software

MDPI

Article

Empirical Analysis of Data Sampling-Based Decision Forest Classifiers for Software Defect Prediction

Fatima Enehezei Usman-Hamza [1], Abdullateef Oluwagbemiga Balogun [2,*], Hussaini Mamman [2], Luiz Fernando Capretz [3], Shuib Basri [2], Rafiat Ajibade Oyekunle [4], Hammed Adeleye Mojeed [1,5] and Abimbola Ganiyat Akintola [1]

[1] Department of Computer Science, University of Ilorin, Ilorin 1515, Nigeria; usman-hamza.fe@unilorin.edu.ng (F.E.U.-H.); mojeed.ha@unilorin.edu.ng (H.A.M.); akintola.ag@unilorin.edu.ng (A.G.A.)
[2] Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Bandar Seri Iskandar 32610, Perak, Malaysia; hussaini_21000736@utp.edu.my (H.M.); shuib_basri@utp.edu.my (S.B.)
[3] Department of Electrical and Computer Engineering, Western University, London, ON N6A 5B9, Canada; lcapretz@uwo.ca
[4] Department of Information Technology, University of Ilorin, Ilorin 1515, Nigeria
[5] Department of Technical Informatics and Telecommunications, Gdańsk University of Technology, Gabriela Narutowicza 11/12, 80-233 Gdańsk, Poland
* Correspondence: abdullateef.ob@utp.edu.my

Abstract: The strategic significance of software testing in ensuring the success of software development projects is paramount. Comprehensive testing, conducted early and consistently across the development lifecycle, is vital for mitigating defects, especially given the constraints on time, budget, and other resources often faced by development teams. Software defect prediction (SDP) serves as a proactive approach to identifying software components that are most likely to be defective. By predicting these high-risk modules, teams can prioritize thorough testing and inspection, thereby preventing defects from escalating to later stages where resolution becomes more resource intensive. SDP models must be continuously refined to improve predictive accuracy and performance. This involves integrating clean and preprocessed datasets, leveraging advanced machine learning (ML) methods, and optimizing key metrics. Statistical-based and traditional ML approaches have been widely explored for SDP. However, statistical-based models often struggle with scalability and robustness, while conventional ML models face challenges with imbalanced datasets, limiting their prediction efficacy. In this study, innovative decision forest (DF) models were developed to address these limitations. Specifically, this study evaluates the cost-sensitive forest (CS-Forest), forest penalizing attributes (FPA), and functional trees (FT) as DF models. These models were further enhanced using homogeneous ensemble techniques, such as bagging and boosting techniques. The experimental analysis on benchmark SDP datasets demonstrates that the proposed DF models effectively handle class imbalance, accurately distinguishing between defective and non-defective modules. Compared to baseline and state-of-the-art ML and deep learning (DL) methods, the suggested DF models exhibit superior prediction performance and offer scalable solutions for SDP. Consequently, the application of DF-based models is recommended for advancing defect prediction in software engineering and similar ML domains.

Keywords: software defect prediction; data sampling; tree classifiers; decision forest

check for updates

Received: 15 December 2024
Revised: 13 March 2025
Accepted: 20 March 2025
Published: 21 March 2025

Citation: Usman-Hamza, F.E.; Balogun, A.O.; Mamman, H.; Capretz, L.F.; Basri, S.; Oyekunle, R.A.; Mojeed, H.A.; Akintola, A.G. Empirical Analysis of Data Sampling-Based Decision Forest Classifiers for Software Defect Prediction. *Software* **2025**, *4*, 7. https://doi.org/10.3390/software4020007

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The frequent and sustained solicitation of system specifications from end-users often contributes to budget and schedule overruns in software development projects. Improving the timely completion of software projects has remained a critical challenge within the software development industry, prompting extensive efforts aimed at addressing this issue [1,2]. Studies have shown a significant correlation between project budget and software complexity, revealing that as the allocated budget for software development increases, there is an initial decline in the success rate of the project [3,4]. Humphrey's investigation into project failures provides a detailed examination of the root causes and offers a comprehensive analysis of the factors that influence the effectiveness of large-scale software projects [5,6]. The study highlights a notable trend: for projects with budgets below $750,000, the success rate is approximately 55%. However, as the project scale increases, the probability of success declines precipitously. For projects exceeding $10,000,000, the likelihood of success approaches nearly zero [6,7]. This phenomenon represents a significant challenge for enterprises engaged in software development, underscoring the complexities associated with large-scale project management. That is, large codebases and increased complexity in software systems are strongly associated with a higher incidence of defects. The multifaceted nature of complexity, including architectural, cognitive, and technical aspects, creates numerous opportunities for defects to arise [8,9].

The introduction of defects in software modules can be traced to invalid programming practices or erroneous code, both of which result in inaccurate outputs and suboptimal software quality [10]. Defective software modules not only elevate development and maintenance costs but also lead to customer dissatisfaction, which can ultimately result in contract terminations or revisions [11]. Consequently, the presence of software defects is a primary contributor to the failure of software projects [12]. In other words, as systems scale, maintaining high levels of code quality and ensuring thorough testing becomes increasingly challenging, leading to a greater likelihood of undetected errors and defects [13].

Primarily, to mitigate these risks, software metrics are widely used as a means of evaluating the effectiveness and quality of software products. These metrics allow software engineers to conduct risk assessments and predict defects with a high degree of accuracy, thereby enabling the enhancement of software project quality [14]. However, due to the scale of modern systems and proliferation of software metrics, tracking and detecting defects is quite troubling. Invariably, addressing these issues requires a combination of improved software engineering (SE) practices, more sophisticated testing strategies and the implementation of software defect prediction (SDP) models that can help software teams focus their efforts on the most fault-prone areas of the system [10,15,16].

The use of SDP processes has emerged as a highly effective approach during the testing phases of the software development lifecycle [17,18]. SDP identifies software modules that are likely to contain defects, thus prioritizing them for more extensive testing. That is, SDP is essential for improving software quality by identifying potential defects early in the development cycle. It enhances software reliability, reduces costs, and ensures efficient resource allocation by directing testing efforts toward high-risk components. SDP minimizes the expense of fixing defects in later stages, accelerates time-to-market, and mitigates security vulnerabilities, making it a crucial practice in modern software engineering [15,19]. However, while SDP provides substantial benefits in software testing, accurately predicting which modules will be defective remains a complex task. Various challenges can impede the smooth implementation and application of SDP models, including data quality issues, model selection, and inherent limitations in the available metrics [20–22].

Typically, to integrate SDP optimally, modern software development leverages machine learning (ML) models trained on software metrics data gathered from prior systems,

previous software releases, or analogous software projects. Combining SDP with static and dynamic analysis tools enhances detection accuracy, while its integration into DevOps and CI/CD pipelines ensures continuous monitoring. Historical data, code complexity metrics, and automated feedback loops help developers address risks proactively, improving overall software quality [23]. Additionally, SDP supports Agile methodologies by allowing teams to prioritize defect-prone areas during sprint planning, leading to early defect resolution [14,24,25].

Once validated, these SDP models can predict which program modules are most susceptible to defects during the development process. The goal of SDP is to achieve high levels of software reliability and quality through the efficient deployment of resources [13]. Despite the widespread use of best practices in software engineering, achieving a defect-free system remains a formidable challenge. It is not uncommon for systems to contain undiscovered bugs or unforeseen defects, even when rigorous adherence to software development best practices is maintained [15]. To address these persistent issues, numerous ML models based on various computational characteristics have been proposed by researchers and software engineers for SDP tasks, often with varying degrees of success [8,17–21]. These studies emphasize the potential of artificial intelligence (AI), specifically, ML models for SDP; however, the predictive performance of these models has frequently been suboptimal, highlighting the need for further advancements in this area.

Notably, the predictive efficacy of SDP models is contingent upon the quality of the software metric datasets utilized in their development. The software features utilized for constructing SDP models affect the predictive effectiveness of these models [26–28]. The software features are complex and skewed, attributable to the issue of data quality problems. A notable example of the data quality problem is the inherent class imbalance problem that is present in datasets. Class imbalance in SDP arises when there is an unbalanced distribution of class labels, with non-defective instances constituting the majority and defective instances the minority. It is a latent issue that innately arises within software features and hinders the prediction effectiveness of SDP models [29–31]. Hence, it is imperative to develop ML models or methods that can accommodate or address this class imbalance issue to develop efficient and effective SDP models [32].

In response to these challenges, this study proposes the development of data sampling-based decision forest classifiers for SDP. These models aim to improve predictive accuracy and scalability, addressing the limitations observed in traditional SDP approaches. By leveraging advanced ML techniques, the data sampling-based decision forest models have the potential to enhance the effectiveness of SDP, ultimately contributing to more successful software project outcomes. Specifically, the decision forest models, including cost-sensitive forest (CS-Forest), forest penalizing attributes (FPA), functional trees and their enhanced variations based on homogeneous (bagging, boosting, cascade generalization, dagging, and rotation forest) ensembles, are employed for SDP. Decision forest models produce highly efficient decision trees (DTs) by leveraging the capabilities of all attributes within a dataset, informed by the diversity of tree models and their predictive performance [33]. This feature of the decision forest contrasts with conventional DTs that employ only a subset of the attributes [34,35].

CS-Forest, as a decision forest model, operates on the principle of minimizing the total misclassification cost rather than simply maximizing overall accuracy. By incorporating cost considerations throughout training and prediction, CS-Forest is particularly effective for applications where certain misclassification errors are significantly more detrimental than others [36,37]. FT similarly arises from the functional induction of multivariate decision trees and discriminant functions. FT employs positive induction to hybridize a DT with a linear function, resulting in a DT characterized by multivariate decision nodes and

leaf nodes that utilize discriminant functions for predictions [38,39]. FPA, as a decision forest model, modifies the traditional Random Forest (RF) framework to prioritize and penalize attributes based on their predictive utility or impact on classification performance. Its working principle revolves around dynamically penalizing or weighting attributes during the construction of DTs to improve classification performance, especially for class imbalanced problems [34,35].

Additionally, enhanced DF models utilizing diverse homogeneous ensemble techniques are proposed. The main advantage of homogeneous ensembles is their ability to reduce variance, improve generalization, and maintain simplicity, making them a powerful and practical approach for boosting model performance without introducing the complexity of heterogeneous ensembles [40,41]. They are particularly well suited for problems requiring scalability, interpretability, and robustness to overfitting. Hence, homogeneous ensemble methods are proposed to enhance the predictive performance of DF models, thereby producing robust and generalizable SDP models. The synthetic minority oversampling technique (SMOTE) is utilized as an effective method to address the latent class imbalance issue in software defect datasets [20,31].

Summarily, the ongoing refinement of SDP models is therefore critical, as it enables the early detection of defects, the efficient allocation of resources, and the improvement of software quality across a wide range of development projects. This study aims to examine the effectiveness of DF models (CS-Forest, FPA, and FT), and their enhanced ensemble variants) in addressing the class imbalance problem in SDP.

This study's primary accomplishment is summarized as follows:

1. This study aims to empirically assess the effectiveness of decision forest (DF) models (CS-Forest, FPA, and FT) on balanced and imbalanced SDP datasets;
2. The objective is to create improved ensemble variants of decision forest models (CS-Forest, FPA, and FT) utilizing diverse homogeneous ensemble techniques;
3. To empirically assess and compare DF models (CS-Forest, FPA, and FT) alongside their enhanced ensemble variants with current SDP models.

The subsequent sections of this paper are organized as follows. Section 2 presents a detailed analysis of contemporary SDP solutions. Section 3 outlines the experimental framework and emphasizes the proposed solutions. Section 4 provides a detailed analysis of the research results and observations, while Section 5 presents the threat to validity and Section 6 concludes the study.

2. Related Works

The review and evaluation of current SDP models and related solutions from existing studies reveal a variety of approaches built on different computational paradigms. Specifically, SDP techniques utilizing statistical, ML, and deep learning (DL) methods are examined.

It is worth noting that SDP is not the only method available for detecting flaws or bugs in software systems. For instance, techniques like model checking [42] and static code analysis tools, such as Coverity [43], are widely employed for defect detection. These methodologies primarily rely on fault localization to identify and isolate issues within software systems. Fault localization typically involves analyzing the discrepancies between the outputs of successful and failed software tests to locate problems in the source code. However, traditional techniques like model checking and static analysis are limited to identifying defects in the existing codebase, whereas SDP provides a proactive means to detect potentially defect-prone modules in a software system.

Initial SDP methods were heavily reliant on Software Requirements Specification (SRS) documents to predict potential deficiencies. For example, Smidts et al. proposed a software

reliability model based on SRS and failure data [44]. Similarly, Cortellessa et al. integrated Unified Modeling Language (UML) representations of software architecture with a Bayesian framework for SDP [45]. However, these approaches did not support code reuse, implying that failures in components were treated independently. Gaffney and Davis proposed a phase-based framework for software reliability that relied on defect data collected at various stages of development [46,47]. Despite their utility, such models are often tailored to specific organizations, limiting their general applicability. Fuzzy logic has also been explored in SDP. For instance, Al-Jamimi employed a Takagi-Sugeno fuzzy inference engine for defect prediction [48]. Similarly, Yadav utilized fuzzy logic to improve SDP across different phases of development [49]. Also, Adak combined multivariate analysis of variance (MANOVA) and fuzzy logic for SDP. Their experiment results indicated the effective of the hybrid approach [50]. While these methods introduced a range of features for defect prediction, they are hindered by decidability issues, which limit their practical application. Fuzzy logic usually struggles with noisy or sparse data and lacks self-learning capabilities, reducing its effectiveness compared to statistical or modern ML methods [51].

Statistical methods have been employed to improve SDP models. For example, researchers proposed a kernel discrimination classifier (KDC) to handle the non-linear separability and imbalance common in SDP datasets. This approach demonstrated efficiency comparable to existing methods [52]. Another study introduced a subclass discriminant analysis (ISDA) method for within-project SDP, which was later extended with a semi-supervised transfer component analysis (SSTCA) for cross-project SDP [53]. Despite their simplicity and ease of implementation, statistical methods often struggle with non-separable datasets and underperform compared to ML and DL-based approaches.

