Journal of Electronics, Electromedical Engineering, and Medical Informatics
Multidisciplinary: Rapid Review: Open Access Journal **Vol. 6, No. 2, July 2024, pp: 289-301; eISSN: 2656-8632**

RESEARCH ARTICLE **OPEN ACCESS**

Manuscript received May 2, 2024; revised May 23, 2024; accepted May 27, 2024; date of publication July 8, 2024
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v6i3.455

How to cite: Wildan Nur Hidayatullah, Rudy Herteno, Mohammad Reza Faisal, Radityo Adi Nugroho, Setyo Wahyu Saputro, and Zarif 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, July 2024.

A Comparative Analysis of Polynomial-fit-SMOTE Variations with Tree-Based Classifiers on Software Defect Prediction

Wildan Nur Hidayatullah[1], Rudy Herteno[1], Mohammad Reza Faisal[1], Radityo Adi Nugroho[1], Setyo Wahyu Saputro[1], and Zarif Bin Akhtar[2]

[1] Department of Computer Science, Mathematics and Natural Science Faculty, Lambung Mangkurat University, Banjarbaru, South Kalimantan, Indonesia
[2] Department of Engineering, University of Cambridge, Cambridge, United Kingdom

Corresponding author: rudy.herteno@ulm.ac.id

ABSTRACT Software defects present a significant challenge to the reliability of software systems, often resulting in substantial economic losses. This study examines the efficacy of polynomial-fit SMOTE (pf-SMOTE) variants in combination with tree-based classifiers for software defect prediction, utilising the NASA Metrics Data Program (MDP) dataset. The research methodology involves partitioning the dataset into training and test subsets, applying pf-SMOTE oversampling, and evaluating classification performance using Decision Trees, Random Forests, and Extra Trees. Findings indicate that the combination of pf-SMOTE-star oversampling with Extra Tree classification achieves the highest average accuracy (90.91%) and AUC (95.67%) across 12 NASA MDP datasets. This demonstrates the potential of pf-SMOTE variants to enhance classification effectiveness. However, it is important to note that caution is warranted regarding potential biases introduced by synthetic data. These findings represent a significant advancement over previous research endeavors, underscoring the critical role of meticulous algorithm selection and dataset characteristics in optimizing classification outcomes. Noteworthy implications include advancements in software reliability and decision support for software project management. Future research may delve into synergies between pf-SMOTE variants and alternative classification methods, as well as explore the integration of hyperparameter tuning to further refine classification performance.

INDEX TERMS Polynomial-Fit-SMOTE, Decision Tree, Random Forest, Extra Trees, Software Defect Prediction.

I. INTRODUCTION

The increasing complexity of modern software in the current era has heightened the importance of software reliability due to the frequent occurrence of defects and potential failures [1]. Software defects, manifesting as bugs, errors, and inconsistencies in the source code, requirements, or design, can severely compromise software quality and reliability [2]. Defects in software can lead to failures that prevent desired outcomes and cause significant economic losses to organizations [3]. These losses can result in substantial financial losses for the company or even threaten the safety of human lives [4]. Throughout the development and implementation of a software project, project managers employ software quality assurance techniques, such as software testing and code inspection, with the objective of identifying and rectifying as many defects as possible [3]. It is estimated that more than 80% of the costs associated with software development and maintenance are dedicated solely to defect correction alone, which could be reduced if software defects could be detected early without requiring additional cost and effort [4]. For instance, NASA's $125 million Mars Climate Orbiter (MCO) spacecraft crashed in 1998 due to a minor data conversion error [5] or unit of measure inconsistency, marking one of the most disastrous incidents in the history of the software industry [6]. Such incidents underscore the critical need for effective software defect prediction techniques to ensure the timely delivery of reliable and cost-effective software products.

Software defect prediction is prediction techniques that use models that combine software modules and their labels

(defective or non-defective) to predict defects, intending to increase software reliability and reduce development costs [7]. The software defect prediction process involves three primary steps: collecting a historical defect data set, training a regression or classification model using deep learning or machine learning techniques on the historical data, and using the trained model to predict the number or probability of software defects [8]. Machine learning models, including Naïve Bayes (NB), decision trees (DT), K-nearest Neighbors (k-NN), Support Vector Machines (SVM), and Artificial Neural Networks (ANN), are frequently employed to identify software defects, leveraging their automated defect pattern recognition capabilities within software data [9]. Furthermore, recent investigations have explored the integration of tree-based ensemble learning methods, such as Extra Trees (ET), and Random Forest (RF), to enhance software defect prediction within machine learning classifiers [10].

However, highly imbalanced datasets can disrupt algorithmic classification by prioritizing the majority class with high data weights, resulting in poor performance on minority sets [11]. An imbalanced dataset occurs when the number of samples in one class (minority class) is three times smaller than the number of samples in another class (majority class). To address this issue, resampling techniques are employed, such as oversampling (the addition of samples from the minority class) or undersampling (the removal of samples from the majority class), in order to achieve balanced data and to improve classification performance [12]. SMOTE, the most prevalent oversampling technique, has been demonstrated to be an effective approach for addressing data imbalance, and it is a primary approach for tackling imbalanced learning problems [2], [3], [9] - [30].

In Research [31], the K2 algorithm, which employs a greedy approach for automating Bayesian Network structure learning, achieved accuracies of 0.9183 on CM1, 0.8079 on JM1, and 0.8483 on KC1. The hill-climbing algorithm, which began with a randomly generated Bayesian network, yielded comparable accuracies of 0.9183 on CM1, 0.8079 on JM1, and 0.8862 on KC1. The TAN algorithm, a tree-augmented naïve Bayesian network, yielded accuracies of 0.92 on CM1, 0.8236 on JM1, and 0.8815 on KC1. Another study [9] optimized algorithms with hyperparameter tuning, random search, PCA for dimensionality reduction, and SMOTE oversampling. The k-nearest neighbor (k-NN) model achieved accuracies of 0.9649 on CM1, 0.7791 on JM1, and 0.791 on KC1. The SVM model achieved 0.9766 on CM1, 0.6735 on JM1, and 0.7184 on KC1, while the SHL-MLP model reached 0.9708 on CM1, 0.7109 on JM1, and 0.7337 on KC1. In a separate study [10], applying SMOTE oversampling, the Decision Tree model demonstrated an accuracy of 0.8228, 0.7808, 0.691, 0.9768, and 0.8859 and AUC of 0.82, 0.78, 0.69, 0.98, and 0.89 on CM1, JM1, KC1, MC1, and PC1, respectively. The Random Forest model demonstrated accuracies of 0.9123, 0.8392, 0.7255, 0.9897, and 0.9379, accompanied by AUCs of 0.97, 0.91, 0.79, 1, and 0.99. The Extra Trees model achieved accuracies of 0.9053, 0.8319, 0.7307, 0.9928, and 0.9519, with AUCs of 0.98, 0.91, 0.8, 1, and 0.99 on the same datasets.

In a comprehensive analysis of 85 minority oversampling techniques across 104 imbalanced datasets, Kovacs [16] identified the top 10 oversampling techniques based on their combined performance scores. The methods mentioned above include Polynom-fit-SMOTE [17], ProWSyn [18], SMOTE-IPF [19], Lee [20], SMOBD [21], G-SMOTE [22], CCR [23], LVQ-SMOTE [24], Assembled-SMOTE [25], and SMOTE-TomekLinks [26]. It is regrettable that despite its ranking among the top ten, the detailed results of the four approaches proposed by the polynomial-fit-SMOTE method [17], including star topology, bus topology, polynomial curve topology, and mesh topology, were not presented in the research [16] or in several other comparative evaluations that assessed oversampling methods [27], [28], [29].

The objective of this research is to examine the effectiveness of variations of Polynomial-fit-SMOTE when employed in conjunction with tree-based classifiers. The tree-based algorithms utilized in this study encompass Decision Tree, Random Forest, and Extra Trees, as all three algorithms are rooted in the tree structure. Additionally, the Random Forest and Extra Trees algorithms, which are ensembles of several decision tree algorithms, are included in this investigation. This research direction is motivated by the prominence of Polynomial-fit-SMOTE, its first ranking among the top 10 oversampling techniques in the study [16], and the lack of detailed exploration of its methods. This investigation addresses the aforementioned gap by examining in detail the performance of Polynomial-fit-SMOTE when combined with tree-based classifiers. The smote-variants package [30], which provides Python implementations of 85 oversampling techniques, was used to facilitate this research.

