Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control
Journal homepage: http://kinetik.umm.ac.id
ISSN: 2503-2267
Vol. 10, No. 2, May, Pp. 229-240

Integrating adaptive sampling with ensembles model for software defect prediction

Muhammad Yusuf[1], Arinal Haq[1], Siti Rochimah[1]
Department of Informatics Engineering, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia[1]

Article Info
Keywords:
Software Defect Prediction, ADASYN, Class Imbalance, Ensemble Learning, Stacking

Article history:
Received: December 12, 2024
Accepted: March 10, 2025
Published: May 31, 2025

Cite:
M. Yusuf, A. Haq, and S. Rochimah, "Integrating Adaptive Sampling with Ensembles Model for Software Defect Prediction", *KINETIK*, vol. 10, no. 2, May 2025.
https://doi.org/10.22219/kinetik.v10i2.2191

*Corresponding author.
Siti Rochimah
E-mail address:
siti@its.ac.id

Abstract
Handling class imbalance is a challenge in software defect prediction. Imbalanced datasets can cause bias in machine learning models, hindering their ability to detect defects. This paper proposes an integration of Adaptive Synthetic Sampling (ADASYN) and ensemble learning methods to improve prediction accuracy. ADASYN enhances the handling of imbalanced data by generating synthetic samples for hard-to-classify instances. At the same time, the ensemble stacking technique leverages the strengths of multiple models to reduce bias and variance. The machine learning models used in this study are K-Nearest Neighbors (KNN), Decision Tree (DT), and Random Forest (RF). The results demonstrate that ADASYN, combined with ensemble stacking, outperforms the traditional SMOTE technique in most cases. For instance, in the Ant-1.7 dataset, ADASYN achieved a stacking accuracy of 90.60% compared to 89.32% with SMOTE. Similarly, in the Camel-1.6 dataset, ADASYN achieved 91.56%, slightly exceeding SMOTE's 91.32%. However, SMOTE performed better in simpler models like Decision Tree for certain datasets, highlighting the importance of choosing the appropriate resampling method. Across all datasets, ensemble stacking consistently provided the highest accuracy, benefiting from ADASYN's adaptive resampling strategy. These results underscore the importance of combining advanced sampling methods with ensemble learning techniques to address class imbalance effectively. This approach improves prediction accuracy and provides a practical framework for reliable software defect prediction in real-world scenarios. Future work will explore hybrid techniques and broader evaluations across diverse datasets and classifiers.

1. Introduction

Software engineering is a branch of engineering focused on creating a wide range of software products that are reliable, efficient, and perform their intended functions effectively. [1][2]. It covers all aspects of software development, from the initial specification phase to the maintenance stage after delivery to the user. [2]. As such, the quality of the software depends largely on the development process and the effectiveness of testing. However, mistakes can arise at any stage of the development process. [3][4]. Predicting software issues in the early stages of software development is a critical focus in the logistics software field, as many companies invest substantial resources in addressing errors caused by software faults. These software faults are commonly referred to as software defects. [5][6].

Software defect prediction plays a vital role in ensuring the quality of software systems while minimizing project costs. [7]. Predicting potential defects early in the software development lifecycle allows teams to address issues proactively, leading to more reliable software products.[8]However, one key challenge in this process is the presence of imbalanced datasets, where the number of defect-free instances significantly outweighs the number of defective ones. [2][9]. This imbalance can cause prediction models to underperform, as they tend to favor the majority class. To overcome this challenge, various techniques have been developed to improve model performance on imbalanced data [10].

The Synthetic Minority Over-sampling Technique (SMOTE) is one of the most widely used methods for handling imbalanced datasets [11]. SMOTE balances the dataset by creating synthetic instances for the minority class, enabling models to learn more effectively from the underrepresented data. By applying SMOTE, researchers and practitioners can ensure that the prediction models do not overfit to the majority class and are better able to detect software defects. This technique has proven effective across various domains and has become a standard approach in handling imbalanced data [12].