ML-based methods have also been extensively studied. For instance, tree-based classifiers, known for their simplicity and effectiveness, have been applied to SDP. A study evaluating ten tree-based classifiers on NASA datasets found Random Forest (RF) to outperform other classifiers [54]. Another study compared k-nearest neighbor (kNN), RF, and multilayer perceptron (MLP) for SDP, concluding that kNN performed best [55]. However, this study was limited in scope due to the small number of classifiers and datasets used, and kNN's performance heavily depended on parameter tuning. To address parameter optimization issues, a study explored 26 ML models across 18 datasets, revealing that tuning parameter significantly improved model stability and performance [56]. Despite these advancements, ML models often face challenges such as misclassification and increased preprocessing overhead.

In recent years, DL methods have gained traction for SDP. These approaches include multilayer perceptron (MLP), Convolutional Neural Networks (CNNs), and hybrid models [57,58]. One study proposed combining word embedding with a semantic Long Short-Term Memory (LSTM) network, which extracted features from source code and predicted defects effectively [59]. Another study developed gated hierarchical LSTM networks (GH-LSTMs) to enhance prediction accuracy [60]. Similarly, a hybrid model combining CNN and Bidirectional LSTM (Bi-LSTM) demonstrated strong performance by extracting Abstract Syntax Tree (AST) semantics [60]. Another study integrated Bi-LSTM with BERT-based semantic features, yielding results competitive with state-of-the-art methods [61]. While DL models show significant potential for SDP due to their ability to process complex data with minimal preprocessing, they come with notable drawbacks. These include computational expense, dependence on parameter tuning, and the "black box" nature of DL, which complicates interpretability.

In summary, as presented in Table 1, numerous methods and models, ranging from statistical approaches to ML and DL, have been proposed for SDP. However, the persistent challenges of improving accuracy, scalability, and interpretability underscore the need for

further research. Furthermore, recent SDP studies are advocating for the use of data sampling methods as a solution to class imbalance problem in SDP. For instance, Li, et al. [62] leveraged on a novel data sampling method to design a non-parametric data selection and sampling based domain programming predictor (DSSDPP) for cross-project defect prediction (CPDP). Similarly, Bennin, et al. [63] presented a comprehensive analysis on the statistical and practical impact of data sampling methods in SDP. They investigated the stability of diverse balancing ratio of data instances across prominent resampling approaches on prediction performance. Hence, this study contributes to this ongoing effort by proposing data sampling-based enhanced DF models as novel approaches to advancing SDP and paving the way for a more efficient and accurate SDP landscape.

Table 1. Analyses of key related literature on SDP.

References	SDP Models	Class Imbalance	Findings	Limitations
Al-Jamimi [48]	Fuzzy logic (Takagi-Sugeno fuzzy inference engine)	No	The reported findings demonstrated the ability of fuzzy logic to produce transparent defect prediction models.	Scalability challenges, static rule bases, and dependence on expert-defined membership functions, making it less adaptable to complex and dynamic systems. While hybrid approaches combining fuzzy logic with others exist, they often face integration and computational complexity issues.
Yadav and Yadav [49]	Fuzzy inference system	No	The projected defect density indicators assist analyze fault severity in software project SDLC artefacts.	
Adak [50]	MANOVA, Fuzzy logic, and Gini decision tree	No	Hybrid fuzzy logic models with statistical method provide better outcomes than pure fuzzy or data mining models.	
Ma, Qin and Zhu [52]	kernel discrimination classifier (KDC)	No	KDC can tackle nonlinearly separable and class-imbalanced problems. Experiments show that KDC can give good performances among the comparative methods on the test sets.	They often require extensive computational resources to process high-dimensional data and tune kernel parameters effectively, limiting their scalability. Additionally, the interpretability of KDC or ISDA models is reduced, making it challenging to understand the impact of individual features or justify predictions, especially in domains where transparency is critical.
Jing, Wu, Dong and Xu [53]	Improved subclass discriminant analysis (ISDA)	No	ISDA performed better than other state-of-the-art within-project class-imbalance learning methods	
Naseem, Khan, Ahmad, Almogren, Jabeen, Hayat and Shah [54]	Credal Decision Tree (CDT), CS-Forest), Decision Stump (DS), FPA, Hoeffding Tree (HT), DT, Logistic Model Tree (LMT), RF, Random Tree (RT), and REP-Tree (REP-T).	No	RF outperformed other classifiers.	Tree-based classifiers, such as DT and RF, often struggle with imbalanced datasets and the latent class imbalance problem was not considered in the performance of the experimented tree-based classifiers.

Table 1. *Cont.*

References	SDP Models	Class Imbalance	Findings	Limitations
Abdulshaheed, Hammad, Alqaddoumi and Obeidat [55]	kNN, MLP, and RF	No	kNN outperformed other methods such as RF and MLP	The study was limited in scope due to the small number of classifiers and datasets used, and kNN's performance heavily depended on parameter tuning.
Al Qasem, Akour and Alenezi [57]	MLP and CNN	No	The study found that adding layers positively impacts the ideal number of layers for each dataset. The best performance was achieved using the ReLU activation function.	DL models are sensitive to hyperparameter settings and could overfit to noise or specific patterns in training data, necessitating careful tuning and regularization techniques. DL models face significant challenges with imbalanced datasets, as they tend to prioritize the majority class, leading to poor performance on minority class predictions.
Liang, Yu, Jiang and Xie [59]	Semantic LSTM	No	This method outperformed recent defect prediction methods in most projects.	
Farid, Fathy, Eldin and Abd-Elmegid [60]	CNN and Bi-LSTM (CBIL)	No	The proposed method significantly improved base models.	
Uddin, Li, Ali, Kefalas, Khan and Zada [61]	Bi-LSTM and BERT	No	It employs BiLSTM to leverage contextual information derived from the embedded token vectors obtained via the BERT model. Additionally, it employs an attention mechanism to identify significant features of the nodes.	
Qiao, et al. [64]	Deep Neural Network (DPNN)	No	The research results demonstrate that the proposed approach is precise and enhances existing state-of-the-art methods.	

3. Methodology

This section presents the research framework and experimental methodology employed in this study. It specifically examines the baseline ML classifiers, the software defect datasets utilized, the homogeneous ensemble methods, the performance evaluation metrics, and the detailed experimentation procedures. Through this approach, this study outlines how these elements are integrated to assess and validate the effectiveness of the proposed models.

3.1. Decision Forest Classifiers

Decision forest (DF) classifiers are a type of tree-based classification algorithm that uses a collection of DTs to perform classification, regression, and other predictive tasks. DFs are highly regarded for their robustness, scalability, and ability to handle complex data

patterns [35]. DFs operate by combining the outputs of multiple decision trees. Each tree is trained on a subset of the data, and their predictions are aggregated based on diverse computational structures and operations. At each split in a tree, a random subset of features is considered [65]. This further decorrelates the trees in the forest, enhancing generalization performance. DFs can handle high-dimensional data, missing values, and both numerical and categorical variables without requiring extensive preprocessing [54]. Their continuous evolution and adaptation to emerging challenges ensure their relevance in ML tasks. In this study, the trio of CS-Forest, FPA, and FT classifiers are selected as DFs for investigation in SDP. A detailed explanation of the selected DFs is presented in the preceding subsections.

3.1.1. Cost-Sensitive Forest (CS-Forest) Classifier

In CS-Forest, each DT is trained to optimize a loss function that integrates the costs associated with different types of misclassification errors. Unlike conventional DTs that solely focus on accuracy, the CS-Forest algorithm modifies the training process to account for class imbalance and cost-sensitive considerations [35]. As presented in Algorithm 1, the construction of individual trees incorporates a modified decision tree algorithm that evaluates splitting criteria using an adjusted information gain ratio, emphasizing the cost implications of various splits. During prediction, each tree independently evaluates the input data and assigns a predicted class label, leveraging the cost-sensitive criteria embedded during its training. These individual tree predictions are then combined to form the forest's overall output [36]. However, rather than treating each tree equally, CS-Forest employs a weighted averaging mechanism for aggregation. The weights are determined by assessing the trees' performance on a validation set and accounting for their respective misclassification costs. This approach ensures that trees with better cost-sensitive accuracy have greater influence on the final prediction, effectively reducing biases toward the majority class and enhancing the model's reliability in identifying minority class instances [66].

Extending the basic principle, CS-Forest can be enhanced with advanced ensemble techniques such as boosting or bagging to improve its robustness and accuracy further. These enhancements can dynamically adjust tree weights or create diverse forest compositions, improving sensitivity to rare events or underrepresented classes. The method has broad applicability in domains where misclassification costs are non-uniform, where it helps to balance sensitivity and specificity effectively.

3.1.2. Forest Penalizing Attribute (FPA) Classifier

The FPA classifier, as outlined by [34] and presented in Algorithm 2, fosters diversity in DFs by addressing weight-related concerns through strategies like weight assignment and weight increment. FPA builds a collection of highly accurate DTs by leveraging the full range of non-class attributes in a dataset. A key characteristic of FPA is its dynamic weight adjustment mechanism, where attribute weights are updated within a predefined Weight Range (WR), controlled by factors such as attribute level (denoted as λ) and overlap prevention (ρ). This ensures that the WRs for different levels remain distinct and effective in optimizing attribute significance during tree construction. FPA's design is particularly effective in mitigating the adverse effects of neglected attributes. It incrementally adjusts the weights of attributes not utilized in the latest tree, ensuring these attributes have the potential to contribute to subsequent trees. This systemic weight update mechanism not only enhances model reliability but also reinforces its adaptability across various datasets and applications [34]. FPA has demonstrated its effectiveness in complex ML tasks, such as Intrusion Detection Systems (IDS), a critical component in network security, as explored in prior research. Furthermore, FPA has been successfully integrated with heuristic techniques

to optimize its performance, making it a versatile choice for scenarios requiring robust and adaptive classification models. FPA's ability to balance attribute importance dynamically enhances its capability to generalize effectively across datasets with varying characteristics [67]. This positions FPA as a reliable method for developing robust ML solutions in domains with complex data distributions and class imbalance challenges.

Algorithm 1 Cost-Sensitive Forest Classifier (CS-Forest)

Input:
Training dataset with features and corresponding class labels: $\boldsymbol{D}$
Misclassification cost matrix indicating the penalty for misclassifying one class as another: $\boldsymbol{C}$
Number of trees in the forest: $\boldsymbol{n}$
Test dataset for evaluation: $\boldsymbol{T}$
Maximum depth or criteria for splitting a decision tree: $\boldsymbol{k}$
Output:
Predicted class labels for instances in $\boldsymbol{T}$
Begin
1. Initialization:

- Define the number of trees $\boldsymbol{n}$ to construct in the forest.
- Prepare a cost matrix $\boldsymbol{C}$
- Initialize an empty set $\boldsymbol{F}$ to hold all the trained trees

2. Training trees:
For each tree $\boldsymbol{i}$ (where $\boldsymbol{i} \in \{1, 2, 3, \ldots, n\}$):

- Bootstrap Sampling:
  Generate a bootstrapped sample $\boldsymbol{D_i}$ from $\boldsymbol{D}$ by random sampling with replacement.
- Cost-Sensitive Decision Tree Construction:
  Begin at the root node with the entire dataset $\boldsymbol{D_i}$
  At each node, evalaute possible splits using an adjusted information gain ratio

$$\textbf{\textit{Adjusted Gain Ratio}} = \textbf{\textit{Information Gain}} - \textbf{\textit{Cost Factor}}$$

Select the split that maximizes the adjusted gain ratio, considering the misclassification cost $\boldsymbol{C}$

- Stopping Criteria:
  Stop growing the tree when:

The maximum depth $\boldsymbol{k}$ is reached

A node becomes pure (contains instances of only one class).
Further splitting does not improve the adjusted gain ratio significantly

- Add the trained decision tree $\boldsymbol{T_i}$ to $\boldsymbol{F}$

3. Prediction:

- For each instance $\boldsymbol{x}$ in the test dataset $\boldsymbol{T}$
  Pass x through each tree $\boldsymbol{T_i}$ ***in*** $\boldsymbol{F}$ to get individual predictions
  Assign weights to each tree's prediction based on tree's performance and cost $\boldsymbol{C}$
- Combine predictions using a weighted voting scheme: $\hat{y} = \textbf{\textit{arg max}}_y \sum_{i=1}^{n} w_i * \boldsymbol{P_i}(\boldsymbol{y}|\boldsymbol{x})$,

where w_i is the weight of tree T_i and $P_i\ (y|x)$ is its probability estimate for class y
4. Output:
Output the predicted class labels for all instance in $\boldsymbol{T}$
End

Algorithm 2 Forest Penalizing Attribute (FPA) Classifier

Input:
Dataset $\boldsymbol{D}$ with attributes $\boldsymbol{A}$ and class labels $\boldsymbol{C}$
Number of decision trees $\boldsymbol{T}$
*Weights Range (**WR**) configuration*
Parameters: $\boldsymbol{\lambda}$ (attribute level), $\boldsymbol{\rho}$ *(overlap prevention factor)*
Maximum depth or criteria for splitting a decision tree: $\boldsymbol{k}$
Output:
*Predicted class labels for instances by **FPA***
Begin
1. Initialize Parameters:

- Assign initial weights to all attributes $\boldsymbol{\omega(a) = 1}$ ***for*** $\boldsymbol{a \in A}$.
- Set ***WR*** values for different levels using $\boldsymbol{\lambda}$ and $\boldsymbol{\rho}$ *to define non-overlapping ranges*
- Initialize an empty set $\boldsymbol{F}$ to hold all the trained trees

2. Build Ensembles:
For each tree $\boldsymbol{t}$ in $\boldsymbol{T}$:

- Select Attributes:
  Randomly sample as a subset of attributes $\boldsymbol{A_t \subseteq A}$ based on their current weights $\boldsymbol{\omega(a)}$
- Train Decision Tree:
  Construct a decision tree t using $\boldsymbol{A_t}$ as the feature space and $\boldsymbol{D}$ as the training data
  Use a splitting criterion to determine splits
- Evaluate Attribute Usage:
  Identify attributes used in t
- Update Weights:
  For attributes used in $\boldsymbol{t}$: Update weights based on ***WR*** configuration

For attributes not used in t: incrementally increase weights for testing in subsequent trees.
3. Aggregate Prediction:

- For a given input $\boldsymbol{x}$, collect predictions from all trees in the ensemble.
- Final prediction using weighted or majority voting

4. Output:
Output the predicted class labels for all instance in $\boldsymbol{D}$
End

3.1.3. Functional Tree (FT) Classifier

Algorithm 3 presents FT, which is an advanced tree-based classifier that combines multivariate DTs with discriminant functions using constructive induction. This integration allows FT to generalize multivariate trees by incorporating features into both decision nodes and leaf nodes [68]. Unlike standard DTs that split input data by comparing attributes to constants, FT leverages linear regression (LR) functions for internal node splits (termed oblique splits) and applies functional models in leaf nodes for classification or regression tasks. FT constructs its trees dynamically based on the data, forming decision nodes as the classification tree evolves. At the pruning stage, functional models replace traditional leaf nodes, resulting in functional leaves. This process helps FT accommodate complex relationships within the data, often leading to improved performance. For prediction tasks, a dataset traverses the tree from root to leaf, with features expanded at each decision node based on node-built functions [39]. The decision test at each node determines the traversal path, and the final classification or prediction is made using either a function associated with the leaf or a related constant. A notable advantage of FT is its ability to partition the input space into hyper-rectangles, fitting data within each partition using constructor functions. This approach is particularly beneficial for handling datasets with complex, non-linear relationships [69].