The study aims to enhance software defect prediction accuracy and AUC metrics by integrating various methods. The anticipated outcomes include:

a. Determining the optimal tree-based classifier in conjunction with polynomial-fit-SMOTE oversampling, offering valuable insights into effective classification strategies for software defect prediction.
b. Investigating the impact of oversampling techniques on dataset characteristics and predictive performance, particularly focusing on the NASA MDP dataset pre- and post-oversampling with polynomial-fit-SMOTE.
c. Providing a comprehensive analysis of the combined performance of Polynomial-fit-SMOTE and Tree-Based Classifiers in enhancing classification accuracy and AUC metrics.
d. Exploring the potential implementation of these methods in software defect prediction to achieve more specific and optimal results.

II. MATERIAL AND METHODS

The proposed research methodology, as depicted in FIGURE 1, employs a machine learning model that utilizes tree-based classification techniques, including Decision Tree, Random

Forest, and Extra Trees, to address data imbalances. The three algorithms are classified as tree-based classification algorithms due to their similar classification performance characteristics and utilization of tree structures in their methodologies. This is achieved by employing variations of the polynomial-fit-SMOTE (pf-SMOTE) algorithm based on network topology, namely polynomial-fit-SMOTE-Bus (pf-SMOTE-Bus), polynomial-fit-SMOTE-Mesh (pf-SMOTE-Mesh), polynomial-fit-SMOTE-Poly (pf-SMOTE-Poly), and polynomial-fit-SMOTE-Star (pf-SMOTE-Star). The procedural workflow commences with the partitioning of the NASA MDP dataset version D" [32], detailed in TABLE 1 into distinct training and testing subsets, distributed randomly at an 8:2 ratio. Subsequently, the training dataset undergoes two scenarios: scenario 1 for classification without data balancing, and scenario 2 for classification combined with pf-SMOTE Variation. Subsequently, the performance of the classifiers is evaluated through repeated stratified k-fold cross-validation with 5 splits and 3 repeats, following the methodology proposed by [16], [27]. The assessment encompasses cross-validation with four pf-SMOTE variants and the three tree-based classifiers. The evaluation metrics utilized to compare the outcomes include accuracy and AUC values.

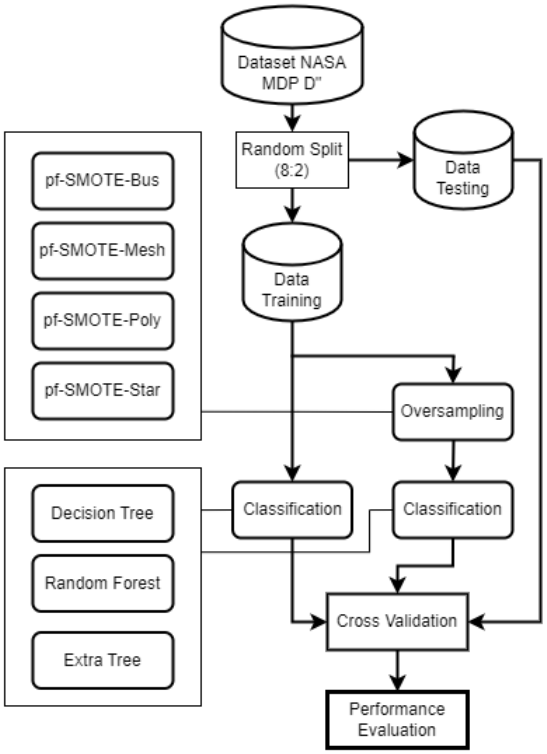


FIGURE 1. Flowchart of Research Methods

A. DATASET

This research employs the NASA Metrics Data Program (MDP) dataset, which was selected due to its status as a standard dataset in software defect prediction research. This dataset encompasses a range of software metrics features that are pertinent for the identification of defect-prone modules. Each dataset comprises features and corresponding output classes, categorised as positive (defect-prone) or negative (defect-free). The dataset utilized is an enhanced version, designated as DS" or D", obtained from the [32] https://github.com/klainfo/NASADefectDataset. However, it should be noted that this dataset is subject to certain limitations. These include the presence of noisy attributes, the high dimensionality of the dataset, and the imbalanced class records, whereby one class is significantly more numerous than the other. These factors can lead to biased models that perform poorly on minority class prediction [33].

TABLE 1
NASA MDP D" Datasets [32]

Dataset	Attributes	Modules	Defective	Non-Defective	Defective (%)
CM1	38	327	42	285	12.8
JM1	22	7,720	1,612	6,108	20.8
KC1	22	1,162	294	868	25.3
KC3	40	194	36	158	18.5
MC1	39	1,952	36	1,916	1.8
MC2	40	124	44	80	35.4
MW1	38	250	25	225	10
PC1	38	679	55	624	8.1
PC2	37	722	16	706	2.2
PC3	38	1,053	130	923	12.3
PC4	38	1,270	176	1,094	13.8
PC5	39	1,694	458	1,236	27.0

B. POLYNOMIAL-FIT-SMOTE OVERSAMPLING

Introduced in 2008 [17], the Polynomial-Fit-SMOTE (pf-SMOTE) algorithm provides multiple oversampling approaches tailored to the underlying network topologies of the minority class. Synthetic instances are produced using the Curve Fitting Method to determine coefficients for the polynomial $p(x)$ of degree n, aligning with the minority instances. This algorithm presents four specific network topologies used to synthesize samples [18], which will be detailed as follows:

1. BUS TOPOLOGY

In the bus topology method (FIGURE 2a), pathways linking minority data points to their nearest neighbors are constructed using straight lines. These pathways are utilized to choose synthetic samples along this line. Initially, a straight line connecting one minority data point to the next is plotted for each feature of the minority class matrix. The line connecting two consecutive data points is described by the following linear function (Eq. (1)) [17] :

$$f_i(x) = ax + b \tag{1}$$

Subsequently, the coefficients "a" and "b" are determined such that the function $f(x)$ accurately fits the data. Then, a set of k linearly-spaced value x_k ($k \in [-1,1]$) is generated based on the oversampling rate. Finally, new synthetic minority samples are generated between each pair of consecutive data points by evaluating the value of the function f_i at x_k [17].

2. MESH TOPOLOGY

In the mesh topology approach (FIGURE 2b), synthetic samples are generated by plotting straight lines connecting

each minority data point to all others within the minority class matrix. This process is analogous to that employed in the bus topology, where synthetic data is added along each line connecting two consecutive data points but with more connecting lines [17].

3. POLYNOMIAL CURVE TOPOLOGY

In the polynomial curve topology approach (FIGURE 2c), each feature within the minority class matrix is fitted to a polynomial curve that best represents the "trend curve" observed in the instances, and generates synthetic samples along this curve. This process involves computing the coefficients of a polynomial p(x) of degree n using the method of least square error using Eq. (2) [17].

$$p(x) = p_1 x^n + p_2 x^{n-1} + p_3 x^{n-2} + \cdots + p_n x^n + p_{n+1} \quad (2)$$

Subsequently, synthetic samples are generated along the curve of these polynomials, with the polynomial degree empirically determined to optimize the true positive (*TP*) rate [17].

4. STAR TOPOLOGY

In star topology approach (FIGURE 2d), involves generating synthetic samples along straight lines that connect each minority class data point to the mean of all the data points. In this method, a matrix of size $n \times m$, representing n *feature values* and m *instances*, is considered. Next, a group of linear functions $f_i(x)$ is defined to connect each feature value to the Eq. (3) [17]

$$\text{mean of all ones: } \begin{cases} f_i(x) = a_1 x + b_1 \\ \cdots \\ f_n(x) = a_n x + b_n \end{cases} \quad (3)$$

The coefficients a_i and b_i are then determined such that each function $f_i(x)$ accurately fits the mean value and the corresponding data points. Following this, k linearly-spaced value x_k ($k \in [-1,1]$) is generated according to the oversampling rate, and the new synthetic minority samples are generated by evaluating the value of the function f_i at each x_k. [17].

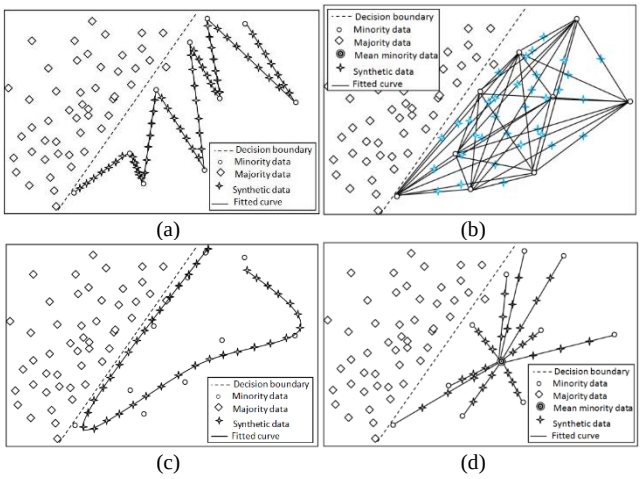


FIGURE 2. Polynom-fit-SMOTE oversampling base on network topologies using: (a) Bus topology, (b) Mesh topology, (c) Polynomial Curve topology, and (d) Star topologyIs [17].