Various sampling techniques have been suggested in the literature to address imbalanced data, aiming to transform it into a balanced dataset [2][10]. Balancing the datasets enhances the accuracy of classification. The primary methods employed to address imbalanced data include undersampling, oversampling, and the generation of synthetic data [1][13]. In addition to balancing datasets, classification algorithms play a central role in software defect prediction. Popular machine learning methods such as Random Forest, Decision Tree, and Support Vector Machine (SVM) have been widely applied in defect prediction tasks [8]. These methods are known for their ability to handle complex classification problems, providing reliable predictions of software defects. Random Forest and Decision Tree, for instance, are valued for their interpretability and robustness, while SVM is often favored for its ability to handle high-dimensional data [11]. These techniques form the foundation of predictive models used in the field [14].

Despite the effectiveness of SMOTE in handling class imbalance, it has several limitations. One of the main drawbacks is that it generates synthetic samples without considering the difficulty of classification for each instance [15]. This may lead to over-generalization, where less informative samples are created, reducing the overall predictive power of the model. Additionally, traditional oversampling techniques do not always improve model performance in highly imbalanced datasets, as they may still lead to models that favor the majority class. Furthermore, previous studies that apply ensemble learning with SMOTE-based methods have not fully explored the impact of alternative oversampling techniques, such as ADASYN, in improving software defect prediction [10].

These imbalances highlight the need for adaptive resampling techniques, such as ADASYN, to address the issue of data imbalance and improve model accuracy in detecting software defects that are rare but carry significant risks. ADASYN (Adaptive Synthetic Sampling) has emerged as a more adaptive technique for handling imbalanced data [14] Unlike traditional SMOTE, ADASYN generates synthetic samples by focusing more on harder-to-classify instances, thereby enhancing model performance in more challenging scenarios. This dynamically adjusts the sampling process based on the distribution of the data, ensuring that models are better trained to detect rare and difficult defects. Additionally, ensembles, such as stacking, offer further potential for improving model performance. By combining predictions from multiple models through a meta-model, stacking enhances the final prediction, producing a more accurate and reliable output than any single model alone [16].

The implementation of ADASYN remains underexplored in the context of software defect prediction, particularly when combined with ensemble learning techniques. Previous research has primarily focused on SMOTE, but the integration of ADASYN with ensemble methods such as stacking has not been thoroughly investigated in this domain. This paper proposes to integrate ADASYN for oversampling in software defect prediction, while also incorporating an ensemble learning approach. Specifically, the use of ADASYN is expected to better handle imbalanced data by focusing more on difficult-to-classify samples. At the same time, the ensemble of classifiers ensures that the model leverages the strengths of multiple learning algorithms to reduce bias and variance. The primary contributions of this study are:

- The introduction of the use of Integration of ADASYN with an ensemble stacking approach as a novel technique, which effectively focuses on hard-to-classify instances, thereby improving the handling of imbalanced datasets in software defect prediction.
- An analysis of the impact of combining ADASYN techniques with ensemble methods on software defect prediction.
- A comparative analysis of the ADASYN-ensemble approach against traditional SMOTE-based methods, demonstrating improvements in handling class imbalance and prediction accuracy.

2. Related Work

This paper highlight that addressing class imbalance in software defect prediction is a complex challenge requiring advanced methods to enhance the model performance [17]. Although traditional resampling techniques such as SMOTE are widely used, they can introduce noise that negatively impacts classification performance. ADASYN emerges as an alternative by focusing on hard-to-classify instances, thereby reducing noise risk and improving the model's ability to handle imbalanced datasets [18].

Rizky et al. [19] proposed a novel pre-processing method called RO-SMOTE, which improves on the traditional SMOTE technique by integrating an outlier detection algorithm to remove outliers introduced during the oversampling process. This approach enhances data quality by reducing noise and making the dataset more balanced for effective software defect prediction. Comprehensive evaluations across various classifiers and datasets validate the superiority of the RO-SMOTE method, demonstrating its effectiveness in improving classification performance in terms of accuracy, precision, recall, F1-score, and AUC compared to traditional SMOTE.

Mulia et al. [20] contributions can be summarized as follows: First, they replaced the traditional SMOTE technique to better address class imbalance in software defect prediction, with a particular focus on handling more difficult-to-classify instances. This approach leads to more effective management of imbalanced datasets. Lastly, the proposed method demonstrates improved accuracy when compared to SMOTE-based approaches, as confirmed by extensive evaluations across different classifiers and datasets, validating its effectiveness in handling imbalances and improving model performance.