Algorithm 3 Functional Tree (FT) Classifier

Input:
Training Dataset $\boldsymbol{D} = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), (\boldsymbol{x_2}, \boldsymbol{y_2}), \ldots, (\boldsymbol{x_n}, \boldsymbol{y_n})\}$ where $\boldsymbol{x_i}$ = inputs features, $\boldsymbol{y_i}$ = class labels
Parameters for pruning and constructor functions configurations.
Output:
Predicted class labels for instances by ***FT***
Begin
1. Initialize Parameters:
   - Start with the entire dataset at the root node.
   - Define a splitting criterion (e.g., information gain, Gini index).
   - Choose a functional model (e.g., linear regression) for leaf prediction.
2. Tree Construction:
   - Check stopping criteria:
     If all samples belong to the same class, assign the class label to the node and terminate splitting.
     If the dataset is too small, fit a functional model and assign the node as a leaf.
   - Evaluate potential splits
     For each attribute, calculate the best split using the selected criterion.
     Allow splits to be oblique by using a linear combination of features if applicable.
   - Select the best split:
     Choose the attribute or linear function that maximizes the splitting criterion.
   - Partition the dataset:
     Split the dataset into subsets based on the selected attribute or linear function.
   - Recursively build child nodes
     Apply the algorithm on each subset
3. Functional Leaf Construction:
   - For each leaf node
     Fit a functional model (e.g., linear regression or another specified constructor function) using the data in that partition.
     Store the functional model as part of the leaf.
4. Tree Pruning:
   - Evaluate the performance of subtrees using a validation dataset.
   - Replace subtrees with functional leaves if pruning improves validation performance.
   - Ensure that functional models are optimized for predictive accuracy during pruning.
5. Prediction:
   - To classify a new instance $\boldsymbol{x}$.
     Traverse the tree from the root, applying decision tests at each node to determine the path.
     Once a leaf is reached, use the functional model in the leaf to predict the class label.
6. Output:
An FT classifier capable of classifying new data instances.
End

In summary, DFs provide a flexible and robust framework for ML tasks, excelling in situations requiring advance computation and modeling. Its ability to adaptively partition data and apply advanced functional modeling makes it a powerful alternative to traditional DTs. Table 2 presents the parameter setting of the selected DFs as used in this study.

Table 2. Parameter Configuration of Implemented Decision Forest Classifiers.

DF Models	Parameter Configuration
CS-Forest	BatchSize = 100; confidenceLevel = 0.25; costGoodness = 0.2; costMatric = (2 × 2) with default value "1"; MinRecLeaf = 10; numberTrees = 60; separation = 0.3
FPA	BatchSize = 100; numberTrees = 10; seed = 1; simpleCartMinimumRecords = 2; simpleCartPruningFolds = 2
FT	BatchSize = 100; binSplit = False; errorOnProbabilities = False; minNumInstances = 15; modelType = InnerLeaves; numBoostingIterations = 15; useAIC = False; weighTrimBeta = 0.0;

3.2. Homogeneous Ensemble Methods

Homogeneous ensemble methods leverage multiple models of the same type (e.g., DTs, neural networks, etc.) to improve the accuracy, robustness, and generalization of predictions [40]. By combining outputs from multiple instances of the same algorithm, these methods reduce bias, variance, and susceptibility to overfitting, enhancing the overall performance of the machine learning system [41]. This study included bootstrap aggregating (bagging), boosting, disjoint aggregating (dagging), rotation forest and cascade generalization.

3.2.1. Bootstrap Aggregating (Bagging) Technique

Bagging, or bootstrap aggregating, is a robust homogeneous ensemble method widely used to enhance the predictive accuracy and stability of classification algorithms. It operates by training multiple base classifiers, each on a unique bootstrap sample derived from the original dataset. These bootstrap samples are created by randomly sampling with replacement, ensuring each subset may contain duplicate entries while omitting others. Once the classifiers are trained on their respective subsets, their predictions are aggregated to form a final ensemble output. The aggregation of outputs ensures that the ensemble leverages the strengths of individual classifiers while mitigating their weaknesses [70]. Specifically, bagging effectively reduces variance without increasing bias, as each model learns slightly different aspects of the data. This makes the method particularly useful for high-variance algorithms, such as DTs, where individual models may overfit the training data. By averaging predictions, bagging smooths out anomalies and yields more generalizable results. Moreover, the independence of models in a bagging ensemble means that they can be trained in parallel, making the method computationally efficient for large datasets. However, while bagging is highly effective at variance reduction, it may not significantly improve performance when the primary issue is high bias in the model [71]. Additionally, computational demands increase with the number of base classifiers, and interpretability may diminish as the ensemble size grows. Despite these challenges, bagging remains a cornerstone method in ensemble learning, forming the basis for advanced techniques like RFs, which incorporate feature randomness to further enhance model diversity and accuracy. Algorithm 4 illustrates the pseudocode for Bagging technique.

3.2.2. Boosting Technique

Boosting is a homogeneous ensemble learning technique designed to convert weak learners into strong predictors by iteratively training models in sequence. Unlike parallel methods such as bagging, boosting focuses on adjusting the training process based on the performance of previous classifiers. Specifically, it assigns higher weights to misclassified data points in each iteration, encouraging the next classifier to focus more on these difficult instances. At the end of the iterative process, the individual predictions from all weak classifiers are aggregated, typically through a weighted voting mechanism for classification tasks or weighted averaging for regression tasks. This process ensures that the final ensemble model leverages the strengths of each weak learner while mitigating

their individual limitations [70]. The sequential nature of boosting ensures that each weak learner is fine-tuned to address the shortcomings of its predecessor, resulting in a robust model capable of handling complex datasets. However, boosting can be computationally intensive and prone to overfitting, particularly if the number of iterations is too high or the base learners are too complex [72]. Despite these challenges, boosting is widely used in predictive analytics, where high accuracy is critical. In this study, adaptive boosting (Adaboost) is implemented due to its ability to adjust the weight of training instances after each iteration based on the errors made by the current model [41]. Algorithm 5 presents the pseudocode for the Boosting approach.

Algorithm 4 Bagging Technique

Input:
Training Dataset ***D*** *with instances* ***N***
Number of base classifiers ***T***
Base Classifiers {CS-Forest, FPA, FT}
Output:
The ensemble model ***E*** *for predictions.*
Begin
1. Initialize Parameters:
   - An empty ensemble E = $\{C_1, C_2, \ldots, C_T\}$ ***where*** C_i ***represents a base model***
2. Training Phase:
   - For each base classifier $t = 1, 2, \ldots,$ ***T***:
     Bootstrap Sampling:
     Randomly sample ***N*** instances with replacement from the original dataset ***D*** to create a bootstrap dataset D_t
     Train Classifier:
     Train the base classifier C_t on D_t
3. Aggregation Phase:
   - Prediction
     For a new input instance ***x***:
     Each base classifier C_t ***generates a prediction*** $\hat{y}_t$
     Combine prediction based on Majority voting to determine the final class label
4. Output:
The ensemble model *E* for predictions
End

In summary, homogeneous ensemble methods are powerful tools for achieving state-of-the-art results across a wide range of ML tasks. By leveraging diverse strategies, they address specific challenges such as overfitting, bias, and variance, making them indispensable in modern ML pipelines. Table 3 presents the parameter setting of the selected homogeneous ensemble methods as used in this study.

Table 3. Parameter Configuration of Implemented Homogeneous Ensemble Methods.

Homogeneous Ensemble	Parameter Configuration
Bagging	bagSizePercent = 100; calcOutOfBag = False; numIterations = 10; classifiers = {FPA, CS-Forest, FT}; outputOutOfBagComplexityStatistics = False
Boosting	batchSize = 100; resume = False; numIterations = 10; classifiers = {FPA, CS-Forest, FT}; useResampling = False; weightThreshold = 100

Algorithm 5 Boosting Technique (Adaboost)

Input:
Training Dataset $\boldsymbol{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ *where* x_i = *inputs features,* y_i = *class labels*
Number of Iterations $\boldsymbol{T}$
Base Classifiers {CS-Forest, FPA, FT}
Output:
The ensemble (strong) model $\boldsymbol{H(x)}$ *for predictions.*
Begin
1. Initialize Weights:
   - Assign an initial weight to each training sample:
     $w_i^{(1)} = \{1/n, \forall i = 1, 2, \ldots, n\}$ ***where*** $n =$ ***total number of training instances***
2. For $t = 1, 2, \ldots, T$:
   Train a Weak Classifier:
   Train a weak classifier $h_t(x)$ using the weighted training dataset.
   $\varepsilon_t = \frac{\sum_{i=1}^{n} w_i^{(t)} * \alpha(h_t(x_i) \neq y_i)}{\sum_{i=1}^{n} w_i^{(t)}}$ where α equals 1 if the condition is true and 0 otherwise.
   Compute Classifier Weight:
   Calculate the weight of the weak classifier $\beta_t : \beta_t = \frac{1}{2} In\left(\frac{1-\varepsilon_t}{e_t}\right)$
   Update Sample Weight:
   Update the weights of the training samples: $w_i^{(t+1)} = w_i^{(t)} * exp(-\beta_t * y_i * h_t(x_i))$
   Normalize the weights such that $\sum_{i=1}^{n} w_i^{(t+1)} = 1$
3. Aggregation Weak Classifier:
   - The final strong classifier is weighted sum of all weak classifiers $H(x) = sign\left(\sum_{t=1}^{T} \beta_t * h_t(x)\right)$
4. Output:
The ensemble (strong) model $\boldsymbol{H(x)}$ for predictions
End

3.3. Synthetic Minority Oversampling Technique (SMOTE)

SMOTE (synthetic minority over-sampling technique) is a widely used statistical method designed to address class imbalance in datasets by generating synthetic examples for the minority class. As presented in Algorithm 6, unlike the simple duplication of minority instances, SMOTE synthesizes new data points by interpolating between existing instances within the minority class [73]. This approach reduces the imbalance ratio between the minority and majority classes, enhancing the dataset's overall balance while ensuring that the majority class does not grow disproportionately [74]. In SDP, where the goal is to identify defective software modules, datasets often suffer from significant class imbalance, with non-defective instances overwhelmingly outnumbering defective ones [75]. This imbalance can skew model training, leading to biased predictions that favor the majority class. By applying SMOTE, the minority (defective) class is bolstered, enabling classification models to better discern between defective and non-defective instances without introducing bias. This leads to improved detection of defects and enhances the model's robustness.

3.4. Software Defect Datasets

For this research, software metrics derived from publicly available NASA repositories were employed to train and evaluate the proposed models. Specifically, the [76] release of the NASA corpus as presented in Table 4 was utilized, which includes comprehensive datasets tailored for SDP tasks. These datasets consist of metrics derived from static code analysis, a method that examines software without executing it, ensuring the detection of potential defects early in the development cycle. Static code metrics often include

measures like cyclomatic complexity, lines of code, coupling, cohesion, and others that are critical indicators of software quality and maintainability [32]. The NASA datasets are widely recognized for their use in machine learning experiments due to their diversity and robust representation of real-world defect scenarios. They serve as a benchmark for evaluating the performance of predictive models in the field of software engineering. The reliance on these datasets also enables reproducibility and comparability across studies. Previous research has extensively validated these metrics for their ability to predict defective and non-defective software modules, making them a reliable foundation for this study's experimentation and validation phases [19,21,26,53,75]. Furthermore, using such a well-established corpus ensures that the findings contribute to the broader body of knowledge in SDP, offering insights into the effectiveness of the investigated models under realistic conditions. The datasets' variety, which spans different software projects and defect patterns, also ensures that the proposed models are evaluated against diverse challenges, enhancing their generalizability and robustness.

Algorithm 6 Synthetic Minority Oversampling Technique (SMOTE)

Input:
Minority class dataset: $\boldsymbol{D_m}$
Number of synthetic samples to generate: $\boldsymbol{N}$
Number of nearest neighbors: $\boldsymbol{k}$
Output:
Augmented dataset with $\boldsymbol{N}$ *synthetic samples.*
Begin
1. Compute k-nearest neighbors:
For each instance $\boldsymbol{x_i}$ in the minority class $\boldsymbol{D_m}$ calculate its k-nearest neighbors using a distance metric (e.g., Euclidean distance)
2. Select neighbors for oversampling:
Randomly select one or more neighbors $\boldsymbol{x_j}$ from the k-nearest neighbors for each $\boldsymbol{x_i}$
3. Generate synthetic instances
- For each selected neighbors $\boldsymbol{x_j}$, generate a synthetic instance based on the following: $\boldsymbol{x_{new} = x_i + \rho \times (x_j - x_i),\ where\ 0 \leq \rho \leq 1}$
- This interpolated a new instance $\boldsymbol{x_{new}}$ between $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$

4. Repeat:
Continue the process until $\boldsymbol{N}$ synthetic instances have been generated.
5. Augment dataset:
Combine the synthetic instance with the original dataset to form the augmented dataset.
End

Table 4. Software Metric Datasets.