The selected techniques were chosen for their superiority to other oversampling techniques, including Polynomial-fit-SMOTE, which ranked first among the top 10 oversampling techniques in this study [16]. Polynomial-fit-SMOTE was selected for its ability to generate synthetic samples that are more representative of minority groups, reduce the risk of overfitting, and improve classification performance under imbalanced data conditions.

C. CLASSIFICATION

Supervised machine learning methods rely on established training datasets to forecast class labels for novel data. In the context of defect prediction, this approach involves the discernment of flawed systems by categorizing modules as either defect-prone or not [34]. This research uses three classifiers with their performance characteristics based on tree structures in their methodology, Tree-based classifiers, include Decision Trees (DT), Random Forests (RF), and Extra Trees (ET) for the analysis..

1. DECISION TREE CLASSIFICATION

The Decision Tree (DT) method is a logic-driven learning approach that arranges instances based on their feature values using a tree-shaped structure (see FIGURE 3) [35]. Each node in the tree represents a feature in an instance for classification, while each branch indicates a potential value for that feature. Instance classification begins at the root node, organizing instances according to their feature values. Visual demonstrations of the Decision Tree algorithm are provided in FIGURE 4.

DT algorithm is among the top ten in machine learning and has several enhanced versions, including CART, CHAIR, ID3, and C4.5, all of which are derived from the fundamental DT algorithm [36]. C4.5, in particular, is widely utilized. In the context of C4.5, let S denote the training dataset, and $|S|$ represent the number of samples within it. The term $freq(C_j, S)$ signifies the count of samples belonging to class j within S. The average information entropy for a given class is expressed using Eq (4) [36]:

$$Info(S) = -\sum_{j=1}^{k} \frac{freq(C_j,S)}{|S|} * log_2\left(\frac{freq(C_j,S)}{|S|}\right) \quad (4)$$

Hence, for the attribute A, the requisite information to partition the dataset S into n distinct subsets $\{S_j\}$ is determined using Eq (5) [36]:

$$Info(A,S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} * Info(S_i) \quad (5)$$

The Information Gain Ratio is computed using Eq (6) [36]:

$$GainRatio(A) = \frac{Gain\ (A)}{splitInfo(A)} = \frac{Info(A) - Info(A,S)}{splitInfo(A,S)} \quad (6)$$

The optimal attribute can be identified by assessing the information entropy ratio and the gain ratio of potential attributes. This allows the generation of a branch for each possible attribute while selecting the root node, employing the maximum gain ratio as a split criterion.

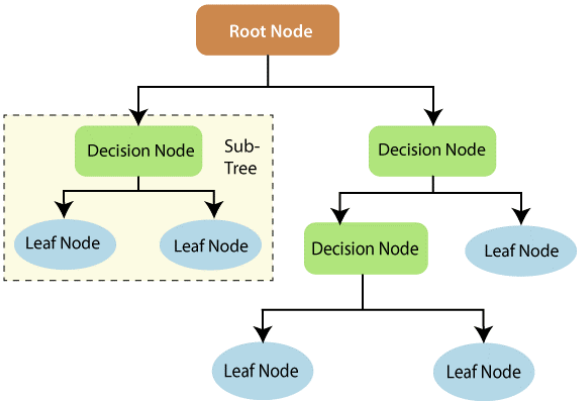


FIGURE 3. Decision tree algorithm illustration [37].

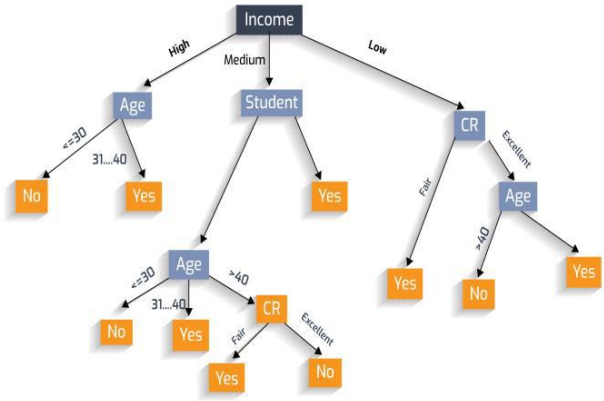


FIGURE 4. Decision tree algorithm example [37].

Decision trees, characterized by their straightforward yet resilient tree structure, are pervasively employed in domains such as machine learning [37]. Despite their efficacy in addressing nonlinear relationships, they are susceptible to noisy data and prone to overfitting [35].

2. RANDOM FOREST CLASSIFICATION

Random Forest (RF) classifier, introduced by Leo Breiman in 2001 [38], is a set of tree-based classifiers $\{h(\boldsymbol{x}, \Theta_k), k = 1, \ldots,\}$ where the $\{\Theta_k\}$ are independent and identically distributed random vectors. Each tree contributes a single vote for the most prevalent class at input **x** [38]. RF is a collection of small decision trees assembled into an ensemble, where each tree is constructed using random subsets of the dataset [10]. RF is renowned for its stability and effective handling of imbalanced data because rather than using the original sample, multiple decision trees are constructed using bootstrap samples, a technique referred to as bootstrap aggregating or bagging. This enhances generalization and minimizes overfitting [39]. The general model of the random forest is illustrated in FIGURE 5.

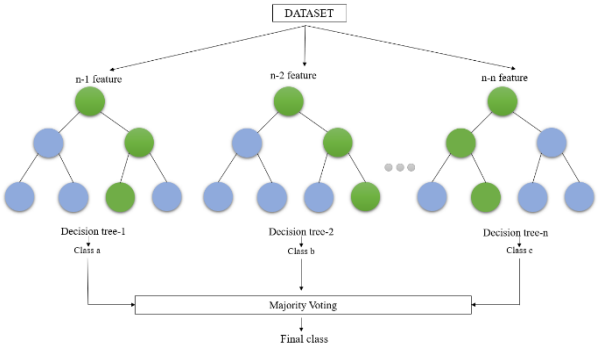


FIGURE 5. Random Forest Structure illustration [40].

The algorithm functions as follows in TABLE 2:

TABLE 2
Random Forest Algorithm [39]

```
for i ← 1 to B do
    bootstrap_sample ← draw_bootstrap_sample(N, training_data);
    while node_size ! = minimum_node_size do
        predictor_subset ← randomly_select_predictor_subset(m, p);
        for j ← 1 to m do
            if jth_predictor_optimizes_splitting_criterion(j,
            predictor_subset) then
                split_internal_node_into_two_child_nodes();
                break;
            end
        end
    end
end
return ensemble_tree_of_all_B_subtrees_generated();
```

3. EXTRA TREES CLASSIFICATION

The Extra Trees (ET), Extremely Randomized Trees algorithm, introduced by Pierre Geurts et al. in 2006 [41], builds an ensemble by creating a series of unpruned decision or regression trees using traditional top-down methods. It integrates the selection of attributes and cutpoints during node splitting in a highly randomized manner, potentially resulting in the generation of fully randomized trees with structures unrelated to the original values of the training sample [42]. ET algorithm consists of multiple decision trees, each tree having a root node, split nodes, and leaf nodes, as shown in FIGURE 6 [43]. Given a data set X, ET starts at the root node, where it selects a splitting criterion based on a random subset of features and a partially random cutoff point. This process continues at each child node until a leaf node is reached. In addition, the key parameters of ET include the number of trees in the ensemble (k), the number of randomly selected attributes/features (f), and the minimum number of samples/instances required to split a node (n_{min}).

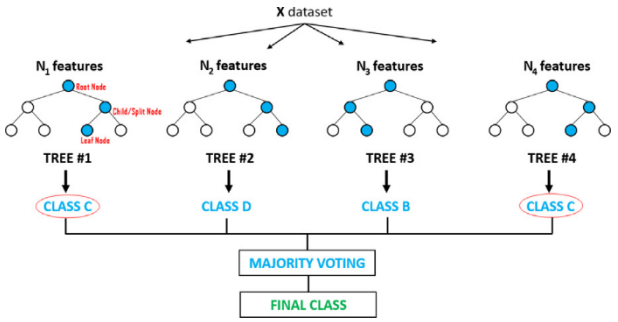


FIGURE 6. Extra Trees Structure illustration [43].

ET algorithm, along with the methods for splitting the nodes for both numeric and categorical attributes, is shown in detail in TABLE 3.

TABLE 3
Extra Trees Algorithm [41]

BuildExtraTreeEnsemble(S).

