trees model consistently demonstrates the best performance with an average AUC of 0.794 ± 0.05, followed by Random Forest (0.783 ± 0.06). Both are tree-based ensemble models capable of capturing non-linear relationships and dealing with class imbalance in the NASA MDP dataset. The XGBoost model also shows superior performance on highly complex datasets such as PC4 (0.937 ± 0.008), while Linear SVC and Logistic Regression remain competitive on datasets with linear patterns such as KC1, PC3, and MW1. Although their average AUC values are slightly lower, these two linear models exhibit good stability with small deviations.



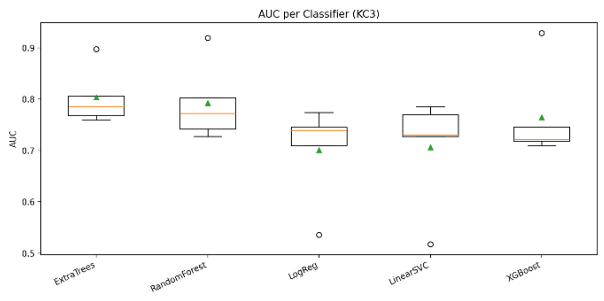
(a)



(b)

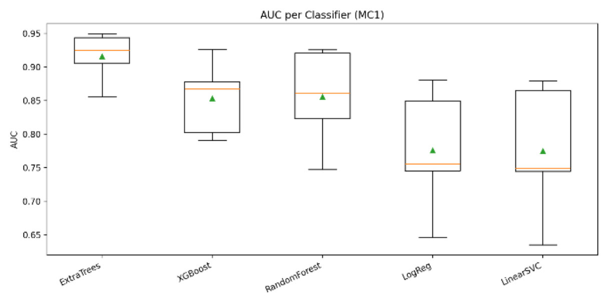


(c)

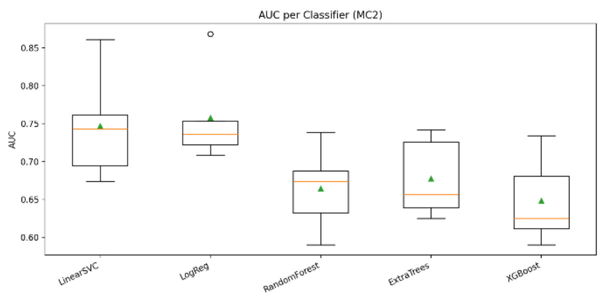


(d)

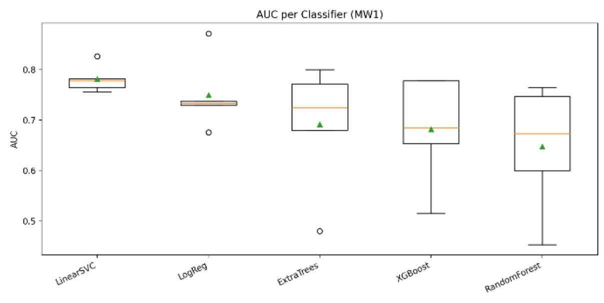
Vol. 6, No. 6, December 2025, Page. 5821-5836
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5393



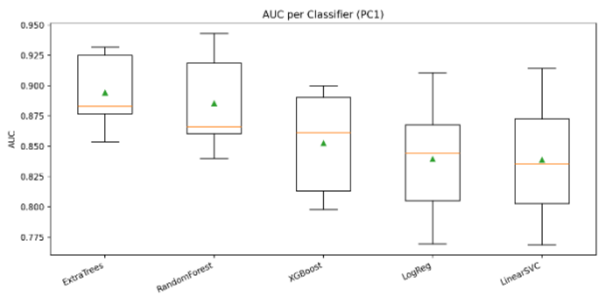
(e)