Abdul et al. [21] introduced a novel method to tackle the dual challenges of class imbalance and overlap in software defect prediction. Their approach combines overlap-driven undersampling with a balanced random forest classifier, resulting in a significant improvement in detecting defective software modules. By applying the balanced random forest, the authors demonstrated superior performance across 15 datasets compared to nine existing models, particularly excelling in recall, G-mean, F-measure, and AUC metrics.

Somya Goyal et al [22] propose a heterogeneous stacked ensemble classifier for software defect prediction (SDP). Their approach focuses on addressing the challenge of class imbalance in SDP datasets through a two-level stacked ensemble method. The model integrates five base classifiers—Artificial Neural Networks (ANN), Decision Trees (DT), Naïve Bayes (NB), K-Nearest Neighbor (KNN), and Support Vector Machine (SVM)—to improve prediction accuracy and generalizability. They demonstrate the model's effectiveness using datasets from the PROMISE corpus, achieving superior performance compared to traditional classifiers across metrics such as ROC, AUC, and accuracy. The research statistically validates the proposed model's performance advantage, presenting it as a robust solution to the class imbalance problem in software defect predictions.

Shuo Feng et al [23] investigate the stability of SMOTE-based oversampling techniques for software defect prediction (SDP). Their contribution lies in identifying and addressing the instability caused by randomness in SMOTE-based methods, which can lead to misleading performance results when comparing SDP techniques. They propose a series of stable SMOTE-based oversampling techniques that reduce randomness at each step, thereby improving performance stability. These techniques were validated both mathematically and empirically across 26 PROMISE datasets and four classifiers, showing significantly lower variance and higher stability in metrics such as AUC, Balance, and MCC. The authors suggest that these stable SMOTE-based methods should replace traditional SMOTE-based approaches to ensure more reliable and convincing defect prediction outcomes.

From the various approaches discussed, it is evident that no single solution can comprehensively address all challenges related to class imbalance in software defect prediction. However, approaches that combine resampling techniques with more advanced machine learning models, such as the one introduced by Abdul et al., appear to provide a more robust framework for improving prediction accuracy compared to the conventional methods like SMOTE. In this study, this paper contributes by integrating ADASYN with ensemble learning, leveraging ADASYN’s ability to generate synthetic data for hard-to-classify instances while reducing noise, combined with the robustness of ensemble learning methods to improve generalization and classification performance. This integration enhances the model's ability to handle imbalanced datasets more effectively, ensuring better defect prediction accuracy while mitigating the drawbacks of standalone resampling techniques.

2. Research Method

This section presents the proposed methodology, which combines ADASYN and efficient algorithms to address the outlier issue in imbalanced data. The workflow of the proposed method in this paper is shown in Figure 1. Subsection 3.1 discusses ADASYN [24] which forms the foundation of the proposed method by focusing on generating synthetic samples in regions with difficult-to-classify instances. Subsection 3.2 covers the utilization of ensemble learning [25] to enhance the prediction of software defects, optimizing the model's performance through the combination of multiple classifiers. The detailed of stacking ensemble model used in this study is shown in Figure 2.