Datasets	Instances	Features	Defective Instances	Non-Defective Instances
CM1	327	38	42	285
KC1	1126	22	294	868
KC3	194	40	36	158
MW1	250	38	25	225
PC1	679	38	55	624
PC3	1053	38	130	923
PC4	1270	38	176	1094
PC5	1694	39	458	1236

3.5. Experimental Framework

This section details the experimental procedures undertaken in this study to evaluate the proposed SDP methods. Figure 1 provides a schematic overview of the experimental framework used to validate the effectiveness of the suggested approaches. The framework was designed to ensure an empirical evaluation of the models, leveraging software metric datasets sourced from the NASA repository. A cross-validation (CV) strategy was employed for training and testing the models, selected for its well-documented capability to develop predictive models with reduced bias and variance. CV ensures robust performance assessment by iteratively using each instance in the dataset for both training and testing, allowing the model to generalize effectively across diverse data partitions.

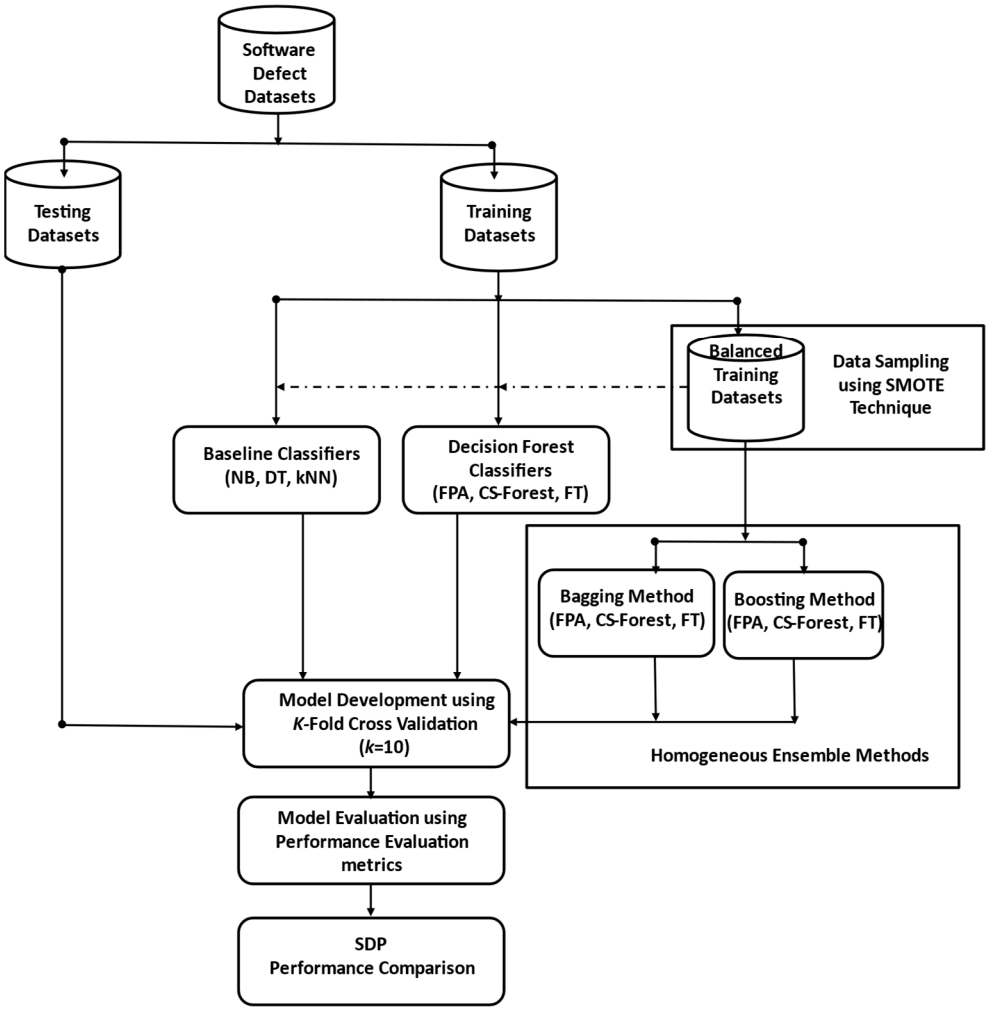

Figure 1. Experimental Framework.

The CV methodology has been extensively discussed in existing literature, confirming its reliability and utility for model evaluation in software defect prediction (SDP) tasks. References [21,22,41,56,64,65] highlight its ability to provide consistent performance metrics by minimizing overfitting and improving model stability.

For simplicity, the experimental procedure is broken down into multiple experimentation stages. The essence of the multiple experimentation stages is to effectively investigate and validate the prediction performances of the implemented DF methods in a stepwise approach with respect to baseline and state of the arts SDP models. Specifically, within this research, the SDP performances of the proposed DF (FPA, CS-Forest, and FT) models are systematically compared against prominent baseline classifiers, such as k-nearest neighbor

(kNN), naïve bayes (NB), and DT to validate their effectiveness. Additionally, this study extended the investigation by analyzing the performance of the homogeneous ensemble variants of the DF models. This is to enable a comprehensive analysis of performance improvements based on the deployment of the ensemble methods. Furthermore, the predictive performances of the DF models and their respective homogeneous ensemble variants in the presence of CIP is investigated. That is, the SDP performances of the DF models and their enhanced variants on original and balanced software defect datasets are analyzed. This is to understand how DF models and homogeneous ensemble-enhanced DF models will respond to the class imbalance problem.

The entire suite of models was implemented using the open source ML libraries available in WEKA 3.9.6 [77], a robust platform that facilitates the implementation, evaluation, and visualization of machine learning algorithms. This experimental structure ensures that the proposed methodologies are rigorously tested, offering credible insights into their advantages over traditional approaches. By integrating CV techniques and a robust experimental framework, this research contributes to the ongoing efforts to improve the reliability and efficiency of SDP models.

3.6. Experimental Performance Metrics

Accuracy and area under the curve (AUC), metrics were deployed to rigorously assess and compare the predictive performance of various SDP models. These metrics were chosen due to their wide acceptance and application in prior research on SDP methods, ensuring consistency with industry standards and prior academic work [17,18,22,48,60]. Each metric provides a unique perspective: accuracy evaluates the proportion of correctly predicted instances, AUC measures the ability to differentiate between classes making it particularly robust for imbalanced datasets. These performance indicators were selected not only for their frequent use but also for their ability to collectively capture a holistic view of model performance, addressing strengths and limitations in various predictive scenarios [31]. The inclusion of metrics like AUC is particularly significant as it provides insights into the correlation between actual and predicted classifications, even when dataset distributions are skewed [78]. By employing these metrics, this study ensures a reliable, multi-dimensional evaluation framework that highlights the strengths and weaknesses of the tested models. This approach enhances the interpretability of the results and facilitates a robust comparison with existing models, contributing valuable insights to the field of SDP.

4. Results and Discussion

This section evaluates the effectiveness of the proposed DF models (See Table 2) and their enhanced variations in SDP. We compare their predictive performances with established models (prominent baseline and state of the art models) on benchmarked SDP datasets. The aim is to assess whether the implemented DF models outperform these benchmark models, and also to investigate the predictive performances of these DF models in the presence of class imbalance problem. For simplicity, the experiments and findings are analyzed and discussed in multiple stepwise stages/scenarios based on the objectives of this research. This was done to enhance the clarity of the observed empirical results. The first scenario assesses the prediction capabilities of the implemented DF models (CS-Forest, FPA, FT) in comparison to selected standard models, such as NB, kNN, and DT that are prominent in SDP. In the second scenario, the predictive performances of the homogeneous ensemble-enhanced variations of the DF models were investigated and compared. More importantly, in both cases, the predictive performances of the DF and their enhanced variants are analyzed and evaluated with and without a data sampling method. In this case, SMOTE data sampling is implemented to address the spontaneous class imbalance

problem present in SDP dataset. Finally, in the third scenario, the prediction capabilities of the proposed and implemented models are evaluated against standard and current models used on similar research data.

4.1. Scenario 1: Experimental Results of DF Models and the Baseline Classifiers

Table 5 highlights the comparative accuracy of decision forest (DF) models and selected baseline classifiers for software defect prediction (SDP) across multiple datasets. The DF models demonstrated strong predictive performance, achieving accuracy values that ranged between 70% and 91% across the tested datasets. These results indicate that DF models are robust and capable of generalizing effectively to diverse datasets, thereby reducing their dependence on specific training data. For instance, the performance on the PC1 dataset showed that CSForest and forest penalizing attributes (FPA) achieved an accuracy of 91.9%, while functional trees (FT) recorded 89.7%. Similarly, on the MW1 dataset, CSForest maintained a high accuracy of 90%, followed closely by FPA with 89.2% and FT with 88.8%. These results underscore the ability of DF models to consistently outperform baseline classifiers while adapting to different data distributions and complexities. The observed variation in accuracy values suggests that while DF models are generally effective, specific models like CSForest exhibit superior adaptability and precision in identifying defect-prone modules. Such performance trends highlight the practical applicability of DF models in real-world SDP scenarios, where datasets often vary in size, feature distributions, and defect ratios. The inclusion of advanced ensemble methods and tailored decision-making algorithms in DF models likely contributes to their strong generalization capabilities. These findings provide valuable insights into the role of DF models in achieving reliable and efficient defect prediction in software engineering.

Table 5. Accuracy values of DF models and prominent baseline models on the original datasets.

Accuracy	**Original Datasets**								
SDP Models	**CM1**	**KC1**	**KC3**	**MW1**	**PC1**	**PC3**	**PC4**	**PC5**	**Average**
CSForest	87.16	74.78	81.44	90.00	91.90	87.56	87.65	74.11	84.33
FPA	86.85	76.94	79.90	89.20	91.90	87.00	88.19	76.86	84.61
FT	83.49	76.42	82.47	88.80	89.70	85.70	88.89	74.81	83.79
NB	81.35	73.58	78.87	81.60	89.10	35.65	87.18	74.34	75.21
kNN	77.98	73.24	72.16	83.60	90.70	84.96	85.70	72.53	80.11
DT	81.04	74.18	79.38	90.40	91.50	84.96	88.34	73.40	82.90

In comparison to the baseline models, the DF models had comparable accuracy values. In most cases, the DF models recorded superior prediction accuracy values when compared to the baseline classifiers. For example, on the CM1 dataset, the least and best DF models (FT and CSForest, respectively) have accuracy values of 83.49% and 87.16%, respectively, which are +2.63% and +7.14% increments over the best baseline model, NB, with an accuracy value of 81.35%. On average across all the datasets, the DF models recorded better accuracy values compared to the baseline models. Specifically, FPA recorded a consistent performance across all the datasets with an average accuracy value of 84.61%, followed by CSForest (84.33%) and FT (83.79%), respectively. It is worthy to note that the baseline models also had good results with DT having the best average accuracy value of 82.9%. However, their accuracy values are less than the values of the DF models. Figure 2 presents the box-plot representation of the accuracy values of the DF models and the baseline models indicating variability of the accuracy values.

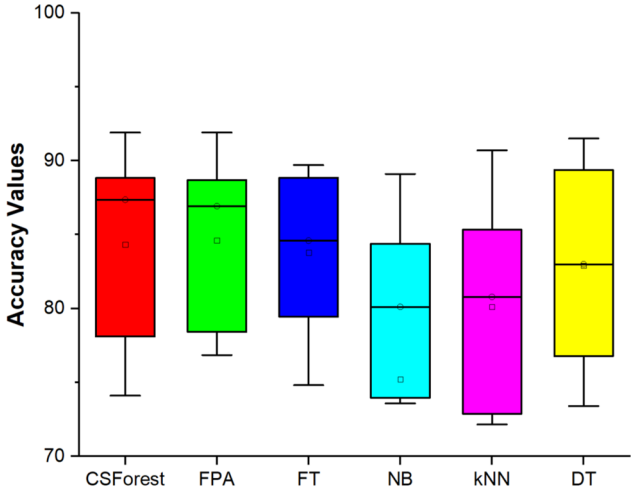

Figure 2. Box-plot representation of the accuracy values of DF and baseline models on original datasets.

To provide a comprehensive evaluation, Figure 2 illustrates the variability in accuracy using box-plot representations of the DF and baseline models. Despite good results from baseline models, the DF models demonstrated superior consistency and predictive performance. Given the frequent imbalance in software defect datasets, previous research recommends the use of complementary metrics such as AUC, precision, and recall for a holistic evaluation of SDP models [18,21,29,79]. Therefore, this study also assessed the DF models and baseline classifiers using AUC metrics to offer deeper insights into their performance beyond accuracy.

Table 6 provides a detailed comparative analysis of the AUC values obtained by DF models and baseline classifiers when applied to various SDP datasets. The results highlight the competitive performance of DF models, which demonstrated AUC values ranging from a maximum of 0.924 to a minimum of approximately 0.582. This variability indicates the robustness of DF models in adapting to different dataset characteristics while maintaining their ability to distinguish defective from non-defective instances effectively. The high AUC values recorded by models like CSForest and FPA on datasets such as PC4 (0.924 and 0.904, respectively) showcase their exceptional separability and predictive capabilities. Meanwhile, models like FT showed slightly reduced effectiveness with lower AUC values, such as 0.762 on PC4 and 0.689 on MW1, suggesting areas for potential improvement. Despite its lower relative performance, the FT model still achieved better-than-random classification, which can be significant in datasets with inherent noise or severe class imbalances. This underscores its potential utility in specific contexts where data complexities make other models less reliable. Importantly, the DF models consistently outperformed baseline classifiers in most cases, reflecting their superior ability to generalize across datasets. For example, even in scenarios where the baseline classifiers demonstrated strong performance, the DF models achieved higher AUC scores, indicating a greater degree of discriminatory power. The findings reinforce the reliability and robustness of DF models, particularly CSForest and FPA, in addressing the challenges of SDP tasks. By achieving higher AUC values, these models exhibit their capability to generalize effectively and handle imbalanced datasets. This analysis supports the adoption of DF models for applications requiring high accuracy and reliable classification, while also suggesting that future research could focus on optimizing models like FT to bridge performance gaps. Figure 3 presents the box-plot representation of the AUC values of the DF models and the baseline models indicating variability of the AUC values.

Table 6. AUC values of DF models and prominent baseline models on the original datasets.