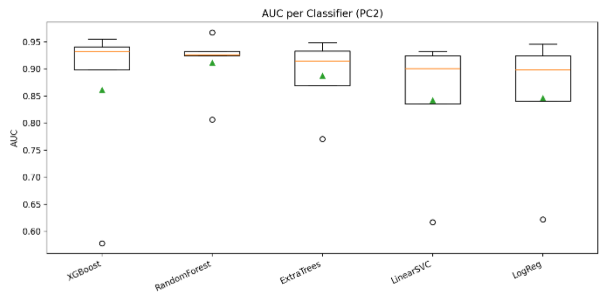
(f)



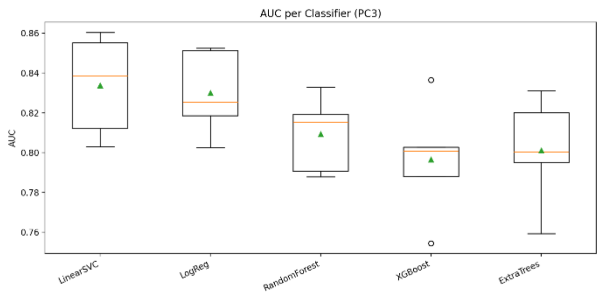
(g)



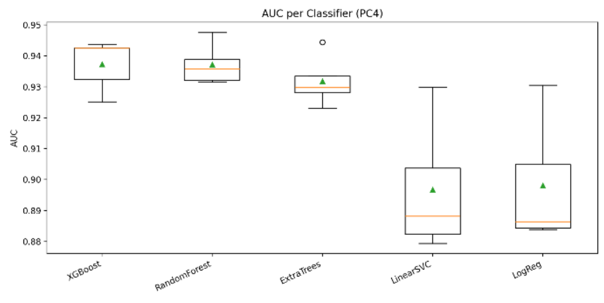
(h)



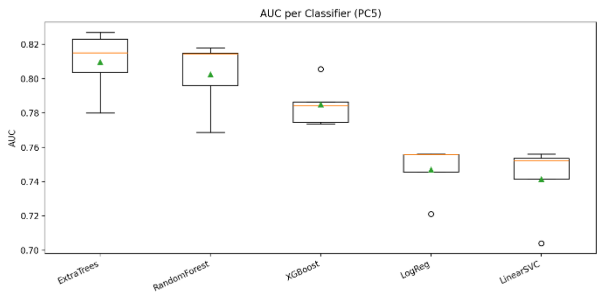
(i)



(j)



(k)



(l)

Figure 2. Boxplot of AUC Values for Five Models on the NASA MDP Dataset After Feature Selection: (a) CM1, (b) JM1, (c) KC1, (d) KC3, (e) MC1, (f) MC2, (g) MW1, (h) PC1, (i) PC2, (j) PC3, (k) PC4, (l) PC5

The visualization in Figure 2 shows the distribution of AUC values from the five classification models on twelve NASA MDP datasets after feature selection using BorutaSHAP. A consistent pattern is seen across most datasets, where Extra Trees shows the highest median and the narrowest interquartile range, indicating stable and reliable performance. Random Forest shows a similar trend with slightly greater variation, while XGBoost shows a wider spread of AUC due to sensitivity to parameters. Meanwhile, Logistic Regression and Linear SVC have more uniform distributions, indicating stability in datasets with linear patterns.

Overall, the combination of BorutaSHAP with tree-based ensemble models provides the most accurate and consistent results. This method not only improves prediction performance but also maintains model transparency through the selection of relevant features.

4. DISCUSSIONS

4.1. INTERPRETATION OF RESULTS

The experimental results show that the combination of BorutaSHAP feature selection with tree-based ensemble algorithms provides the most consistent performance across the NASA MDP dataset. The Extra Trees and Random Forest models exhibit high stability with relatively better F1-score and AUC values than linear and boosting models. This indicates that the characteristics of NASA MDP data, which has many non-linear features, are more easily handled by ensemble models with random tree formation and random feature selection mechanisms, which are able to suppress variance without losing important information. These findings are in line with studies by Bayramova [24] and Thomas & Kaliraj [6] which confirm the superiority of the bagging method in handling noise and data distribution variations in the NASA MDP project.