Input: training set S.
Output: a tree ensemble $T = \{t_1, \dots, t_m\}$
for $i = 1$to M
  Generate a tree: t_i = **BuildExtraTree**(S);
end
return T

BuildExtraTree(S).

Input: training set S.
Output: a tree t
Return a leaf labeled by class frequencies in S **if**
  (i) $|S| < n_{min}$, or
  (ii) all candidate attributes are constant in S,or
  (iii) the output variable is constant in S
Otherwise:
  1. Select randomly K attributes, $\{a_1, \dots, a_k\}$, without replacement, among all (non constant in S) candidate attributes;
  2. Generate K splits $\{s_1, \dots, s_k\}$, where s_i = **PickRandomSplit**(S, a_i), $\forall_i = 1, \dots, K$;
  3. Select a split s_* such that Score$(s_*, S) = max_{i=1,\dots,K}$ Score(s_i, S) ;
  4. Split S into subsets S_l and S_r according to the test s_*;
  5. Build t_l = **BuildExtraTree**(S_l) and t_r = **BuildExtraTree**(S_r) from these subsets;
  6. Create a node with the split s_*, attach t_l and t_r as left and right subtrees of this node and return the resulting tree t.

PickRandomSplit (S, a)

Input: training set S and attribute a.
Output: a split
if the attribute a is numerical:
  Compute the maximal and minimal value of a in S, denoted respectively by a^s_{min} and a^s_{max};
  Draw a cut-point a_c uniformly in $[a^s_{min}, a^s_{max}]$;
  return the split $[a < a_c]$.
if the attribute a is categorical (denote by A its set of possible values):
  Compute A_s the subset of A of values of a that appear in S;
  Randomly draw a proper non empty subset A_1 of A_s and a subset A_2 of $A \backslash A_s$;
  return the split $[a \in A_1 \cup A_2]$
end

As shown in FIGURE 5 and FIGURE 6, while RF and ET may appear similar based on algorithm illustrations, ET differs from RF by increasing randomness in two aspects: (1) it uses the entire dataset to construct each decision tree, and (2) it randomly selects splits at each node [10]. The three algorithms in question offer a satisfactory balance between interpretability, accuracy, and computational efficiency, rendering them well-suited for software defect prediction tasks.

D. EVALUATION

1. CONFUSION MATRIX

The Confusion Matrix serves as a central tool for evaluating the effectiveness and accuracy of classification algorithms, which is critical for understanding their performance in the face of model building and data preprocessing challenges, especially in high-dimensional data sets [44]. To illustrate the performance of the classifiers, a confusion matrix was used for the binary classification model, which provides a concise summary of the model's prediction results. The matrix is shown in TABLE 4 [40].

TABLE 4
Confusion Matrix [40]

	Actually Positive	Actually Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Using the confusion matrix, various evaluation metrics including the Accuracy in Eq. (9) and AUC in Eq.(11) can be calculated using the formula [45], [46], [47].

- Sensitivity (SN) (Eq. (7)) [45], also called True Positive Rate (TPR) or Recall, measures the ability of a model to predict positive cases, calculated as the ratio of true positive predictions to the total number of actual positive cases.

$$SN = TPR = recall = \frac{TP}{TP+FN} \quad (7)$$

- Specificity (SP) (Eq. (8)) [45], also called True Negative Rate (TNR), refers to the ability of the model to predict negative cases, calculated as the ratio of true negative predictions to the total number of actual negative cases.

$$SP = TNR = \frac{TN}{TN+FP} \quad (8)$$

- Accuracy (ACC) (Eq. (9)) [46], is the percentage of correct predictions and indicates the ability of the classifier to predict a condition effectively.

$$ACC = \frac{TP+TN}{TP+FN+TN+FP} \quad (9)$$

- False Positive Rate (FPR) (Eq. (10)) [46], is a measure of how often a classification model incorrectly predicts positive cases. It is calculated as the ratio of false positive predictions to the total number of actual negative cases..

$$FPR = 1 - SP = \frac{FP}{TN+FP} \quad (10)$$

2. AREA UNDER CURVE (AUC)

Area Under the Curve (AUC) (Eq. (11)) [45], a metric for comparing classifiers, quantifies the area under the receiver operating characteristic (ROC) curve in ROC space, where the curve plots sensitivity versus specificity relative to a discrimination threshold [45]. ROC curve shown in FIGURE 7 [46] uses the False Positive Rate (FPR) as the X-axis and the

True Positive Rate (TPR) as the Y-axis. Different decision thresholds are required at each point to produce different FPR and TPR values.

$$AUC = \frac{Sensitivity * Specificity}{2} \quad (11)$$

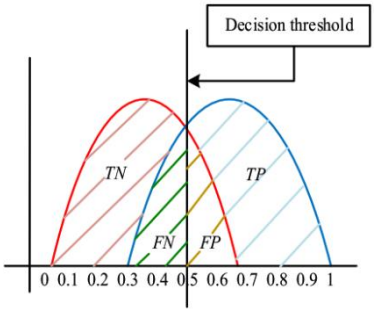


FIGURE 7. ROC curve where both TP and FP will change when the decision threshold is adjusted. [46].

Here is a reference for categorizing test accuracy based on the AUC value [44]. The classification of AUC values is shown in TABLE 5.

TABLE 5
Classification score accuracy is based on auc value [44]

AUC Value	Category
0.90 – 1.00	Excellent classification
0.80 – 0.90	Good classification
0.70 – 0.80	Fair classification
0.60 – 0.70	Poor classification
0.50 – 0.60	Failure classification

III. RESULTS

This study employs tree-based classification approach on the NASA MDP dataset, which includes 12 modules. The evaluation combines these classifiers with four Polynomial-fit-SMOTE variations, assessing each combination based on Accuracy and AUC metrics.

A. POLYNOM-FIT-SMOTE PROCESS

The oversampling procedure uses four pf-SMOTE variants, applied to the training data divided into two sets: Training Data and Testing Data, with an 80:20 ratio. Detailed results of the dataset, both before and after oversampling, are shown in TABLE 6. The *Minority Sample* and *Majority Sample* sets shown represent training data samples before implementing pf-SMOTE.

TABLE 6
Detailed results of 12 Nasa MDP datasets before and after applying each each pf-SMOTE variation.

Data Sample	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Minority Sample (before)	34	1337	251	29	37	35	22	49	13	107	142	337
Majority Sample (before)	227	4888	695	126	1553	65	180	515	583	754	887	991
Minority sample after applying oversampling												
Pf-SMOTE-Bus	232	5345	751	113	1513	69	169	529	541	743	847	1129
pf-SMOTE-Mesh	227	4888	695	126	1553	65	180	515	583	754	887	991
pf-SMOTE-Poly	227	4888	695	126	1553	65	180	515	583	754	887	991
pf-SMOTE-Star	238	5348	753	116	1554	70	176	539	585	749	852	1131

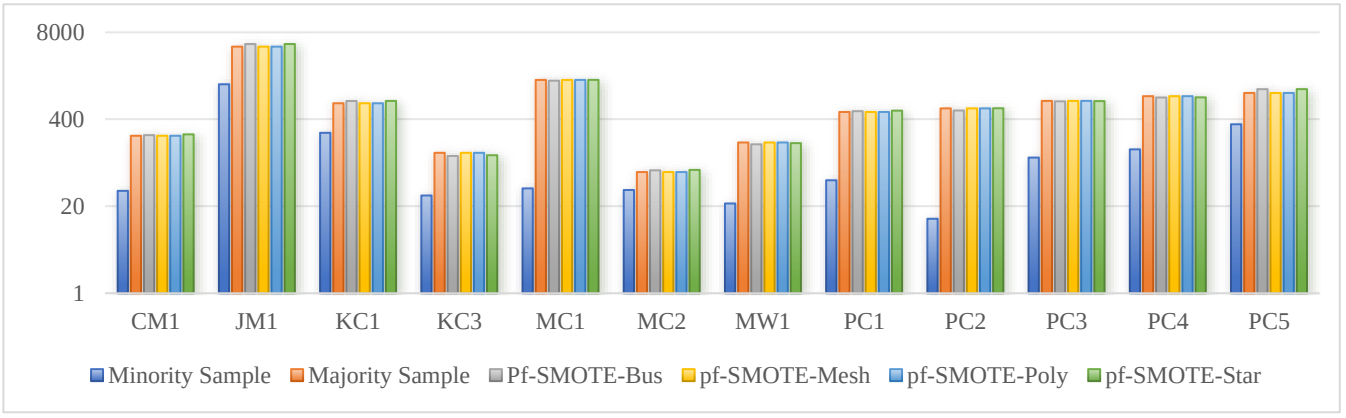


FIGURE 8. Comparison results from 12 datasets are provided both before and after applying each pf-SMOTE variation.