AUC	Original Datasets								
SDP Models	**CM1**	**KC1**	**KC3**	**MW1**	**PC1**	**PC3**	**PC4**	**PC5**	**Average**
CSForest	0.725	0.686	0.745	0.802	0.833	0.832	0.924	0.778	0.791
FPA	0.652	0.713	0.713	0.702	0.775	0.778	0.904	0.776	0.752
FT	0.591	0.600	0.656	0.689	0.629	0.582	0.762	0.650	0.645
NB	0.645	0.681	0.662	0.779	0.790	0.766	0.833	0.690	0.731
kNN	0.521	0.633	0.539	0.607	0.679	0.643	0.697	0.654	0.622
DT	0.570	0.604	0.653	0.503	0.598	0.616	0.722	0.651	0.615

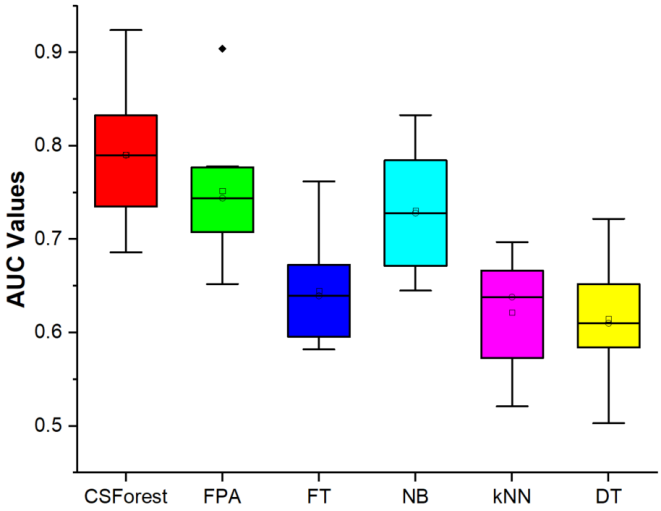

Figure 3. Box-plot representation of the AUC values of DF and baseline models on original datasets.

The variation in accuracy and AUC values of the DF models across multiple datasets highlights the model's sensitivity to data characteristics. This suggests that while the DF models can perform well under favorable conditions, they may struggle with more challenging datasets. One of such detrimental factors to the predictive performances of the DF models is the inherent class imbalance problem. The initial investigation revealed the predictive performances of the DF models on the original imbalanced software defect datasets. Hence, we further evaluated the predictive performances of the DF models on SMOTE balanced datasets. Tables 7 and 8 present the accuracy and AUC values of the DF models and the baseline models on the SMOTE-balanced software defect datasets.

Table 7. Accuracy values of DF models and prominent baseline models on the balanced datasets.

Accuracy	Balanced Datasets								
SDP Models	**CM1**	**KC1**	**KC3**	**MW1**	**PC1**	**PC3**	**PC4**	**PC5**	**Average**
CSForest	81.40	76.04	76.27	86.67	94.31	89.71	93.11	76.25	84.22
FPA	88.25	83.12	87.03	91.33	93.83	91.04	93.51	82.06	88.77
FT	84.39	74.17	82.60	86.00	88.86	87.43	91.49	76.17	83.89
NB	63.68	60.94	64.56	71.88	65.86	59.81	76.62	59.07	65.30
kNN	87.54	79.84	85.76	91.55	92.22	88.81	91.62	77.82	86.90
DT	85.44	78.57	81.65	87.77	92.86	88.23	91.17	79.03	85.59

Table 8. AUC values of DF models and prominent baseline models on the balanced datasets.

AUC	Balanced Datasets								
SDP Models	**CM1**	**KC1**	**KC3**	**MW1**	**PC1**	**PC3**	**PC4**	**PC5**	**Average**
CSForest	0.952	0.884	0.915	0.959	0.985	0.972	0.985	0.897	0.944
FPA	0.946	0.898	0.930	0.960	0.984	0.969	0.982	0.900	0.946
FT	0.844	0.742	0.826	0.860	0.889	0.874	0.915	0.762	0.839
NB	0.771	0.691	0.684	0.822	0.839	0.804	0.876	0.714	0.775
kNN	0.875	0.799	0.863	0.916	0.922	0.886	0.916	0.778	0.869
DT	0.848	0.824	0.835	0.869	0.934	0.897	0.918	0.799	0.866

Table 7 presents a detailed analysis of the predictive accuracy achieved by the DF models across various datasets, demonstrating their effectiveness in SDP. Among the DF models, CSForest achieved the highest accuracy of 94.31% on the PC1 dataset, closely followed by FPA with 93.83%. On the lower end, FT recorded an accuracy of 74.17% on the KC1 dataset, marking it as the least effective in this specific scenario. Averaging across all datasets, FPA outperformed other DF models with a mean accuracy of 88.77%, while CSForest and FT followed with average accuracy values of 84.22% and 83.89%, respectively.

When compared with baseline models, such as DT and kNN, FPA demonstrated superior average accuracy. However, DT and kNN occasionally exceeded the accuracy of CSForest and FT, though their performance is often compromised by their susceptibility to overfitting. Notably, addressing class imbalance in the datasets had a minimal impact on the accuracy values of DF models. For instance, FPA and FT showed incremental gains of +4.92% and +0.12%, respectively, after balancing the datasets, suggesting their inherent robustness to class imbalance.

Table 8 provides a further comparative analysis based on the area under the receiver operating characteristic curve (AUC) metric for the balanced datasets. The results highlight significant improvements in AUC values across all DF models after dataset balancing. The minimum AUC value observed was 0.742, while the maximum reached an impressive 0.985, indicating a high level of classification effectiveness. These AUC values suggest that DF models are proficient at separating defective and non-defective instances, making them particularly suitable for applications where minimal classification errors are critical.

Among the DF models, FPA retained its position as the top performer, achieving the highest average AUC value, followed by CSForest and FT. The AUC values for these models increased significantly on the balanced datasets, with FPA showing a +25.8% improvement, CSForest achieving a +19.34% increment, and FT demonstrating the most significant increase of +30.08%. While baseline models also recorded notable gains in AUC values after balancing, their performance remained consistently below that of the DF models, underscoring the superiority of DF models in achieving robust and reliable defect predictions across diverse datasets. Figures 4 and 5 present the box-plot representation of the accuracy and AUC values of the DF models and the baseline models on the balanced datasets.

Summarily, the performance metrics presented in Tables 6 and 7 affirm the potential and capability of DF models to generalize effectively across datasets, accommodate class imbalance, and deliver competitive prediction accuracy and AUC values compared to prominent baseline models. These findings underscore their applicability in real-world scenarios where accurate defect prediction is crucial for improving software quality and reliability.

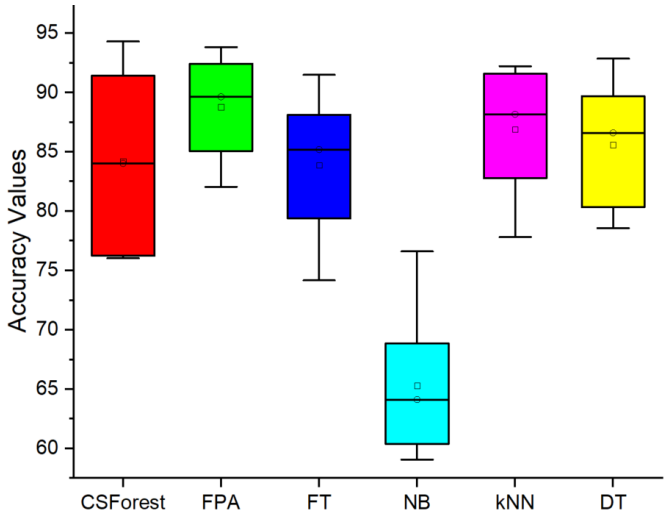

Figure 4. Box-plot representation of the accuracy values of DF and baseline models on balanced datasets.

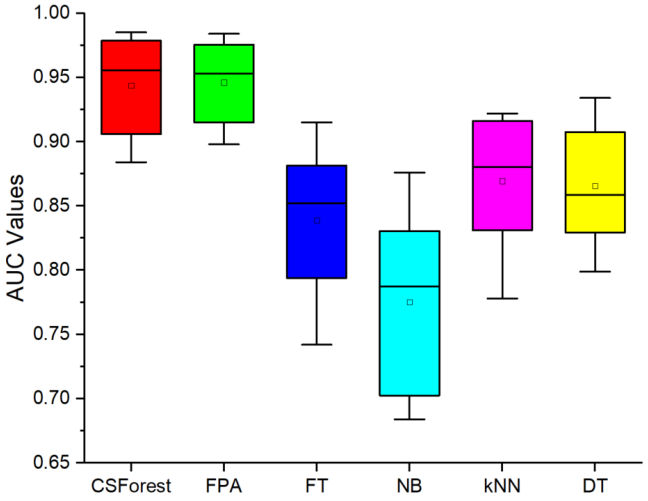

Figure 5. Box-plot representation of the AUC values of DF and baseline models on balanced datasets.

To improve the predictive accuracy and performance of the DF models, enhanced variants of the DF models based on homogeneous ensemble methods are proposed. Findings from existing studies have shown that ensemble methods can reduce performance variability and have the potential to cope with imbalanced datasets [17,19,75]. Hence, in the next subsection, the predictive performances of the homogeneous ensemble-enhanced DF models with or without the class imbalance on SDP dataset will be presented and analyzed.

4.2. Scenario 2: Experimental Results of Enhanced DF Models

Table 9 presents the comparison of the predictive accuracy values of the DF models and their enhanced variations on the original software defect datasets. From Table 9, it can be observed that there is improvement (increments) in the predictive accuracy of the enhanced DF models on the original datasets. Specifically, the bagged and boosted variants of the CSForest, FPA, and FT showed improved accuracy values across all the datasets and on average. In some cases, the performances were comparable while significant at some other points. The highest and lowest accuracy values observed were 91.9% and 76.56%. This indicates the applicability and acceptability of the accuracy values of the enhanced DF models on the original dataset.

Table 9. Accuracy values of DF models and their enhanced variants on original datasets.

Accuracy	Original Dataset								
	CSForest	**Bagged CSForest**	**BoostCSForest**	**FPA**	**Bagged FPA**	**BoostFPA**	**FT**	**Bagged FT**	**BoostFT**
CM1	87.16	87.16	87.16	86.85	86.85	86.85	83.49	85.32	85.93
KC1	74.78	76.94	76.94	76.94	78.14	78.14	76.42	77.80	77.80
KC3	81.44	81.44	81.44	79.90	80.93	80.93	82.47	81.96	81.96
MW1	90.00	90.00	90.40	89.20	89.60	89.60	88.80	88.80	88.80
PC1	91.90	91.90	91.61	91.90	91.90	91.90	89.70	91.31	90.72
PC3	87.56	87.56	88.02	87.00	87.65	87.65	85.70	85.52	84.22
PC4	87.65	89.43	89.43	88.19	89.88	89.36	88.89	90.21	91.14
PC5	74.11	77.50	77.50	76.86	82.66	82.38	74.81	76.56	76.56
Average	84.33	85.24	85.31	84.61	85.95	85.85	83.79	84.69	84.64

On the CSForest model, the bagged and boosted CSForest recorded +1.09% and +1.17% increments in their predictive accuracy when compared with CSForest on the original dataset. A similar occurrence was observed with FPA and FT, where their respective bagged and boosted variants had +1.59% and +1.47%, and +1.07% and 1.02% increments on accuracy values, respectively. These observations indicated improvements in the predictive accuracy values of the enhanced DF models. That is, the performance accuracy of the DF models was improved by the ensemble methods.

For generalizability, the predictive performance of the DF models and their enhanced variations were further assessed based on the AUC values as presented in Table 10. As observed, there are significant improvements in the AUC values of the base DF models and their enhanced variants. The highest and lowest AUC values observed were 0.933 and 0.656. This indicates the enhanced DF models can in most cases make precise predictions on the original dataset. For instance, on CSForest, both BaggedCSForest and BoostCSForest recorded +3.65% increments in their respective AUC values as compared with the baseline CSForest. The variations (increments) were higher in the case of FPA as the BaggedFPA and BoostFPA had a +5.54% and +3.89% increments in AUC values, respectively. FT had the most significant improvement in its AUC values with BaggedFT and BoostFT models recording +19.31% and +15.10% increments in AUC values, respectively.

Table 10. AUC values of DF models and their enhanced variants on original datasets.

Accuracy	Original Dataset								
	CSForest	**Bagged CSForest**	**BoostCSForest**	**FPA**	**Bagged FPA**	**BoostFPA**	**FT**	**Bagged FT**	**BoostFT**
CM1	0.725	0.728	0.728	0.652	0.714	0.714	0.591	0.723	0.712
KC1	0.686	0.808	0.808	0.713	0.742	0.742	0.600	0.718	0.694
KC3	0.745	0.751	0.751	0.713	0.730	0.730	0.656	0.711	0.659
MW1	0.802	0.863	0.863	0.702	0.756	0.702	0.689	0.735	0.656
PC1	0.833	0.861	0.861	0.775	0.851	0.883	0.629	0.800	0.794
PC3	0.832	0.835	0.835	0.778	0.827	0.778	0.582	0.777	0.753
PC4	0.924	0.925	0.925	0.904	0.933	0.922	0.762	0.921	0.931
PC5	0.778	0.785	0.785	0.776	0.793	0.776	0.650	0.770	0.739
Average	0.791	0.820	0.820	0.752	0.793	0.781	0.645	0.769	0.742

It was also observed that there is not any clear-cut superiority among the enhanced variations as each ensemble method improved the predictive performances (accuracy and AUC values) of the baseline DF models. However, enhanced variants of FPA still had the best accuracy values and enhanced variants of CSForest had the highest AUC values as compared with other variants on both metrics (accuracy and AUC). This can be attributed to the high predictive performances of the DF models as observed and reported in Section 4.1.

Summary, it was observed that the enhancement of the DF models based on the homogeneous (bagging and boosting) ensemble method did not only improve their respective predictive performances but also accommodate the class imbalance problem present in the original software defect dataset. This shows that the ensemble DF models are more robust to noise and data quality problems than the baseline DF models. Figures 6 and 7 depict the graphical representations of the predictive accuracy and AUC values of the enhanced DF models on the original defect datasets, respectively.