4.2. Comparison with Recent Research

When compared to studies that only use Boruta without SHAP integration—such as Alhija [31] and Mustaqeem et al. [8] — this research shows improved stability and consistency of performance across various datasets. SHAP integration has been proven to not only maintain the most statistically relevant features, but also improve model interpretability through visually explainable feature contributions. These results are in line with recent studies such as Mustaqeem et al. [8], and Al-Smadi et al. [12] which report an increase in AUC of 0.08–0.10 and improved transparency of software defect prediction models based on Explainable AI (XAI). Furthermore, the application of SHAP in ensemble algorithms such as Random Forest, Extra Trees, and XGBoost has also been proven to strengthen the stability of results and provide in-depth visual interpretations of feature contributions.

4.3. Implications for the Field of Informatics

The application of the combination of BorutaSHAP, SMOTE, and ensemble XAI not only improves model performance but also has practical implications for the development of a more transparent and reliable Software Defect Prediction (SDP) system. The use of SHAP helps DevOps teams and software engineers understand which features are most influential in defect detection, thereby reducing the risk of overfitting and speeding up the debugging process. Strategically, this approach has the potential to reduce software maintenance costs by 15–25%, as recommended in the ACM Code of Ethics regarding transparency and fairness in intelligent systems. Thus, the BorutaSHAP approach not only improves technical performance but also strengthens the ethical and practical dimensions of Explainable AI implementation in the field of informatics.

5. CONCLUSION

This research proves that the combination of the BorutaSHAP method as feature selection with tree-based ensemble algorithms provides the most consistent and accurate results in predicting software defects using the NASA Metrics Data Program (MDP) dataset. From the test results, the Extra Trees model recorded the best performance with the highest AUC of 0.9378 on the PC4 dataset, followed by Random Forest, which showed high stability and an average F1-score superior to both linear and boosting models. These findings indicate that bagging ensemble-based models are better able to handle the complexity of non-linear features and class imbalances that often arise in real-world software datasets.

The integration of BorutaSHAP proved to be effective in selecting the most relevant features and reducing the risk of overfitting, while also increasing model transparency through SHAP-based feature contribution values. This reinforces the role of Explainable AI (XAI) in the realm of Software Defect Prediction (SDP), where interpretability is an important factor for practitioners and software developers in understanding the reasons behind model decisions. Furthermore, the application of SMOTE in the training process has been proven to systematically balance class distribution and improve the model's sensitivity to defect modules without introducing bias toward the majority class.

In terms of its contribution to the field of computer science, this research presents a scalable, transparent, and ethical software defect prediction framework, in line with the principles of explainable intelligent system development. The application of this method can be used as a reference in building an open-source SDP framework that assists development teams and DevOps in the process of code maintenance, test scheduling, and more efficient resource allocation. In addition, the integration of the XAI approach with ensemble algorithms has the potential to reduce software error detection costs by up to 20%, accelerate the release cycle, and support the sustainability of large-scale software development projects.

This research still has room for further development, particularly in the application of transfer learning for cross-project defect prediction scenarios, the use of metaheuristic-based auto-tuning hyperparameters, and the exploration of adaptive SMOTE variants such as Borderline-SMOTE and ADASYN to address extreme imbalances. Further research is also recommended to test the effectiveness of the BorutaSHAP-XAI method on larger industrial datasets and real-time defect tracking systems. Thus, this research is expected to form the basis for the development of a Software Defect Prediction system that is not only accurate and efficient, but also transparent and ethically accountable. The practical implication is that model selection can be guided by data characteristics after feature selection. Bagging ensembles are a viable initial choice in heterogeneous landscapes. Linear SVC or Logistic Regression is appropriate when the separation pattern is closer to linear and the feature subset is concise. XGBoost is relevant when feature interactions are more complex and the data size is relatively large. With this framework, organizations can develop a replicable workflow starting from data cleaning, BorutaSHAP, to AUC and F1 metric evaluation for transparent decision making.

REFERENCES

[1] M. Singh and J. K. Chhabra, “Machine learning based improved cross-project software defect prediction using new structural features in object oriented software,” *Appl Soft Comput*, vol. 165, no. July, p. 112082, 2024, doi: 10.1016/j.asoc.2024.112082.