B. PERFORMANCE OF TREE-BASE CLASSIFICATION

The tree-based classification model is run on a dataset oversampled with pf-SMOTE. Accuracy values are shown in TABLE 7, while AUC values are shown in TABLE 8,

illustrating the performance of Decision Tree, Random Forest, and Extra Trees before and after oversampling with pf-SMOTE. The first three rows show the classification performance of Decision Trees, Random Forests and Extra Trees without pf-SMOTE, indicating the performance without data balancing. TABLE 7 and TABLE 8 show the improvements in accuracy and AUC for the three tree-based classifiers when the data set is balanced with four pf-SMOTE variants. Several instances in the tables show similar or identical accuracy and AUC values for different classifications combined with pf-SMOTE variations.

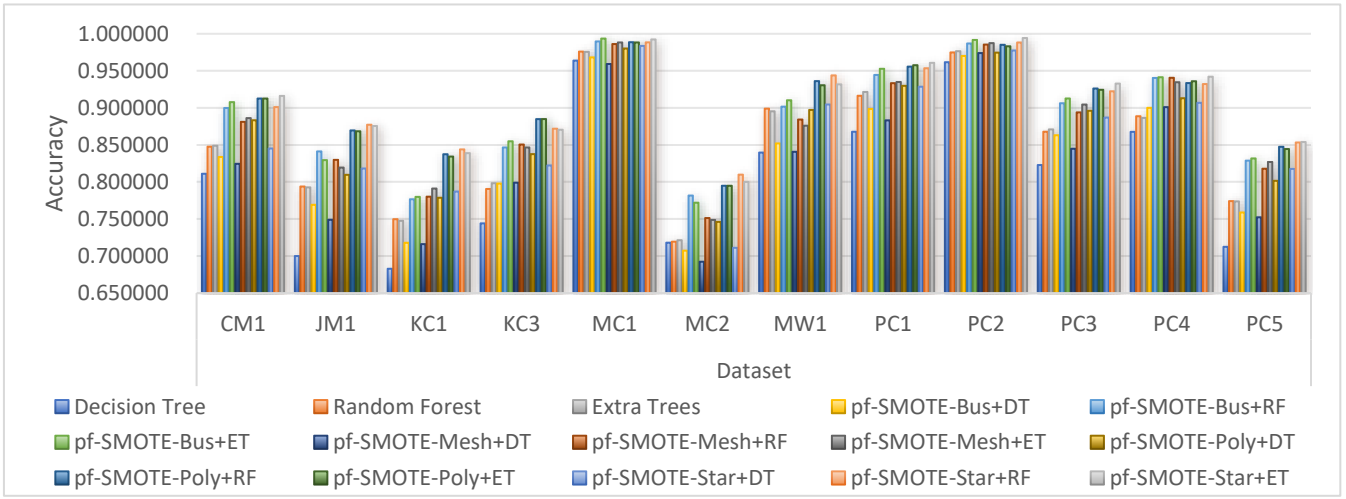
TABLE 7
Accuracy of Tree-base classification without and with polynomial-fit-SMOTE variant

Classifier	Dataset											
	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Decision Tree (base)	0.811026	0.699963	0.682791	0.744086	0.963857	0.718000	0.839675	0.867752	0.961639	0.822831	0.867663	0.712419
Random Forest (base)	0.847615	0.793842	0.749734	0.790538	0.975975	0.719333	0.898959	0.916334	0.974846	0.867826	0.888713	0.774121
Extra Trees (base)	0.848882	0.792675	0.747544	0.798280	0.975639	0.721333	0.895382	0.921522	0.976513	0.870923	0.886447	0.773776
pf-SMOTE-Bus + DT	0.833660	0.769145	0.717851	0.797872	0.968143	0.707217	0.852008	0.898473	0.970053	0.863058	0.900032	0.758805
pf-SMOTE-Bus + RF	0.899817	0.841233	0.776633	0.846543	0.989671	0.781671	0.901656	0.944458	0.986951	0.906261	0.940410	0.828774
pf-SMOTE-Bus + ET	0.907748	0.829473	0.779860	0.854905	**0.993585**	0.771985	0.910283	0.952763	0.991692	0.912705	0.941371	0.831761
pf-SMOTE-Mesh + DT	0.824510	0.748943	0.716067	0.798954	0.959219	0.692308	0.840741	0.883172	0.973984	0.844833	0.900974	0.752284
pf-SMOTE-Mesh + RF	0.881115	0.829821	0.780096	0.850536	0.986157	0.751282	0.884259	0.933333	0.985425	0.893895	0.940625	0.817869
pf-SMOTE-Mesh + ET	0.886203	0.819422	0.791127	0.846562	0.988303	0.748718	0.875926	0.934951	0.987419	0.904509	0.934614	0.826950
pf-SMOTE-Poly + DT	0.883272	0.809602	0.778657	0.837542	0.980147	0.746154	0.897222	0.929773	0.974557	0.896109	0.913005	0.801553
pf-SMOTE-Poly + RF	0.912617	0.869579	0.837410	0.884837	0.988625	0.794872	0.936111	0.955663	0.985131	0.926167	0.933487	0.847465
pf-SMOTE-Poly + ET	0.912617	0.868420	0.834293	**0.884889**	0.988303	0.794872	0.930556	0.957605	0.983127	0.924398	0.935930	0.844442
pf-SMOTE-Star + DT	0.845161	0.818158	0.787072	0.822307	0.983693	0.711111	0.904564	0.928540	0.977457	0.886898	0.906847	0.817630
pf-SMOTE-Star + RF	0.901075	**0.877230**	**0.843924**	0.871882	0.988414	**0.809877**	**0.943845**	0.953517	0.988297	0.922367	0.932147	0.853281
pf-SMOTE-Star + ET	**0.916129**	0.875538	0.838877	0.870493	0.992383	0.800000	0.931716	**0.960788**	**0.994291**	**0.932798**	**0.942115**	**0.853915**

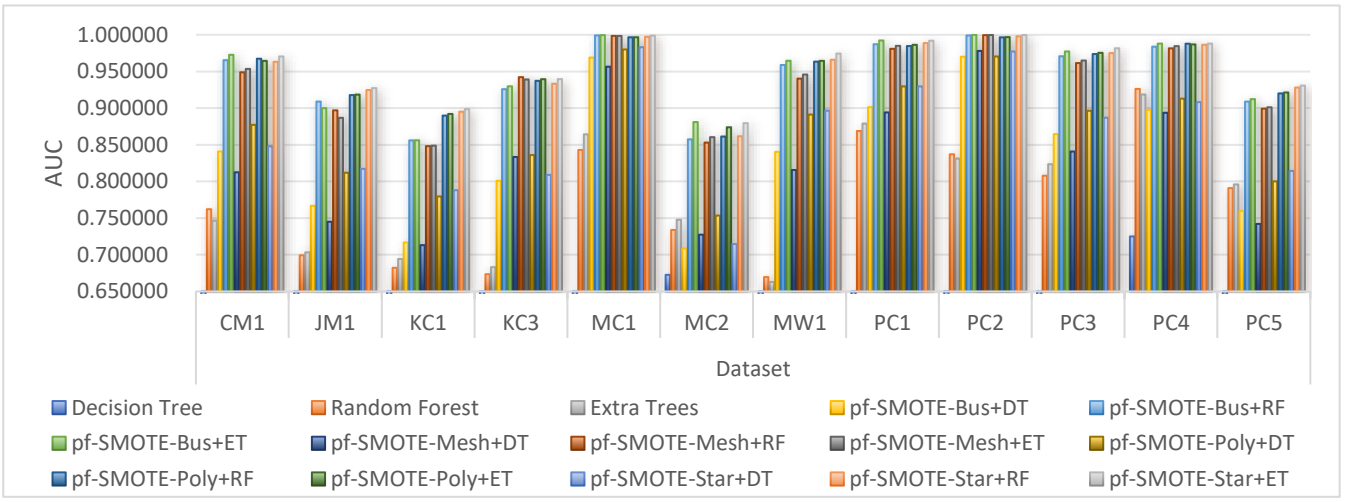
TABLE 8
AUC of Tree-base classification without and with polynomial-fit-SMOTE variant