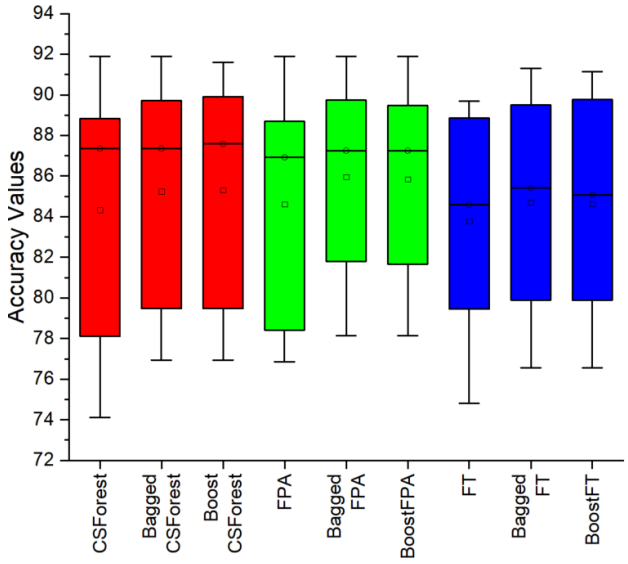

Figure 6. Box-plot representation of the accuracy values of enhanced DF and baseline DF models on original datasets.

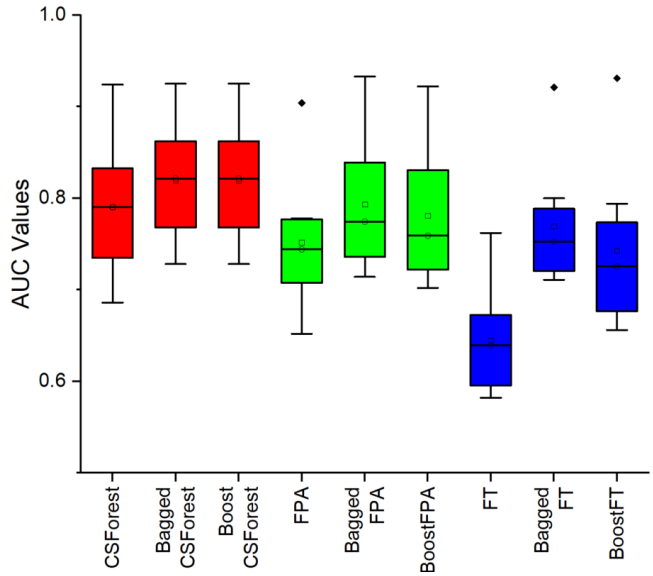

Figure 7. Box-plot representation of the AUC values of enhanced DF and baseline DF models on original datasets.

Like Scenario 1, we further evaluated the predictive performances of the enhanced DF models on SMOTE balanced datasets. Tables 11 and 12 present the accuracy and AUC values of the enhanced DF models and the baseline DF models on the SMOTE-balanced software defect datasets.

Table 11. Accuracy values of DF models and their enhanced variants on balanced datasets.

Accuracy	Balanced Dataset								
	CSForest	**Bagged CSForest**	**BoostCSForest**	**FPA**	**Bagged FPA**	**BoostFPA**	**FT**	**Bagged FT**	**BoostFT**
CM1	81.40	85.44	90.18	88.25	90.00	90.06	84.39	85.80	88.60
KC1	76.04	79.67	81.39	83.12	83.35	83.35	74.17	79.55	79.21
KC3	76.27	79.75	85.76	87.03	87.03	87.03	82.60	84.49	85.13
MW1	86.67	86.67	93.11	91.33	92.44	93.33	86.00	87.56	89.56
PC1	94.31	94.23	96.87	93.83	93.99	95.83	88.86	92.07	93.27
PC3	89.71	90.19	92.20	91.04	92.26	92.26	87.43	89.77	89.50
PC4	93.11	93.02	95.14	93.51	94.19	94.60	91.49	92.84	93.96
PC5	76.25	79.88	82.50	82.06	83.38	83.38	76.17	79.52	77.54
Average	84.22	86.11	89.64	88.77	89.58	89.98	83.89	86.45	87.10

Table 12. AUC values of DF models and their enhanced variants on balanced datasets.

Accuracy	Balanced Dataset								
	CSForest	**Bagged CSForest**	**BoostCSForest**	**FPA**	**Bagged FPA**	**BoostFPA**	**FT**	**Bagged FT**	**BoostFT**
CM1	0.952	0.955	0.962	0.946	0.962	0.964	0.844	0.929	0.943
KC1	0.884	0.898	0.883	0.898	0.907	0.892	0.742	0.858	0.855
KC3	0.915	0.920	0.932	0.930	0.941	0.936	0.826	0.900	0.912
MW1	0.959	0.961	0.972	0.960	0.970	0.973	0.860	0.937	0.961
PC1	0.985	0.986	0.991	0.984	0.988	0.989	0.889	0.974	0.983
PC3	0.972	0.972	0.969	0.969	0.975	0.974	0.874	0.957	0.955
PC4	0.985	0.985	0.987	0.982	0.987	0.987	0.915	0.973	0.981
PC5	0.897	0.955	0.956	0.900	0.910	0.903	0.762	0.878	0.857
Average	0.944	0.954	0.957	0.946	0.955	0.952	0.839	0.926	0.931

Table 10 presents a comparative analysis of the predictive accuracy of the original DF models and their enhanced counterparts on SMOTE-balanced software defect datasets. The results highlight notable improvements in the predictive performance of the enhanced DF models when compared to their original versions. Specifically, the bagged and boosted variations of CSForest, FPA, and FT demonstrated improved accuracy across all the datasets, with incremental gains that reflect the effectiveness of ensemble methods. The observed accuracy values ranged between 94.6% (highest) and 77.54% (lowest), showcasing the reliability and practical applicability of the enhanced DF models for defect prediction tasks. The results suggest that combining data sampling techniques such as SMOTE and ensemble techniques such as bagging and boosting can refine the performance of base models by aggregating predictions from multiple iterations or models, thereby addressing inconsistencies and enhancing overall accuracy.

Focusing on CSForest, the bagged and boosted variants achieved an accuracy increment of +2.24% and +6.44%, respectively, compared to the original CSForest model.

Similarly, the bagged and boosted FPA models recorded respective improvements of +0.91% and +1.36% in their accuracy scores. The FT model also benefited, with enhancements of +3.05% and +3.82% observed for its bagged and boosted versions. These increments affirm that the combination of SMOTE for addressing class imbalance and ensemble methods can significantly enhance the predictive performance of base models by mitigating overfitting, balancing training data, and emphasizing underrepresented patterns. Figure 8 presents the graphical representation of the accuracy values of enhanced DF models on the balanced datasets.

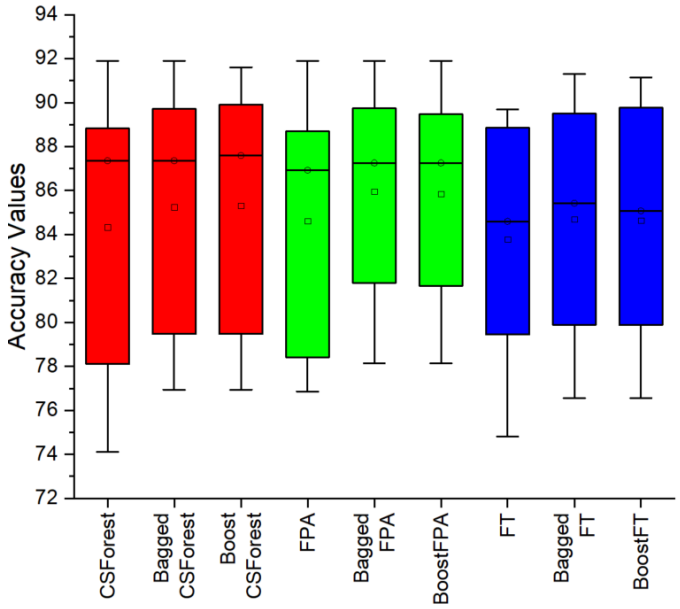

Figure 8. Box-plot representation of the accuracy values of enhanced DF and baseline DF models on balanced datasets.

The consistent improvement across all DF models, coupled with the significant yet dataset-specific gains, underscores the potential of ensemble approaches in software defect prediction. This indicates that ensemble-enhanced DF models not only outperform their original counterparts but also maintain robust performance across varied datasets, making them a compelling choice for real-world applications in software reliability engineering.

To evaluate the generalizability of the enhanced DF models, their predictive performance was further assessed using the AUC metric, as summarized in Table 12. Significant improvements were observed in the AUC values of the base DF models and their enhanced counterparts. The AUC values for the enhanced DF models ranged from 0.989 (highest) to 0.855 (lowest), underscoring their ability to provide precise and reliable predictions across various datasets.

For instance, enhancements in CSForest were notable, with both BaggedCSForest and BoostCSForest achieving a +1.10% and +1.36% increment in their respective AUC values compared to the baseline CSForest. A similar trend was evident with the FPA model, where BaggedFPA and BoostFPA recorded increments of +0.94% and +0.65%, respectively. Among the enhanced DF models on the balanced datasets, FT still demonstrated the most pronounced improvements, with BaggedFT and BoostFT models achieving AUC value gains of +10.34% and +10.95%, respectively.

While no single enhanced DF variant consistently outperformed the others across all metrics, each ensemble technique—whether bagging or boosting—proved effective in improving both the predictive accuracy and AUC values of the baseline DF models. Specifically, FPA's enhanced variants consistently achieved the highest accuracy values,

while CSForest's enhanced versions demonstrated the highest AUC values across the datasets. This distinction can be attributed to the inherent strengths of these models in different performance metrics, as outlined in previous sections. Furthermore, graphical representations in Figure 9 illustrate the incremental performance improvements achieved by the enhanced DF models in terms of AUC values, reinforcing the effectiveness of these advanced methodologies.

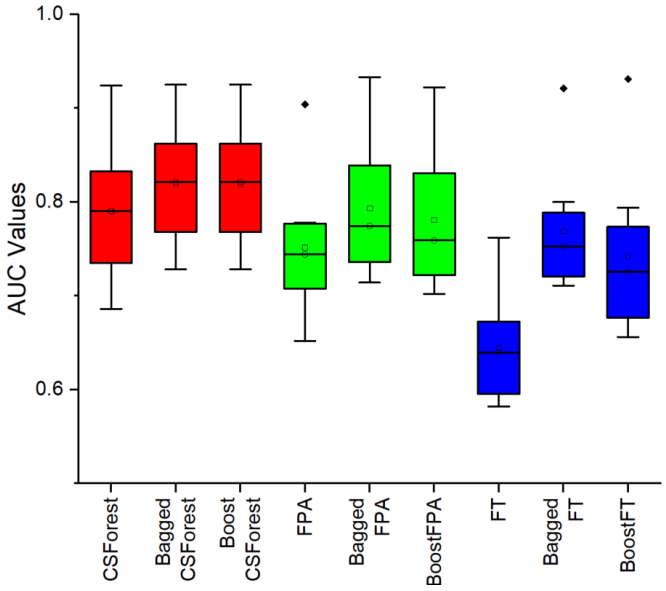

Figure 9. Box-plot representation of the accuracy values of enhanced DF and baseline DF models on original datasets.

The application of either homogeneous ensemble methods such as bagging and boosting and their combination with a SMOTE not only improved the predictive capabilities of the DF models but also demonstrated resilience to issues such as class imbalance and data quality challenges. This robustness highlights the suitability of ensemble DF models and data sampling for real-world scenarios where datasets may exhibit noise or skewed distributions.

In summary, the ensemble-enhanced DF models show significant promise for SDP tasks by improving baseline model performance and handling inherent dataset challenges, making them reliable and adaptable tools in the domain of ML and software reliability engineering.

The next subsection compared and analyzed the predictive performances of the top-performing enhanced DF models with existing SDP models. The essence of this comparison is to validate the performance of the enhanced DF models and their applicability in SDP.

4.3. Scenario 3: Comparison of DF Models and Its Enhanced Variations with Current SDP Models

Table 13 highlights the comparative evaluation of the proposed enhanced decision forest (DF) models against existing approaches in terms of prediction accuracy. The experimental results leverage benchmark studies, including methodologies detailed in [54,80–82], to contextualize the performance of the enhanced DF models. These existing techniques encompass a range of computational approaches, many of which have been recognized for their strong predictive abilities.

For instance, the homogeneous ensemble models described in [80] utilized Bagged Logistic Regression (BaggedLR) and AdaboostSVM for software defect prediction (SDP), achieving commendable results. Similarly, Ref. [83] introduced a heterogeneous stacking ensemble method tailored for SDP, leveraging the strengths of diverse classifiers in a meta-

ensemble framework. While these models demonstrated effective performance across specific datasets, the proposed enhanced DF models outperformed them in most datasets, showcasing their ability to generalize and adapt to varying data characteristics.

Table 13. Accuracy Value Comparison of DF Models with Existing SDP Methods.

SDP Models	CM1	KC1	KC3	MW1	PC1	PC3	PC4	PC5
* BaggedCSForest	85.44	79.67	79.75	86.67	94.23	90.19	93.02	79.88
* BoostCSForest	90.18	81.39	85.76	93.11	96.87	92.20	95.14	82.50
* BaggedFPA	90.00	83.35	87.03	92.44	93.99	92.26	94.19	83.38
* BoostFPA	90.06	83.35	87.03	93.33	95.83	92.26	94.60	83.38
* BaggedFT	85.80	79.55	84.49	87.56	92.07	89.77	92.84	79.52
* BoostFT	88.60	79.21	85.13	89.56	93.27	89.50	93.96	77.54
CG-NB [19]	84.71	77.45	78.35	90.00	92.19	87.84	90.16	77.33
CG-DT [19]	85.32	76.85	79.90	90.00	92.05	87.94	88.98	78.16
CG-kNN [19]	85.32	77.62	80.41	90.40	91.90	86.99	89.09	76.45
BaggedLR [80]	74.00	-	76.00	-	81.00	75.00	83.00	68.00
AdaboostSVM [80]	75.00	-	77.00	-	79.00	74.00	81.00	68.00
kStar [81]	77.55	72.20	75.86	82.67	86.27	82.59	81.89	69.88
CS-Forest [54]	82.53	-	81.44	88.33	91.16	84.77	88.88	-
Rotation Tree [54]	83.33	-	70.61	86.60	91.07	85.54	86.69	-
Dagging_NB [82]	70.80	66.90	70.20	75.30	78.60	78.50	81.60	69.90
Dagging_DT [82]	59.60	68.10	64.50	77.10	76.70	78.20	89.70	76.50
Dagging_kNN [82]	61.10	67.90	62.30	72.20	78.50	75.30	84.40	76.40

* indicates the method proposed in this study.