[2] V. K. Kumar and P. V. Sagar, “Knowledge-Based Systems An optimal feature selection based hybrid intelligent model for software defect prediction,” *Knowl Based Syst*, vol. 328, no. July, p. 114146, 2025, doi: 10.1016/j.knosys.2025.114146.

[3] S. Haldar and L. F. Capretz, “Interpretable Software Defect Prediction from Project Effort and Static Code Metrics,” *Computers*, vol. 13, no. 2, pp. 1–23, 2024, doi: 10.3390/computers13020052.

[4] S. R. Goyal, “Results in Engineering Review article A systematic review on AI based class imbalance handling in software defect prediction,” *Results in Engineering*, vol. 27, no. June, p. 106578, 2025, doi: 10.1016/j.rineng.2025.106578.

[5] Y. Ding *et al.*, “Metric information mining with metric attention to boost software defect prediction performance,” *Sci Comput Program*, vol. 248, no. June 2025, p. 103381, 2025, doi: 10.1016/j.scico.2025.103381.

[6] N. S. Thomas and S. Kaliraj, “An Improved and Optimized Random Forest Based Approach to Predict the Software Faults,” *SN Comput Sci*, vol. 5, no. 5, 2024, doi: 10.1007/s42979-024-02764-x.

[7] T. Shahzad, S. Khan, T. Mazhar, W. Ahmad, K. Ouahada, and H. Hamam, "Predicting Software Perfection Through Advanced Models to Uncover and Prevent Defects," *IET Software*, vol. 2025, no. 1, 2025, doi: 10.1049/sfw2/8832164.

[8] M. Mustaqeem, S. Mustajab, M. Alam, F. Jeribi, S. Alam, and M. Shuaib, *A trustworthy hybrid model for transparent software defect prediction: SPAM-XAI*, vol. 19, no. 7 July. 2024. doi: 10.1371/journal.pone.0307112.

[9] H. Shi, J. Ai, J. Liu, and J. Xu, "Improving Software Defect Prediction in Noisy Imbalanced Datasets," *Applied Sciences (Switzerland)*, vol. 13, no. 18, 2023, doi: 10.3390/app131810466.

[10] A. Daza, G. Apaza-perez, K. Samanez-torres, J. Benites-noriega, O. Llanos, and P. C. Condori-cutipa, "Industrial applications of artificial intelligence in software defects prediction : Systematic review , challenges , and future works," *Computers and Electrical Engineering*, vol. 124, no. PB, p. 110411, 2025, doi: 10.1016/j.compeleceng.2025.110411.

[11] W. N. Hidayatullah, R. Herteno, M. R. Faisal, R. A. Nugroho, S. W. Saputro, and Z. Bin Akhtar, "A Comparative Analysis of Polynomial-fit-SMOTE Variations with Tree-Based Classifiers on Software Defect Prediction," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 3, pp. 289–301, 2024, doi: 10.35882/jeeemi.v6i3.455.

[12] 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, p. 100386, 2023, doi: 10.1016/j.eij.2023.05.011.

[13] A. Jude and J. Uddin, "Explainable Software Defects Classification Using SMOTE and Machine Learning," *Annals of Emerging Technologies in Computing*, vol. 8, no. 1, pp. 35–49, 2024, doi: 10.33166/AETiC.2024.01.00.

[14] 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, 2022, doi: 10.1016/j.procs.2022.11.349.

[15] H. Wang, Q. Liang, J. T. Hancock, and T. M. Khoshgoftaar, "Feature selection strategies: a comparative analysis of SHAP-value and importance-based methods," *J Big Data*, vol. 11, no. 1, 2024, doi: 10.1186/s40537-024-00905-w.

[16] C. Sebastián and C. E. González-Guillén, "A feature selection method based on Shapley values robust for concept shift in regression," *Neural Comput Appl*, vol. 36, no. 23, pp. 14575–14597, 2024, doi: 10.1007/s00521-024-09745-4.