Classifier	Dataset											
	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Decision Tree (base)	0.620777	0.571692	0.595859	0.559495	0.622316	0.672635	0.564938	0.636430	0.565847	0.611604	0.724994	0.642860
Random Forest (base)	0.762092	0.699264	0.682074	0.673454	0.842881	0.733758	0.669527	0.868803	0.837087	0.807795	0.926217	0.790883
Extra Trees (base)	0.746579	0.703265	0.694113	0.683030	0.864259	0.747748	0.662848	0.878868	0.831202	0.823515	0.918524	0.795944
pf-SMOTE-Bus+DT	0.840854	0.766693	0.716809	0.800771	0.968960	0.708473	0.840341	0.901284	0.970081	0.864504	0.897559	0.759530
pf-SMOTE-Bus+RF	0.965392	0.908969	0.855994	0.925812	0.999216	0.857354	0.958797	0.987160	0.999119	0.970722	0.983839	0.908916
pf-SMOTE-Bus+ET	**0.972701**	0.899993	0.856077	0.929730	**0.999597**	**0.881002**	0.964606	**0.992289**	**0.999944**	0.977290	**0.988010**	0.912242
pf-SMOTE-Mesh+DT	0.812486	0.745025	0.713113	0.833352	0.956507	0.727380	0.815600	0.894152	0.978029	0.840836	0.893709	0.742063
pf-SMOTE-Mesh+RF	0.948859	0.897014	0.848173	**0.942237**	0.998479	0.853012	0.940300	0.980892	0.999557	0.961465	0.981525	0.899318
pf-SMOTE-Mesh+ET	0.953447	0.886729	0.848836	0.939109	0.998457	0.860371	0.945819	0.984910	0.999722	0.964949	0.984549	0.901147
pf-SMOTE-Poly+DT	0.877178	0.811947	0.779514	0.835942	0.979984	0.753457	0.891232	0.929622	0.970460	0.896433	0.912904	0.800078
pf-SMOTE-Poly+RF	0.967392	0.917782	0.889822	0.937227	0.996633	0.861266	0.963453	0.984659	0.996518	0.973772	0.987855	0.920086
pf-SMOTE-Poly+ET	0.964451	0.918473	0.892049	0.939450	0.996722	0.873882	0.964497	0.986338	0.996860	0.975416	0.986934	0.921334
pf-SMOTE-Star+DT	0.847905	0.817174	0.788011	0.808912	0.983049	0.714620	0.896590	0.929626	0.977218	0.886892	0.908135	0.814316
pf-SMOTE-Star+RF	0.963215	0.924613	0.895187	0.933239	0.997304	0.861625	0.965983	0.988684	0.997928	0.975051	0.986435	0.928094
pf-SMOTE-Star+ET	0.970487	**0.927317**	**0.898596**	0.939566	0.998877	0.879500	**0.974462**	0.991938	0.999721	**0.981597**	0.987942	**0.930666**

TABLE 7 shows that using training data balanced with pf-SMOTE achieves a higher average accuracy compared to not using pf-SMOTE, where pf-SMOTE-Star + ET achieves the highest accuracy on the PC2 dataset compared to other variations of pf-SMOTE with an accuracy value of 99.43%. This is followed by the combination of pf-SMOTE-Bus + ET on MC1, pf-SMOTE-Poly + RF on MC1, and pf-SMOTE-Mesh + ET on MC1 with accuracy values of 99.36%, 98.86%, and 98.83%, respectively. Meanwhile, the highest AUC results as shown in TABLE 8 were achieved by the pf-SMOTE-Bus + ET combination with an accuracy value of 99.99%. This is followed by pf-SMOTE-Mesh + ET, pf-SMOTE-Star + ET, and then pf-SMOTE-Poly + ET with AUC values of 99.9722%, 99.9721%, and 99.69%, respectively. The highest overall AUC results are found in the same data set, namely PC2.



(a)



(b)

FIGURE 9. Comparison of the performance of tree-based classification without and with oversampling using different pf-SMOTE variants, including (a) accuracy performance and (b) AUC performance.

IV. DISCUSSION

TABLE 9 shows that the combination of pf-SMOTE-Star oversampling with ET classification on 12 Nasa MDP datasets achieved the highest average accuracy and AUC compared to all other tree-based classification and oversampling models, namely 90.91% and 95.67% for the oversampling method and other tree-based classifications. Then pf-SMOTE-Poly with RF achieved an average accuracy of 90.60% and with ET an average AUC of 95.14% compared to other classifications. pf-SMOTE-Mesh with ET achieved an average accuracy of 87.87% and with the same classification an AUC of 93.90% compared to other classifications. And pf-SMOTE-Bus with ET obtained the highest average accuracy of 88.98% and with

the same classification obtained the highest AUC of 94.78% compared to other classifications. This shows that on average, the combination classification model of pf-SMOTE variations with ET classification has a higher performance value. Even when comparing the ET classification before and after using pf-SMOTE-Star, the average accuracy value increased by 5.83% from the initial value of 85.07% and the average AUC increased by 17.76% from the initial value of 77.92%.

TABLE 9
Average performance across all datasets

Classifier	Performance	
	Accuracy	AUC
Decision Tree	0.8076	0.6158
Random Forest	0.8498	0.7745
Extra Trees	0.8507	0.7792
pf-SMOTE-Bus+DT	0.8364	0.8363
pf-SMOTE-Bus+RF	0.8870	0.9434
pf-SMOTE-Bus+ET	0.8898	0.9478
pf-SMOTE-Mesh+DT	0.8280	0.8294
pf-SMOTE-Mesh+RF	0.8779	0.9376
pf-SMOTE-Mesh+ET	0.8787	0.9390
pf-SMOTE-Poly+DT	0.8706	0.8699
pf-SMOTE-Poly+RF	0.9060	0.9497
pf-SMOTE-Poly+ET	0.9050	0.9514
pf-SMOTE-Star+DT	0.8658	0.8644
pf-SMOTE-Star+RF	0.9072	0.9514
pf-SMOTE-Star+ET	**0.9091**	**0.9567**

Table 10 presents a comparison between the model utilized in this research and that employed in a previous study. The earlier investigation involved a comparative assessment with three classification models: K2, Hill Climbing, and TAN. Furthermore, other research endeavors have utilized models that underwent hyperparameter tuning, random search, PCA for dimensionality reduction, and SMOTE oversampling. These models encompass k-NN, SVM, and SHL-MLP. The datasets CM1, JM1, and KC1 were utilized in this study.

TABLE 10
Comparison with other research

Research	Method	Datasets (Accuracy%)		
		CM1	JM1	KC1
[31]	K2	0.9183	0.8079	0.8483
	Hill Climbing	0.9183	0.8079	**0.8862**
	TAN	0.92	0.8239	0.8815
[9]	Optimized k-NN	0.9649	0.7791	0.7931
	Optimized SVM	**0.9766**	0.6735	0.7184
	Optimized SHL-MLP	0.9708	0.7109	0.7337
Proposed Research	pf-SMOTE-star + DT	0.8452	0.8182	0.7871
	pf-SMOTE-star + RF	0.9011	**0.8772**	0.8439
	pf-SMOTE-star + ET	0.9161	0.8755	0.8389

A comparison of the three preceding investigations reveals that each study demonstrates proficiency in a particular dataset. For instance, in [31] the Hill Climbing model attained the highest accuracy of 0.8862 on KC1. Similarly, [9] achieved the highest accuracy on CM1 with the optimized SVM model, recording a value of 0.9766. In contrast, the present study achieved the highest accuracy of 0.8772 on JM1 through data oversampling using pf-SMOTE-Star with the Random Forest classification model. In addition, Table 10 also illustrates that in [9], the optimized model achieved the highest average results on CM1 across all classifications.

Table 11 presents a comparison of the performance of the model utilized in this study with those from previous research. Previous studies have employed various methodologies, including Decision Trees (DT), Random Forests (RF), and Extra Trees (ET), in conjunction with SMOTE for oversampling. These studies have been conducted using the PC1, PC2, PC3, and PC4 datasets.

TABLE 11
Comparison with previous research

Datasets	Performance	Other Research Method with SMOTE [10]			Proposed Research with pf-SMOTE-Star		
		DT	RF	ET	DT	RF	ET
PC1	Accuracy	0.8859	0.9379	0.9519	0.9285	0.9535	**0.9608**
	AUC	0.89	0.99	0.99	0.9296	0.9887	**0.9919**
PC2	Accuracy	0.9678	0.9849	0.987	0.9775	0.9883	**0.9943**
	AUC	0.97	**1**	**1**	0.9772	0.9979	0.9997
PC3	Accuracy	0.8579	0.9051	0.9062	0.8869	0.9224	**0.9328**
	AUC	0.86	0.96	0.97	0.8869	0.9751	**0.9816**
PC4	Accuracy	0.9144	0.955	**0.9577**	0.9068	0.9321	0.9421
	AUC	0.91	0.99	**1**	0.9081	0.9864	0.9879

A comparison of the two previous studies indicates that different SMOTE techniques can significantly impact the performance of classification models. The use of pf-SMOTE-Star demonstrates a superior ability to address data imbalance,

potentially matching or enhancing the classification accuracy seen in earlier research. However, this improvement is not consistent across all datasets; for instance, the highest accuracy and AUC for the PC4 dataset were achieved by prior studies using SMOTE with the Extra Trees (ET) classification.