In comparison with advanced instance-learning-based models such as kStar [81] and tree-structure-enhanced techniques like CS-Forest and rotation forest [54], the proposed DF models displayed comparable or superior prediction accuracy. Additionally, the enhanced DF models surpassed recent methods based on dagging meta-learners, as proposed in [82], further affirming their robustness and practical applicability in SDP tasks.

Table 14 extends this comparative analysis by evaluating the AUC performance of the enhanced DF models relative to existing methodologies. The findings corroborate those from Table 10, with the DF models exhibiting superior AUC values across most datasets. The comparative techniques, including homogeneous ensemble models [80], heterogeneous ensembles [83], tree-based approaches [84], instance-learning methods [81], and dagging meta-learners [82], performed well but generally lagged behind the DF models. The enhanced DF models' ability to achieve high AUC values highlights their effectiveness in distinguishing between defective and non-defective software instances, making them particularly suitable for imbalanced datasets where class separability is crucial. These results underscore the DF models' superiority in both accuracy and AUC metrics, emphasizing their potential as reliable tools for SDP and reinforcing their efficacy over a diverse array of existing computational approaches.

Furthermore, aside the ability of the proposed DF models in addressing inherent SDP challenges like class imbalance, cost-aware classification, and interpretability, the SDP models can be integrated into modern software engineering processes. Specifically, the proposed DF models can be integrated into CI/CD pipelines and Agile workflows as these models can provide real-time defect predictions, enabling developers to take proactive measures. By combining SMOTE-based sampling, and DF models, software teams

can achieve higher defect prediction accuracy, reduced testing overhead, and improved software reliability, leading to more robust software products in modern development environments.

Table 14. AUC value comparison of CG-based models with existing SDP methods.

SDP Models	CM1	KC1	KC3	MW1	PC1	PC3	PC4	PC5
* BaggedCSForest	0.955	0.898	0.920	0.961	0.986	0.972	0.985	0.955
* BoostCSForest	0.962	0.883	0.932	0.972	0.991	0.969	0.987	0.956
* BaggedFPA	0.962	0.907	0.941	0.970	0.988	0.975	0.987	0.910
* BoostFPA	0.964	0.892	0.936	0.973	0.989	0.974	0.987	0.903
* BaggedFT	0.929	0.858	0.900	0.937	0.974	0.957	0.973	0.878
* BoostFT	0.943	0.855	0.912	0.961	0.983	0.955	0.981	0.857
CG-NB [19]	0.704	0.732	0.723	0.731	0.885	0.846	0.937	0.803
CG-DT [19]	0.717	0.723	0.731	0.723	0.861	0.834	0.925	0.806
CG-kNN [19]	0.689	0.733	0.703	0.724	0.864	0.830	0.935	0.800
BaggedLR [80]	0.650	-	0.660	-	0.770	0.740	0.870	0.680
AdaboostSVM [80]	0.680	-	0.660	-	0.760	0.730	0.820	0.680
Stacking (NB, MLP, J48) [83]	-	-	-	-	0.749	-	-	-
Stacking (NB, MLP, J48)+SMOTE [83]	-	-	-	-	0.871	-	-	-
J48 [84]	0.594	0.689	-	-	0.668	-	-	-
kStar [81]	0.538	0.651	0.528	0.543	0.673	0.749	0.734	0.629
Dagging_NB [82]	0.708	0.669	0.702	0.753	0.786	0.785	0.816	0.699
Dagging_DT [82]	0.596	0.681	0.645	0.771	0.767	0.782	0.897	0.765
Dagging_kNN [82]	0.611	0.679	0.623	0.722	0.785	0.753	0.844	0.764

* indicates the method proposed in this study.

5. Threat to Validity

The current research, like prior studies, faces specific constraints that may affect its findings and broader applicability. Key threats to validity are categorized into two areas: external validity, which concerns the generalizability of results beyond the specific dataset or context, and internal validity, which pertains to the accuracy of causal inferences made within the study.

External validity is particularly critical, as it influences the applicability of the predictive models to broader populations or systems. For instance, models developed using open-source tools like WEKA might produce results that are context-specific and may not perform consistently across diverse platforms or datasets. The limited size of the dataset used in this study amplifies these concerns, as smaller datasets often struggle to represent the variability found in real-world environments.

Internal validity challenges arise from the potential for confounding variables that can distort the relationship between independent and dependent variables. For example, gaps in the data or non-cumulative numerical values necessitate careful handling to avoid biases in the analysis. The misinterpretation of causal links between software metrics and defect occurrences can also impact the reliability of conclusions drawn from the models.

The effectiveness of the proposed prediction models is context-dependent, meaning they perform optimally under specific operational conditions but may falter when applied to new systems with differing characteristics. To address these limitations, future

research should incorporate additional replicated experiments across multiple platforms and datasets. Such replication could strengthen the robustness and generalizability of the models, mitigating external validity threats. Despite these limitations, this study provides valuable insights into the application of machine learning techniques for software reliability forecasting, particularly in understanding the influence of past dataset failures on predictive accuracy. These findings lay a foundation for future research aimed at improving model scalability, adaptability, and cross-context reliability.

6. Conclusions and Future Work

The application of AI and ML techniques in software engineering is increasingly recognized as a significant and promising area of research. The early identification of software defects, even in the preliminary stages of development, is essential for ensuring high-quality software products. Although challenging, detecting defects at these stages can significantly reduce the likelihood of critical errors in the final product. Over the years, numerous SDP models have been proposed, but the rapid growth in modern software systems' complexity, interdependencies, and volume necessitates more robust and sophisticated approaches. In this study, DF models, specifically CSForest, FPA, and FT, were applied to the task of SDP. These models were evaluated against prominent baseline classifiers, including DT, kNN, and NB to assess their predictive capabilities. Additionally, SMOTE was utilized to mitigate the class imbalance inherent in software defect datasets, which often affects predictive performance.

The results of this research demonstrated that DF models outperformed baseline classifiers in most scenarios. These models achieved high predictive accuracy and robustness, showcasing their ability to generalize across diverse datasets. Furthermore, when applied to SMOTE-balanced datasets, the DF models exhibited significant improvements in predictive performance compared to their performance on the original datasets. This finding underscores the importance of addressing class imbalance as a critical step in enhancing model performance.

To further improve the performance of DF models, enhanced homogeneous ensemble variants incorporating bagging and boosting techniques were also explored. The experimental findings revealed that these ensemble-based DF models achieved superior predictive performance, especially on SMOTE-balanced datasets, surpassing existing state-of-the-art SDP models based on various computational methods. This suggests that ensemble techniques provide an effective mechanism for addressing variability and improving predictive reliability in SDP tasks.

This study highlights the potential of DF models as a valuable tool for SDP and similar ML applications. Their ability to accommodate class imbalance and deliver competitive predictions suggests that they are well-suited for complex and imbalanced datasets. Moreover, incorporating data sampling techniques, such as SMOTE, was demonstrated to be an effective approach for mitigating latent data quality issues.

For future work, the research will extend to other types of defective datasets, such as those available in the PROMISE repository, to validate the models' generalizability. Additionally, challenges related to data quality, including high dimensionality and severe class imbalance, will be addressed. Efforts will focus on developing advanced ML models capable of effectively handling these issues to further improve the accuracy and reliability of SDP systems. This research not only reinforces the utility of DF models but also opens new avenues for exploration and enhancement in the field of software reliability prediction.

Author Contributions: Conceptualization, F.E.U.-H. and A.O.B.; methodology, F.E.U.-H. and A.O.B.; software, H.M. and H.A.M.; validation, L.F.C., S.B. and R.A.O.; formal analysis, A.O.B. and A.G.A.; investigation, F.E.U.-H. and A.O.B.; resources, L.F.C. and S.B.; data curation, H.A.M., A.G.A. and

H.M.; writing—original draft preparation, F.E.U.-H. and A.O.B.; writing—review and editing, H.M., L.F.C. and R.A.O.; visualization, A.O.B. and A.G.A.; supervision, F.E.U.-H. and H.A.M.; project administration, L.F.C. and S.B.; and funding acquisition, A.O.B. and L.F.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: This research/paper was fully supported by Universiti Teknologi PETRONAS, under the STIRF Research Grant Scheme (015LA0-049) and Ministry of Higher Education, Malaysia Fundamental Research Grant Scheme (015MA0-170).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Laplante, P.A.; Kassab, M. *Requirements Engineering for Software and Systems*; Auerbach Publications: Boca Raton, FL, USA, 2022.
2. Westfall, L. Software requirements engineering: What, why, who, when, and how. *Softw. Qual. Prof.* **2005**, *7*, 17.
3. Luo, L.; He, Q.; Xie, J.; Yang, D.; Wu, G. Investigating the relationship between project complexity and success in complex construction projects. *J. Manag. Eng.* **2017**, *33*, 04016036.
4. Nan, N.; Harter, D.E. Impact of budget and schedule pressure on software development cycle time and effort. *IEEE Trans. Softw. Eng.* **2009**, *35*, 624–637.
5. Menzies, T.; Nichols, W.; Shull, F.; Layman, L. Are delayed issues harder to resolve? Revisiting cost-to-fix of defects throughout the lifecycle. *Empir. Softw. Eng.* **2017**, *22*, 1903–1935.
6. Humphrey, W.S. Why big software projects fail: The 12 key questions. In *Software Management*; John Wiley & Sons: Hoboken, NJ, USA, 2006; pp. 21–27.
7. Humphrey, W.S. *Psp (sm): A Self-Improvement Process for Software Engineers*; Addison-Wesley Professional: Boston, MA, USA, 2005.
8. Wu, W.; Wang, S.; Liu, B.; Shao, Y.; Xie, W. A novel software defect prediction approach via weighted classification based on association rule mining. *Eng. Appl. Artif. Intell.* **2024**, *129*, 107622.
9. Leszak, M.; Perry, D.E.; Stoll, D. A case study in root cause defect analysis. In Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, 4–11 June 2000; pp. 428–437.
10. Catal, C. Software fault prediction: A literature review and current trends. *Expert Syst. Appl.* **2011**, *38*, 4626–4636. [CrossRef]
11. Koçan, M.; Yıldız, E. Evaluation of Consumer Complaints: A Case Study Using MAXQDA 2020 Data Analysis Software. *Çankırı Karatekin Üniversitesi İktisadi Ve İdari Bilim. Fakültesi Derg.* **2024**, *14*, 266–289. [CrossRef]
12. Kumar, G.; Imam, A.A.; Basri, S.; Hashim, A.S.; Naim, A.G.H.; Capretz, L.F.; Balogun, A.O.; Mamman, H. Ensemble Balanced Nested Dichotomy Fuzzy Models for Software Requirement Risk Prediction. *IEEE Access* **2024**, *12*, 146225–146243.
13. Bayramova, T.A.; Malikova, N.C. Developing a conceptual model for improving the software system reliability. *Probl. Inf. Soc.* **2024**, *15*, 42–56. [CrossRef]
14. Phung, K.; Ogunshile, E.; Aydin, M. Error-type—A novel set of software metrics for software fault prediction. *IEEE Access* **2023**, *11*, 30562–30574. [CrossRef]
15. Li, Z.; Niu, J.; Jing, X.-Y. Software defect prediction: Future directions and challenges. *Autom. Softw. Eng.* **2024**, *31*, 19.
16. Mashhadi, E.; Chowdhury, S.; Modaberi, S.; Hemmati, H.; Uddin, G. An empirical study on bug severity estimation using source code metrics and static analysis. *J. Syst. Softw.* **2024**, *217*, 112179.
17. Malek, A.; Balogun, A.O.; Basri, S.; Abdullahi, A.; Imam, A.K.A.; Alazzawi, A.K.; Adeyemo, V.E.; Kumar, G. Empirical Analysis of Threshold Values for Rank-Based Filter Feature Selection Methods in Software Defect Prediction. *J. Eng. Sci. Technol.* **2023**, *18*, 187–209.
18. Ali, M.; Mazhar, T.; Arif, Y.; Al-Otaibi, S.; Ghadi, Y.Y.; Shahzad, T.; Khan, M.A.; Hamam, H. Software defect prediction using an intelligent ensemble-based model. *IEEE Access* **2024**, *12*, 20376–20395.
19. Bashir, A.T.; Balogun, A.O.; Adigun, M.O.; Ajagbe, S.A.; Capretz, L.F.; Awotunde, J.B.; Mojeed, H.A. Cascade Generalization-Based Classifiers for Software Defect Prediction. In Proceedings of the Computer Science Online Conference, Online, 25–28 April 2024; Springer: Berlin/Heidelberg, Germany, 2024; pp. 22–42.
20. Odejide, B.J.; Bajeh, A.O.; Balogun, A.O.; Alanamu, Z.O.; Adewole, K.S.; Akintola, A.G.; Salihu, S.A.; Usman-Hamza, F.E.; Mojeed, H.A. An empirical study on data sampling methods in addressing class imbalance problem in software defect prediction. In Proceedings of the Computer Science Online Conference, Online, 26–26 April 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 594–610.