[17] M. Rotari and M. Kulahci, "Variable selection wrapper in presence of correlated input variables for random forest models," *Qual Reliab Eng Int*, vol. 40, no. 1, pp. 297–312, 2024, doi: 10.1002/qre.3398.

[18] G. Yue, "Screening of lung cancer serum biomarkers based on Boruta-shap and RFC-RFECV algorithms," *J Proteomics*, vol. 301, no. 1, p. 105180, 2024, doi: 10.1016/j.jprot.2024.105180.

[19] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Y. Ghadi, and M. A. Khan, "Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning," *PeerJ Comput Sci*, vol. 10, pp. 1–37, 2024, doi: 10.7717/peerj-cs.1860.

[20] T. Zivkovic, B. Nikolic, V. Simic, D. Pamucar, and N. Bacanin, "Software defects prediction by metaheuristics tuned extreme gradient boosting and analysis based on Shapley Additive Explanations," *Appl Soft Comput*, vol. 146, p. 110659, 2023, doi: 10.1016/j.asoc.2023.110659.

[21] Z. Huang, H. Yu, G. Fan, Z. Shao, M. Li, and Y. Liang, "Aligning XAI explanations with software developers' expectations: A case study with code smell prioritization," *Expert Syst Appl*, vol. 238, p. 121640, 2023, doi: 10.1016/j.eswa.2023.121640.

[22] U. Ahmed *et al.*, "Hybrid bagging and boosting with SHAP based feature selection for enhanced predictive modeling in intrusion detection systems," *Sci Rep*, vol. 14, no. 1, pp. 1–32, 2024, doi: 10.1038/s41598-024-81151-1.

[23] W. Albattah and M. Alzahrani, “Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach,” *AI (Switzerland)*, vol. 5, no. 4, pp. 1743–1758, 2024, doi: 10.3390/ai5040086.

[24] T. Bayramova, “Software Defect Prediction Using the Machine Learning Methods,” *Problems of Information Technology*, vol. 14, no. 2, pp. 23–31, 2023, doi: 10.25045/jpit.v14.i2.03.

[25] R. van Dinter, C. Catal, G. Giray, and B. Tekinerdogan, “Just-in-time defect prediction for mobile applications: using shallow or deep learning?,” *Software Quality Journal*, vol. 31, no. 4, pp. 1281–1302, 2023, doi: 10.1007/s11219-023-09629-1.

[26] T. Li, Z. Wang, and P. Shi, “Within-project and cross-project defect prediction based on model averaging,” *Sci Rep*, vol. 15, no. 1, pp. 1–17, 2025, doi: 10.1038/s41598-025-90832-4.

[27] D. P. Gottumukkala, P. R. Prasad, and S. K. Rao, “Topic modeling-based prediction of software defects and root cause using BERTopic, and multioutput classifier,” *Sci Rep*, vol. 15, no. 1, pp. 1–20, 2025, doi: 10.1038/s41598-025-11458-0.

[28] X. Huang and J. Marques-Silva, “On the failings of Shapley values for explainability,” *International Journal of Approximate Reasoning*, vol. 171, pp. 1–57, 2024, doi: 10.1016/j.ijar.2023.109112.

[29] A. B. Nasser *et al.*, “Depth linear discrimination-oriented feature selection method based on adaptive sine cosine algorithm for software defect prediction,” *Expert Syst Appl*, vol. 253, no. February, p. 124266, 2024, doi: 10.1016/j.eswa.2024.124266.

[30] P. Yuen, P. Chan, J. Keung, and Z. Yang, “The Journal of Systems & Software Identifying inconsistent software defect predictions with symmetry metamorphic relation pattern,” *J Syst Softw*, vol. 227, no. February, p. 112449, 2025, doi: 10.1016/j.jss.2025.112449.

[31] H. A. Alhija, M. Azzeh, and F. Almasalha, “Software Defect Prediction Using Support Vector Machine,” *International Journal of Systematic Innovation*, vol. 7, no. 2, pp. 37–47, 2022, doi: 10.6977/IJoSI.202206_7(2).0003.