The comparison presented in TABLES 10 and TABLES 11 indicates that the utilization of pf-SMOTE-Star with tree-based classification does not universally outperform other methods, suggesting potential limitations. These constraints are, in part, attributed to the dataset's characteristics, which may hinder the findings' generalizability to broader software development contexts. Furthermore, the outcomes of the pf-SMOTE oversampling approach demonstrate a discrepancy in performance across methodologies, particularly between the topology-based and star topology approaches. This discrepancy may be attributed to various factors, including an uneven distribution of classes, sensitivity to algorithm parameter settings, or incomplete model coverage. These findings underscore the limitations of the framework and necessitate further validation to confirm their validity and evaluate their relevance in diverse software development contexts.

V. CONCLUSION

This study has investigated various oversampling techniques in conjunction with tree-based classification models for software defect prediction. The findings indicate that oversampling methods, such as pf-SMOTE-Star, pf-SMOTE-Poly, pf-SMOTE-Mesh, and pf-SMOTE-Bus, can significantly enhance classification performance. In particular, the combination of pf-SMOTE-Star oversampling with ET classification demonstrated promising results, achieving the highest average accuracy and AUC across multiple datasets, reaching 90.91% and 95.67%, respectively. However, it is important to note that the effectiveness of these techniques may vary depending on the characteristics of the dataset and the classification model used. The study also identified potential limitations, such as performance imbalances across different oversampling methodologies. Future research endeavors should concentrate on addressing these limitations and further validating the findings in diverse software development contexts. Furthermore, the integration of hyperparameter tuning techniques with pf-SMOTE-ET requires further investigation to achieve higher levels of classification performance. Overall, this study contributes to advancing the understanding of software defect prediction techniques and offers valuable insights for practitioners seeking to improve the reliability of software development projects.

ACKNOWLEDGMENT

We extend our profound gratitude to the entire community of the Computer Science department at Lambung Mangkurat University's Faculty of Mathematics and Natural Sciences for their generous support and invaluable resources, which greatly facilitated the completion of this study. Furthermore, we express our gratitude to our project colleagues for their dedication and teamwork, which were instrumental in the successful completion of this research endeavor.

REFERENCES

[1] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software Defect Prediction via Attention-Based Recurrent Neural Network," *Sci Program*, vol. 2019, 2019, doi: 10.1155/2019/6230953.

[2] A. Alsaeedi and M. Z. Khan, "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study," *Journal of Software Engineering and Applications*, vol. 12, no. 05, pp. 85–100, 2019, doi: 10.4236/jsea.2019.125007.

[3] X. Chen, D. Zhang, Y. Zhao, Z. Cui, and C. Ni, "Software defect number prediction: Unsupervised vs supervised methods," *Inf Softw Technol*, vol. 106, pp. 161–181, Feb. 2019, doi: 10.1016/j.infsof.2018.10.003.

[4] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Inf Softw Technol*, vol. 114, pp. 204–216, Oct. 2019, doi: 10.1016/j.infsof.2019.07.003.

[5] S. Goyal, "Handling Class-Imbalance with KNN (Neighbourhood) Under-Sampling for Software Defect Prediction," *Artif Intell Rev*, vol. 55, no. 3, pp. 2023–2064, Mar. 2022, doi: 10.1007/s10462-021-10044-w.

[6] L. Manservigi *et al.*, "Detection of Unit of Measure Inconsistency in gas turbine sensors by means of Support Vector Machine classifier," *ISA Trans*, vol. 123, pp. 323–338, Apr. 2022, doi: 10.1016/j.isatra.2021.05.034.

[7] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A Semantic LSTM Model for Software Defect Prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019, doi: 10.1109/ACCESS.2019.2925313.

[8] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, Apr. 2020, doi: 10.1016/j.neucom.2019.11.067.

[9] M. Z. F. N. Siswantoro and U. L. Yuhana, "Software Defect Prediction Based on Optimized Machine Learning Models: A Comparative Study," *Teknika*, vol. 12, no. 2, pp. 166–172, Jun. 2023, doi: 10.34148/teknika.v12i2.634.

[10] H. Aljamaan and A. Alazba, "Software defect prediction using tree-based ensembles," in *PROMISE 2020 - Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Co-located with ESEC/FSE 2020*, Association for Computing Machinery, Inc, Nov. 2020, pp. 1–10. doi: 10.1145/3416508.3417114.

[11] F. Thabtah, S. Hammoud, F. Kamalov, and A. Gonsalves, "Data imbalance in classification: Experimental evaluation," *Inf Sci (N Y)*, vol. 513, pp. 429–441, Mar. 2020, doi: 10.1016/j.ins.2019.11.004.

[12] X. W. Liang, A. P. Jiang, T. Li, Y. Y. Xue, and G. T. Wang, "LR-SMOTE — An improved unbalanced data set oversampling based on K-means and SVM," *Knowl Based Syst*, vol. 196, May 2020, doi: 10.1016/j.knosys.2020.105845.

[13] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," *Neurocomputing*, vol. 343, pp. 120–140, May 2019, doi: 10.1016/j.neucom.2018.04.090.

[14] D. Bajer, B. Zonć, M. Dudjak, and G. Martinović, "Performance Analysis of SMOTE-based Oversampling Techniques When Dealing with Data Imbalance," *2019 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2019, doi: 10.1109/IWSSIP.2019.8787306.

[15] A. Fernández, S. García, F. Herrera, and N. V Chawla, "SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary," 2018.

[16] G. Kovács, "An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets," *Applied Soft Computing Journal*, vol. 83, Oct. 2019, doi: 10.1016/j.asoc.2019.105662.

[17] S. Gazzah and N. E. Ben Amara, "New oversampling approaches based on polynomial fitting for imbalanced data sets," in *DAS 2008 - Proceedings of the 8th IAPR International Workshop on Document Analysis Systems*, 2008, pp. 677–684. doi: 10.1109/DAS.2008.74.

[18] S. Barua, Md. M. Islam, and K. Murase, "ProWSyn: Proximity Weighted Synthetic Oversampling Technique for Imbalanced Data Set Learning," *Lecture Notes in Computer Science*, vol. 7819, pp. 317–328, 2013, doi: https://doi.org/10.1007/978-3-642-37456-2_27.

[19] J. A. Sáez, J. Luengo, J. Stefanowski, and F. Herrera, "SMOTE-IPF: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering," *Inf Sci (N Y)*, vol. 291, no. C, pp. 184–203, 2015, doi: 10.1016/j.ins.2014.08.051.

[20] J. Lee, N. R. Kim, and J. H. Lee, "An over-sampling technique with rejection for imbalanced class learning," in *ACM IMCOM 2015 - Proceedings*, Association for Computing Machinery, Inc, Jan. 2015. doi: 10.1145/2701126.2701181.

[21] Q. Cao and S. Wang, "Applying over-sampling technique based on data density and cost-sensitive SVM to imbalanced learning," in *Proceedings - 2011 4th International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII 2011*, 2011, pp. 543–548. doi: 10.1109/ICIII.2011.276.

[22] T. Sandhan and J. Y. Choi, "Handling imbalanced datasets by partially guided hybrid sampling for pattern recognition," in *Proceedings - International Conference on Pattern Recognition*, Institute of Electrical and Electronics Engineers Inc., Dec. 2014, pp. 1449–1453. doi: 10.1109/ICPR.2014.258.

[23] M. Koziarski and M. Wozniak, "CCR: A combined cleaning and resampling algorithm for imbalanced data classification," *International Journal of Applied Mathematics and Computer Science*, vol. 27, no. 4, pp. 727–736, Dec. 2017, doi: 10.1515/amcs-2017-0050.

[24] M. Nakamura, Y. Kajiwara, A. Otsuka, and H. Kimura, "LVQ-SMOTE-Learning Vector Quantization based Synthetic Minority Over-sampling Technique for biomedical data," 2013. [Online]. Available: http://www.biodatamining.org/content/6/1/16

[25] B. Zhou, C. Yang, H. Guo, and J. Hu, "A Quasi-linear SVM Combined with Assembled SMOTE for Imbalanced Data Classification," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013. doi: 10.1109/IJCNN.2013.6707035.

[26] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004, doi: https://doi.org/10.1145/1007730.1007735.

[27] A. Islam, S. B. Belhaouari, A. U. Rehman, and H. Bensmail, "KNNOR: An oversampling technique for imbalanced datasets[Formula presented]," *Appl Soft Comput*, vol. 115, Jan. 2022, doi: 10.1016/j.asoc.2021.108288.