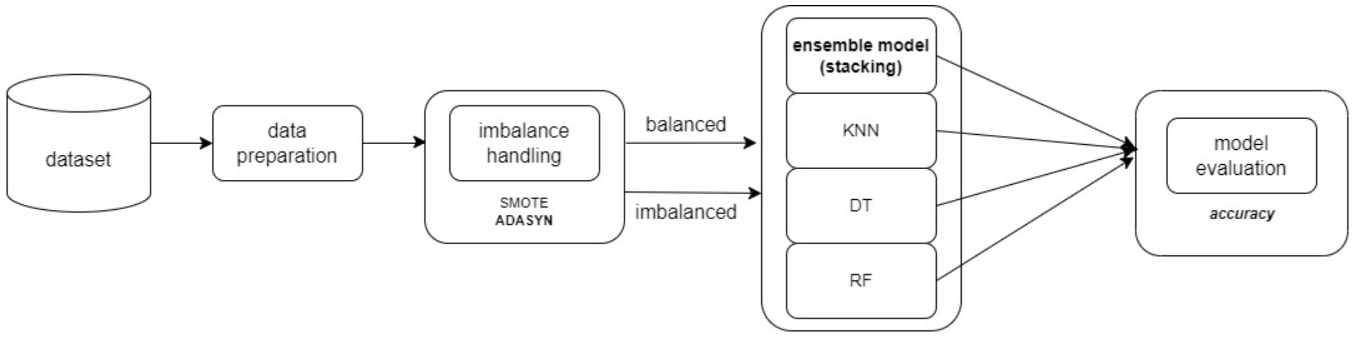

Figure 1. The Workflow Diagram of the Purposed Method

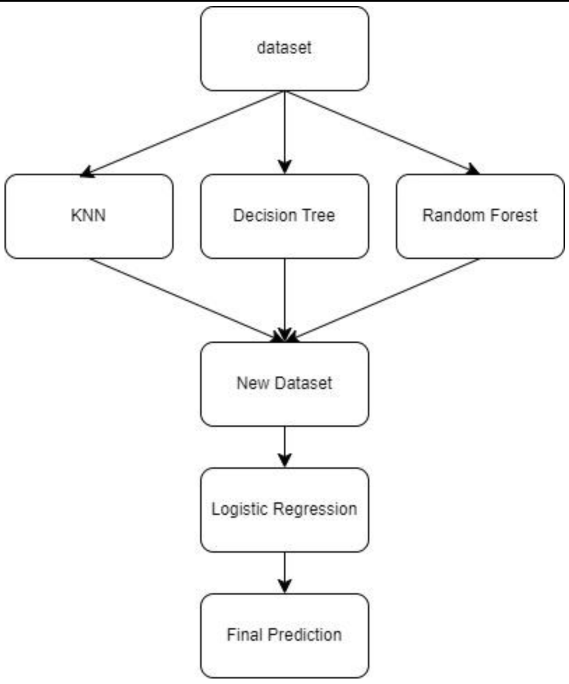

Figure 2. Stacking Ensemble Model

2.1 Imbalance Handling

Handling imbalance is a critical step in data preprocessing, particularly when one class heavily outweighs the other, resulting in biased model performance. ADASYN (Adaptive Synthetic Sampling) is a widely used oversampling technique that tackles this problem by adaptively generating synthetic samples for the minority class [26]. Unlike traditional methods like SMOTE, which creates synthetic samples uniformly, ADASYN focuses more on difficult-to-classify instances.

The ADASYN process is an advanced oversampling technique that specifically targets class imbalance by adaptively generating synthetic samples for the minority class. Unlike uniform methods like SMOTE, ADASYN prioritizes instances that are harder to classify by analyzing the local distribution of data. This makes it particularly effective in improving a model's ability to handle challenging regions of the dataset while maintaining generalization and avoiding overfitting. By dynamically adjusting the oversampling strategy based on the difficulty of classification, ADASYN ensures a more targeted approach to balancing the dataset. [27].

The process begins with the calculation of a difficulty score for each minority class instance. This score is derived by analyzing the $k\ nearest\ neighbors$ Of the instance and determining the proportion of neighbors belonging to the majority class. Instances surrounded by a higher density of majority class neighbors are considered more challenging to classify and are assigned higher difficulty scores. These scores highlight the regions in the dataset where synthetic samples are most needed to improve model learning.

Next, the difficulty scores are normalized to generate weighting factors for each minority class instance. These weighting factors determine the number of synthetic samples generated for each instance. This normalization ensures that the most difficult instances contribute proportionally more to the generation of synthetic data, allowing the model to focus on learning from these challenging examples.

The synthetic sample generation step involves interpolation between a minority instance and one of its $k\ nearest$ Minority neighbors. A synthetic sample is created along the line segment connecting the two instances, using the formula. $x\ new = x_i + \lambda\ (x_k - x_i)$ Where λ is a random value between 0 and 1, this method ensures that the synthetic samples are evenly distributed in the feature space, particularly in regions with higher classification difficulty, without replicating existing instances. Finally, the total number of synthetic samples to be generated is predefined, and ADASYN distributes these samples according to the calculated weighting factors. This approach not only balances the class distribution but also places greater emphasis on the more difficult regions of the dataset. By incorporating these synthetic samples, ADASYN enables models to learn more effectively from the minority class, resulting in improved classification performance and reduced bias toward the majority class.

Using this information, ADASYN generates more synthetic samples for minority instances located in regions with higher classification difficulty, thus improving the model's ability to learn from challenging examples [28]. This method not only balances the dataset but also enhances the model’s performance on the minority class without causing overfitting. By incorporating ADASYN, models can achieve more robust and fair predictions, especially in scenarios where class imbalance significantly affects the learning process.