21. Balogun, A.O.; Basri, S.; Mahamad, S.; Capretz, L.F.; Imam, A.A.; Almomani, M.A.; Adeyemo, V.E.; Kumar, G. A Novel Rank Aggregation-Based Hybrid Multifilter Wrapper Feature Selection Method in Software Defect Prediction. *Comput. Intell. Neurosci.* **2021**, *2021*, 5069016. [PubMed]
22. Balogun, A.O.; Basri, S.; Abdulkadir, S.J.; Hashim, A.S. Performance analysis of feature selection methods in software defect prediction: A search method approach. *Appl. Sci.* **2019**, *9*, 2764. [CrossRef]
23. Nama, P. Integrating AI in testing automation: Enhancing test coverage and predictive analysis for improved software quality. *World J. Adv. Eng. Technol. Sci.* **2024**, *13*, 769–782.
24. Batarseh, F.A.; Gonzalez, A.J. Predicting failures in agile software development through data analytics. *Softw. Qual. J.* **2018**, *26*, 49–66.
25. Khan, M.F.I.; Masum, A.K.M. Predictive Analytics and Machine Learning for Real-Time Detection Of Software Defects And Agile Test Management. *Educ. Adm. Theory Pract.* **2024**, *30*, 1051–1057.
26. Croft, R.; Babar, M.A.; Kholoosi, M.M. Data quality for software vulnerability datasets. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023; pp. 121–133.
27. Shiri Harzevili, N.; Boaye Belle, A.; Wang, J.; Wang, S.; Jiang, Z.M.; Nagappan, N. A Systematic Literature Review on Automated Software Vulnerability Detection Using Machine Learning. *ACM Comput. Surv.* **2024**, *57*, 1–36.
28. Wang, J.; Liu, Y.; Li, P.; Lin, Z.; Sindakis, S.; Aggarwal, S. Overview of data quality: Examining the dimensions, antecedents, and impacts of data quality. *J. Knowl. Econ.* **2024**, *15*, 1159–1178.
29. Balogun, A.O.; Basri, S.; Said, J.A.; Adeyemo, V.E.; Imam, A.A.; Bajeh, A.O. Software defect prediction: Analysis of class imbalance and performance stability. *J. Eng. Sci. Technol.* **2019**, *14*, 3294–3308.
30. Pandey, S.; Kumar, K. Software fault prediction for imbalanced data: A survey on recent developments. *Procedia Comput. Sci.* **2023**, *218*, 1815–1824.
31. Balogun, A.O.; Odejide, B.J.; Bajeh, A.O.; Alanamu, Z.O.; Usman-Hamza, F.E.; Adeleke, H.O.; Mabayoje, M.A.; Yusuff, S.R. Empirical analysis of data sampling-based ensemble methods in software defect prediction. In Proceedings of the International Conference on Computational Science and Its Applications, Malaga, Spain, 4–7 July 2022; pp. 363–379.
32. Pachouly, J.; Ahirrao, S.; Kotecha, K.; Selvachandran, G.; Abraham, A. A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. *Eng. Appl. Artif. Intell.* **2022**, *111*, 104773.
33. Yin, L.; Sun, Z.; Gao, F.; Liu, H. Deep forest regression for short-term load forecasting of power systems. *IEEE Access* **2020**, *8*, 49090–49099.
34. Adnan, M.N.; Islam, M.Z. Forest PA: Constructing a decision forest by penalizing attributes used in previous trees. *Expert Syst. Appl.* **2017**, *89*, 389–403.
35. Siers, M.J.; Islam, M.Z. Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem. *Inf. Syst.* **2015**, *51*, 62–71.
36. Chen, Y.; Yang, X.; Dai, H.-L. Cost-sensitive continuous ensemble kernel learning for imbalanced data streams with concept drift. *Knowl.-Based Syst.* **2024**, *284*, 111272.
37. Wang, N.; Zhao, S.; Wang, S. A novel clustering-based resampling with cost-sensitive boosting method to model and map wildfire susceptibility. *Reliab. Eng. Syst. Saf.* **2024**, *242*, 109742.
38. Cardoso, P.; Guillerme, T.; Mammola, S.; Matthews, T.J.; Rigal, F.; Graco-Roza, C.; Stahls, G.; Carlos Carvalho, J. Calculating functional diversity metrics using neighbor-joining trees. *Ecography* **2024**, *2024*, e07156.
39. Balogun, A.O.; Adewole, K.S.; Bajeh, A.O.; Jimoh, R.G. Cascade generalization based functional tree for website phishing detection. In Proceedings of the Advances in Cyber Security: Third International Conference, ACeS 2021, Penang, Malaysia, 24–25 August 2021; Revised Selected Papers 3; Springer: Singapore, 2021; pp. 288–306.
40. Luong, A.V.; Vu, T.H.; Nguyen, P.M.; Van Pham, N.; McCall, J.; Liew, A.W.-C.; Nguyen, T.T. A homogeneous-heterogeneous ensemble of classifiers. In Proceedings of the Neural Information Processing: 27th International Conference, ICONIP 2020, Bangkok, Thailand, 18–22 November 2020; Proceedings, Part V 27; Springer: Cham, Switzerland, 2020; pp. 251–259.
41. Ramakrishna, M.T.; Venkatesan, V.K.; Izonin, I.; Havryliuk, M.; Bhat, C.R. Homogeneous adaboost ensemble machine learning algorithms with reduced entropy on balanced data. *Entropy* **2023**, *25*, 245. [CrossRef] [PubMed]
42. Jhala, R.; Majumdar, R. Software model checking. *ACM Comput. Surv. (CSUR)* **2009**, *41*, 1–54.
43. Leokhin, Y.; Fatkhulin, T.; Kozhanov, M. Research of Static Application Security Testing Technique Problems and Methods for Solving Them. In Proceedings of the 2024 Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia, 12–14 March 2024; pp. 1–7.
44. Smidts, C.; Stutzke, M.; Stoddard, R.W. Software reliability modeling: An approach to early reliability prediction. *IEEE Trans. Reliab.* **1998**, *47*, 268–278.
45. Cortellessa, V.; Singh, H.; Cukic, B. Early reliability assessment of UML based software models. In Proceedings of the 3rd International Workshop on Software and Performance, Rome, Italy, 24–27 July 2002; pp. 302–309.

46. Gaffney, J.; Davis, C.F. An approach to estimating software errors and availability. In Proceedings of the Eleventh Minnowbrook Workshop on Software ReliabilitySPC-TR-88-007, Version 1.0, Blue Mountain Lake, NY, USA, 26–29 July 1988.
47. Gaffney, J.; Pietrolewiez, J. An automated model for software early error prediction (SWEEP). In Proceeding of 13th Minnow Brook Workshop on Software Reliability, Blue Mountain Lake, NY, USA, 24–27 July 1990; pp. 45–57.
48. Al-Jamimi, H.A. Toward comprehensible software defect prediction models using fuzzy logic. In Proceedings of the 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 26–28 August 2016; pp. 127–130.
49. Yadav, H.B.; Yadav, D.K. A fuzzy logic based approach for phase-wise software defects prediction using software metrics. *Inf. Softw. Technol.* **2015**, *63*, 44–57.
50. Adak, M.F. Software defect detection by using data mining based fuzzy logic. In Proceedings of the 2018 Sixth International Conference on Digital Information, Networking, and Wireless Communications (DINWC), Beirut, Lebanon, 25–27 April 2018; pp. 65–69.
51. Borgwardt, S.; Distel, F.; Peñaloza, R. The limits of decidability in fuzzy description logics with general concept inclusions. *Artif. Intell.* **2015**, *218*, 23–55.
52. Ma, Y.; Qin, K.; Zhu, S. Discrimination Analysis for Predicting Defect-Prone Software Modules. *J. Appl. Math.* **2014**, *2014*, 675368.
53. Jing, X.-Y.; Wu, F.; Dong, X.; Xu, B. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Trans. Softw. Eng.* **2016**, *43*, 321–339.
54. Naseem, R.; Khan, B.; Ahmad, A.; Almogren, A.; Jabeen, S.; Hayat, B.; Shah, M.A. Investigating tree family machine learning techniques for a predictive system to unveil software defects. *Complexity* **2020**, *2020*, 6688075.
55. Abdulshaheed, M.; Hammad, M.; Alqaddoumi, A.; Obeidat, Q. Mining historical software testing outcomes to predict future results. *Compusoft* **2019**, *8*, 3525–3529.
56. Tantithamthavorn, C.; McIntosh, S.; Hassan, A.E.; Matsumoto, K. The impact of automated parameter optimization on defect prediction models. *IEEE Trans. Softw. Eng.* **2018**, *45*, 683–711.
57. Al Qasem, O.; Akour, M.; Alenezi, M. The influence of deep learning algorithms factors in software fault prediction. *IEEE Access* **2020**, *8*, 63945–63960. [CrossRef]
58. Shen, Z.; Chen, S. A survey of automatic software vulnerability detection, program repair, and defect prediction techniques. *Secur. Commun. Netw.* **2020**, *2020*, 8858010. [CrossRef]
59. Liang, H.; Yu, Y.; Jiang, L.; Xie, Z. Seml: A semantic LSTM model for software defect prediction. *IEEE Access* **2019**, *7*, 83812–83824. [CrossRef]
60. Farid, A.B.; Fathy, E.M.; Eldin, A.S.; Abd-Elmegid, L.A. Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM). *PeerJ Comput. Sci.* **2021**, *7*, e739. [CrossRef]
61. Uddin, M.N.; Li, B.; Ali, Z.; Kefalas, P.; Khan, I.; Zada, I. Software defect prediction employing BiLSTM and BERT-based semantic feature. *Soft Comput.* **2022**, *26*, 7877–7891. [CrossRef]
62. Li, Z.; Zhang, H.; Jing, X.-Y.; Xie, J.; Guo, M.; Ren, J. Dssdpp: Data selection and sampling based domain programming predictor for cross-project defect prediction. *IEEE Trans. Softw. Eng.* **2022**, *49*, 1941–1963. [CrossRef]
63. Bennin, K.E.; Keung, J.W.; Monden, A. On the relative value of data resampling approaches for software defect prediction. *Empir. Softw. Eng.* **2019**, *24*, 602–636. [CrossRef]
64. Qiao, L.; Li, X.; Umer, Q.; Guo, P. Deep learning based software defect prediction. *Neurocomputing* **2020**, *385*, 100–110. [CrossRef]
65. Usman-Hamza, F.E.; Balogun, A.O.; Nasiru, S.K.; Capretz, L.F.; Mojeed, H.A.; Salihu, S.A.; Akintola, A.G.; Mabayoje, M.A.; Awotunde, J.B. Empirical analysis of tree-based classification models for customer churn prediction. *Sci. Afr.* **2024**, *23*, e02054. [CrossRef]
66. Ahmadlou, M.; Karimi, M.; Sammen, S.S.; Alsafadi, K. Three novel cost-sensitive machine learning models for urban growth modelling. *Geocarto Int.* **2024**, *39*, 2353252.
67. Van Phong, T.; Ly, H.-B.; Trinh, P.T.; Prakash, I.; Btjvjoes, P. Landslide susceptibility mapping using Forest by Penalizing Attributes (FPA) algorithm based machine learning approach. *Vietnam J. Earth Sci.* **2020**, *42*, 237–246.
68. Gama, J. Functional trees. *Mach. Learn.* **2004**, *55*, 219–250.
69. Mosavi, A.; Shirzadi, A.; Choubin, B.; Taromideh, F.; Hosseini, F.S.; Borji, M.; Shahabi, H.; Salvati, A.; Dineva, A.A. Towards an ensemble machine learning model of random subspace based functional tree classifier for snow avalanche susceptibility mapping. *IEEE Access* **2020**, *8*, 145968–145983.
70. Zhao, C.; Peng, R.; Wu, D. Bagging and boosting fine-tuning for ensemble learning. *IEEE Trans. Artif. Intell.* **2023**, *5*, 1728–1742.
71. Archana, K.; Komarasamy, G. A novel deep learning-based brain tumor detection using the Bagging ensemble with K-nearest neighbor. *J. Intell. Syst.* **2023**, *32*, 20220206.
72. Wu, Y.; Liu, L.; Xie, Z.; Chow, K.-H.; Wei, W. Boosting ensemble accuracy by revisiting ensemble diversity metrics. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 16469–16477.
73. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357.

74. Fernández, A.; Garcia, S.; Herrera, F.; Chawla, N.V. SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *J. Artif. Intell. Res.* **2018**, *61*, 863–905.
75. Balogun, A.O.; Lafenwa-Balogun, F.B.; Mojeed, H.A.; Adeyemo, V.E.; Akande, O.N.; Akintola, A.G.; Bajeh, A.O.; Usman-Hamza, F.E. SMOTE-based homogeneous ensemble methods for software defect prediction. In Proceedings of the Computational Science and Its Applications–ICCSA 2020: 20th International Conference, Cagliari, Italy, 1–4 July 2020; Proceedings, Part VI 20; Springer: Berlin/Heidelberg, Germany, 2020; pp. 615–631.
76. Shepperd, M.; Song, Q.; Sun, Z.; Mair, C. Data quality: Some comments on the nasa software defect datasets. *IEEE Trans. Softw. Eng.* **2013**, *39*, 1208–1215.
77. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA data mining software: An update. *ACM SIGKDD Explor. Newsl.* **2009**, *11*, 10–18.
78. Davari, A.; Islam, S.; Seehaus, T.; Hartmann, A.; Braun, M.; Maier, A.; Christlein, V. On Mathews correlation coefficient and improved distance map loss for automatic glacier calving front segmentation in SAR imagery. *IEEE Trans. Geosci. Remote Sens.* **2021**, *60*, 1–12.
79. Akintola, A.G.; Balogun, A.O.; Capretz, L.F.; Mojeed, H.A.; Basri, S.; Salihu, S.A.; Usman-Hamza, F.E.; Sadiku, P.O.; Balogun, G.B.; Alanamu, Z.O. Empirical analysis of forest penalizing attribute and its enhanced variations for android malware detection. *Appl. Sci.* **2022**, *12*, 4664. [CrossRef]
80. Alsaeedi, A.; Khan, M.Z. Software defect prediction using supervised machine learning and ensemble techniques: A comparative study. *J. Softw. Eng. Appl.* **2019**, *12*, 85–100.
81. Iqbal, A.; Aftab, S.; Ali, U.; Nawaz, Z.; Sana, L.; Ahmad, M.; Husen, A. Performance analysis of machine learning techniques on software defect prediction using NASA datasets. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 300–308.
82. Babatunde, A.N.; Ogundokun, R.O.; Adeoye, L.B.; Misra, S. Software Defect Prediction Using Dagging Meta-Learner-Based Classifiers. *Mathematics* **2023**, *11*, 2714. [CrossRef]
83. El-Shorbagy, S.A.; El-Gammal, W.M.; Abdelmoez, W.M. Using SMOTE and heterogeneous stacking in ensemble learning for software defect prediction. In Proceedings of the 7th International Conference on Software and Information Engineering, Cairo, Egypt, 2–4 May 2018; pp. 44–47.
84. Li, R.; Zhou, L.; Zhang, S.; Liu, H.; Huang, X.; Sun, Z. Software defect prediction based on ensemble learning. In Proceedings of the 2019 2nd International Conference on Data Science and Information Technology, Seoul, Republic of Korea, 19–21 July 2019; pp. 1–6.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.