[28] S. Bej, K. Schulz, P. Srivastava, M. Wolfien, and O. Wolkenhauer, "A Multi-Schematic Classifier-Independent Oversampling Approach for Imbalanced Datasets," *IEEE Access*, vol. 9, pp. 123358–123374, 2021, doi: 10.1109/ACCESS.2021.3108450.

[29] T. Watthaisong, K. Sunat, and N. Muangkote, "Comparative Evaluation of Imbalanced Data Management Techniques for Solving Classification Problems on Imbalanced Datasets," *Statistics, Optimization and Information Computing*, vol. 12, no. 2, pp. 547–570, Mar. 2024, doi: 10.19139/soic-2310-5070-1890.

[30] G. Kovács, "Smote-variants: A python implementation of 85 minority oversampling techniques," *Neurocomputing*, vol. 366, pp. 352–354, Nov. 2019, doi: 10.1016/j.neucom.2019.06.100.

[31] M. J. Hernández-Molinos, A. J. Sánchez-García, R. E. Barrientos-Martínez, J. C. Pérez-Arriaga, and J. O. Ocharán-Hernández, "Software Defect Prediction with Bayesian Approaches," *Mathematics*, vol. 11, no. 11, Jun. 2023, doi: 10.3390/math11112524.

[32] A. Iqbal *et al.*, "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, pp. 300–308, 2019, doi: 10.14569/ijacsa.2019.0100538.

[33] M. N. M. Rahman, R. A. Nugroho, M. R. Faisal, F. Abadi, and R. Herteno, "Optimized multi correlation-based feature selection in software defect prediction," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 22, no. 3, pp. 598–605, Jun. 2024, doi: 10.12928/TELKOMNIKA.v22i3.25793.

[34] C. L. Prabha and N. Shivakumar, "Software Defect Prediction Using Machine Learning Techniques," in *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, 2020. doi: 10.1109/ICOEI48184.2020.9142909.

[35] H. Alsawalqah *et al.*, "Software defect prediction using heterogeneous ensemble classification based on segmented patterns," *Applied Sciences (Switzerland)*, vol. 10, no. 5, Mar. 2020, doi: 10.3390/app10051745.

[36] Z. Tian, J. Xiang, S. Zhenxiao, Z. Yi, and Y. Yunqiang, "Software Defect Prediction based on Machine Learning Algorithms," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, IEEE, 2020. doi: 10.1109/ICCC47050.2019.9064412.

[37] B. Charbuty and A. Abdulazeez, "Classification Based on Decision Tree Algorithm for Machine Learning," *Journal of Applied Science and Technology Trends*, vol. 2, no. 01, pp. 20–28, Mar. 2021, doi: 10.38094/jastt20165.

[38] L. Breiman, "Random Forests," *Mach Learn*, vol. 45, pp. 5–32, 2001, doi: https://doi.org/10.1023/A:1010933404324.

[39] M. Schonlau and R. Y. Zou, "The random forest algorithm for statistical learning," *The Stata Journal: Promoting communications on statistics and Stata*, vol. 20, no. 1, pp. 3–29, Mar. 2020, doi: 10.1177/1536867X20909688.

[40] H. B. Kibria and A. Matin, "The Severity Prediction of The Binary And Multi-Class Cardiovascular Disease -- A Machine Learning-Based Fusion Approach," *Comput Biol Chem*, vol. 98, Mar. 2022, doi: https://doi.org/10.1016/j.compbiolchem.2022.107672.

[41] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach Learn*, vol. 63, no. 1, pp. 3–42, Apr. 2006, doi: 10.1007/s10994-006-6226-1.

[42] E. K. Ampomah, Z. Qin, and G. Nyame, "Evaluation of tree-based ensemble machine learning models in predicting stock price direction of movement," *Information (Switzerland)*, vol. 11, no. 6, Jun. 2020, doi: 10.3390/info11060332.

[43] U. Saeed, S. U. Jan, Y. D. Lee, and I. Koo, "Fault diagnosis based on extremely randomized trees in wireless sensor networks," *Reliab Eng Syst Saf*, vol. 205, Jan. 2021, doi: 10.1016/j.ress.2020.107284.

[44] M. Fawwaz Akbar, M. I. Mazdadi, H. Saragih, and F. Abadi, "Implementation of Information Gain Ratio and Particle Swarm Optimization in the Sentiment Analysis Classification of Covid-19 Vaccine Using Support Vector Machine," *Journal of Electronics, Electromedical Engineering, and Medical informatics (JEEEMI)*, vol. 5, no. 4, pp. 261–270, 2023, doi: 10.35882/jeemi.v5i4.328.

[45] D. Valero-Carreras, J. Alcaraz, and M. Landete, "Comparing two SVM models through different metrics based on the confusion matrix," *Comput Oper Res*, vol. 152, Apr. 2023, doi: 10.1016/j.cor.2022.106131.

[46] C. Y. Lee and W. C. Lin, "Induction Motor Fault Classification Based on ROC Curve and t-SNE," *IEEE Access*, vol. 9, pp. 56330–56343, 2021, doi: 10.1109/ACCESS.2021.3072646.

[47] D. Chicco, N. Tötsch, and G. Jurman, "The matthews correlation coefficient (Mcc) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation," *BioData Min*, vol. 14, pp. 1–22, 2021, doi: 10.1186/s13040-021-00244-z.

AUTHOR BIOGRAPHY

Wildan Nur Hidayatullah was born in Samuda, Central Kalimantan, Indonesia. Since 2020, he has been engaged in the academic world as a student at the Department of Computer Science, Lambung Mangkurat University. His current field of research is in the field of software engineering. He selected this area of special interest because of his interest in the field of software engineering. Additionally, the project ultimately involves research centered on software defect prediction. The objective of this research is to assess the performance of polynomial-fit-SMOTE using tree-based classification in the context of software defects.

Rudy Herteno was born in Banjarmasin, South Kalimantan. After graduating from high school, he pursued his undergraduate studies in the Computer Science Department at Lambung Mangkurat University and graduated in 2011. After completing his undergraduate program, he worked as a software developer to gather experience for several years. He developed a lot of software, especially for local governments. In 2017, He completed his master's degree in Informatics from STMIK Amikom University. Currently, he is a lecturer in the Faculty of Mathematics and Natural Science at Lambung Mangkurat University. His research interests include software engineering, software defect prediction, and deep learning..

Mohammad Reza Faisal was born in Banjarmasin. Following his graduation from high school, he pursued his undergraduate studies in the Informatics department at Pasundan University in 1995, and later majored in Physics at Bandung Institute of Technology in 1997. After completing his bachelor's program, he gained experience as a training trainer in the field of information technology and software development. Since 2008, he has been a lecturer in computer science at Universitas Lambung Mangkurat, while also pursuing his master's program in Informatics at Bandung Institute of Technology in 2010. In 2015, he furthered his education by pursuing a doctoral degree in Bioinformatics at Kanazawa University, Japan. To this day, he continues his work as a lecturer in Computer Science at Universitas Lambung Mangkurat. His research interests encompass Data Science, Software Engineering, and Bioinformatics.

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 is also a practitioner in the field of information technology as a project manager and systems analyst to develop software and information systems used by universities.

Setyo Wahyu Saputro is a lecturer in the Computer Science Department, Faculty of Mathematics and Natural Science, Lambung Mangkurat University in Banjarbaru. He received a bachelor's degree also in Computer Science from Lambung Mangkurat University and received his master's degree in Informatics from STMIK Amikom University. His research interests include software engineering and artificial intelligence applications

Zarif Bin Akhtar studied at Master of Philosophy (MPhil) in Machine Learning and Machine Intelligence within the Department of Engineering at the University of Cambridge, United Kingdom. His academic journey was a bumpy road start during the timeline of his undergrad studies. Initially selected and admitted into The University of Dhaka (DU) at the Institute of Information Technology (IIT) within the Software Engineering (SE) Department for the BSSE program. Unfortunately, at the time due to his dad's heart attack and stroke which ultimately was forwarded into hospitalization and operation during that time, he had moved into The American International University-Bangladesh (AIUB) into the Faculty of Engineering (FE) within the Computer Engineering (CoE) Department for the (B.Sc.) program. He had finished his 4-year undergrad studies from AIUB where he had completed a total number of 198 credits which after mapping was finalized into 159 credits with the awarded degree for Bachelor of Science (B.Sc.) in Computer Engineering (CoE) with a Major in Biomedical Engineering and a Minor in 3D Modeling and Animation. As he was graduating, he had also received Honors which came in the form of the Prestigious Vice-Chancellors (VC) Award and Deans Award for the Best Project & Thesis along with, The Leadership Award at the 20th convocation of AIUB that had taken place on 10 October 2021.