2.2 Ensemble Learning

Ensemble Learning is a machine learning approach that merges multiple base models, also known as weak learners, to develop a more reliable and precise predictive model. This method leverages the advantages of various models while mitigating their limitations.[29]. Figure 3 illustrates the process of ensemble learning, showing how different classifiers (Classifier 1, Classifier 2, ..., Classifier K) are used in combination to improve prediction accuracy. Voting is one of the most straightforward and widely used ensemble techniques, particularly effective in classification tasks. This method combines predictions from multiple base models to determine the outcome. There are two primary types of voting: hard voting and soft voting. In hard voting, each base model selects a specific class, and the class that receives the majority of votes is chosen as the final prediction. For example, if three models predict Class A, Class B, and Class A, the final result will be Class A since it has the highest number of votes. This technique is simple and works best when the base models are diverse and individually perform better than random guessing. On the other hand, soft voting calculates the predicted probabilities of each class from all models and selects the class with the highest average probability as the final decision. [30]. This method is more flexible and often yields better results, especially when the base models produce well-calibrated probabilities.

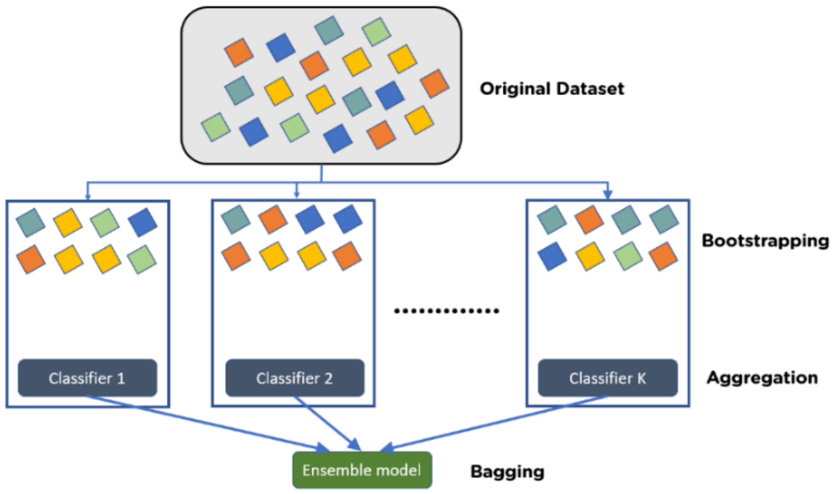

Figure 3. Overview of Ensemble Learning

Stacking is a paradigm that explores a space of different models for the same problem. It involves building multiple learners and using them to create intermediate predictions. A new model is added to the intermediate predictions, learning the same target. The final model, stacked on top of the others, improves overall performance and often outperforms any individual intermediate model. However, stacking does not guarantee success, as is often the case with machine learning technologies [31].

Ensemble learning involves several steps to build a robust and accurate predictive model [32]:

1. Select base models
   The first step involves selecting a diverse set of base models. These models can include algorithms such as Decision Trees, Logistic Regression, and Support Vector Machines (SVM). Diversity is a critical factor in ensemble learning because it ensures that each model captures unique patterns or relationships in the data. For example, Decision Trees excel at handling non-linear relationships, while Logistic Regression is well-suited for linear patterns, and SVM works effectively with high-dimensional data. By combining models with different strengths, the ensemble can compensate for the weaknesses of individual models, resulting in improved overall performance.
2. Train Base Models
   Once the models are selected, the next step is to train each model independently. This training can be done using the entire dataset or subsets of the dataset, depending on the specific ensemble technique being used. For example:
   Using the Same Dataset: Each model is trained on the full dataset, allowing them to develop their unique perspective on the same information.
   Using Subsets of the Dataset: In methods like bagging (e.g., Random Forest), subsets of the dataset are created by sampling with replacement, and each model is trained on a different subset. This introduces variation

among the models, further enhancing diversity. During training, hyperparameters for each model may also be tuned to optimize their individual performance.

3. Combine Predictions
   After training, the predictions from the base models are combined to produce a single output. This step can be taken using one of the following methods:
   Hard Voting: In this approach, each model predicts a specific class, and the final prediction is determined by majority voting. For example, if three models predict Class A, Class B, and Class A, the final prediction will be Class A because it has the most votes.
   Soft Voting: Instead of voting directly for classes, this approach averages the predicted probabilities of each class from all models. The final prediction is based on the class with the highest average probability. Soft voting is often preferred because it considers the confidence levels of the models, leading to more nuanced and accurate predictions. The method chosen depends on the nature of the problem and the calibration of the base models. Make final predictions based on the voting mechanism, the ensemble model outputs the final prediction.

Stacking is an ensemble learning method that improves predictive performance by integrating multiple base models, also known as level-0 models. This technique leverages the strengths of various algorithms by training a meta-model (level-1 model) to learn from the predictions generated by the base models [33]. The base models are generally trained on the same dataset, and their prediction whether class probabilities or continuous output serve as input features for the meta-model. The meta-model then learns to assign optimal weights to these predictions, helping to minimize biases and variances present in individual models. Stacking is commonly applied in both regression and classification tasks, providing better generalization and often surpassing the performance of individual algorithms or simpler ensemble techniques such as bagging and boosting [21].

Stacking, also known as stacked generalization, is an ensemble learning approach aimed at enhancing predictive accuracy and generalization by integrating multiple base models (level-0 models) with a meta-model (level-1 model). Unlike bagging and boosting, stacking trains diverse algorithms separately on the same dataset and utilizes their predictions as input features for the meta-model. Typically, the meta-model is a simpler algorithm, such as linear regression or logistic regression, which learns to optimally combine the outputs of the base models. This technique effectively harnesses the complementary strengths of the base models while reducing their individual weaknesses [29].

The stacking process generally consists of two phases. In the first phase, multiple base models are independently trained on the training dataset. These models can either be homogeneous, such as multiple decision trees, or heterogeneous, including decision trees, support vector machines, and neural networks. In the second phase, the predictions generated by these base models, typically in the form of probabilities or continuous outputs, serve as input data for training the meta-model. The meta-model then learns to optimally weight and combine these predictions, ultimately improving overall model performance [34]. To mitigate overfitting, stacking typically employs cross-validation during training. In this process, the base models are trained on different data folds, and their predictions for the unseen folds are used as input for the meta-model. This approach ensures that the meta-model learns from diverse training subsets, improving its generalization capability.

Stacking is a highly adaptable and effective technique for both classification and regression tasks. It is especially beneficial in situations where base models demonstrate varying performance patterns, as the meta-model can leverage these differences to enhance overall predictive accuracy [35]. By integrating models with complementary strengths, stacking helps reduce bias and variance, often outperforming individual models and simpler ensemble methods. This approach is widely adopted in machine learning competitions and real-world applications that demand high accuracy and robustness [36].

3. Results and Discussion

In this section, this paper discusses the evaluation of ADASYN, which includes assessing its performance with three different classification algorithms: KNN, DT, and RF. This paper conducted this evaluation on datasets from 11 cross-projects for software defect prediction, each with varying imbalance ratios. The evaluation employed multiple metrics, such as balanced accuracy.

3.1 Datasets

The Table 1 below presents a summary of software defect datasets obtained from the PROMISE repository. It includes information about four different software projects: Ant-1.7, Camel-1.6, Velocity-1.6, and Xalan-2.4. Each dataset details the total number of instances (#Instances), the number of attributes (#Attributes), the number of defective classes (#Defects), and the percentage of defective classes (%Defect). These datasets are widely used for research in software defect prediction and provide insights into defect distribution across various software versions. The defect percentages range from 13.36% in Xalan-2.4 to 34.06% in Velocity-1.6, reflecting varying levels of software reliability.

Table 1. Summary of Software Defect Datasets

Datasets	Project	#Instance	#Attributes	#Defects	%Defects
Promise	Ant-1.7	745	21	166	22.28%
	Camel-1.6	965	21	188	19.48%
	Log4j-1.1	109	21	37	33.94%
	Velocity-1.6	229	21	78	34.06%
	Xalan-2.4	1363	21	182	13.36%

3.2 Result of Modelling Data with Imbalance

Modelling with imbalanced data refers to the scenario where the dataset used for training a machine learning model contains a significant disparity in the distribution of its target classes. For example, one class may significantly outnumber the other(s), which can bias the model toward the majority class and result in poor predictive performance for the minority class. In such cases, metrics like accuracy can be misleading as a high accuracy may simply reflect the model’s ability to predict the majority class correctly, while underperforming for the minority class.

Table 2. Performance of Machine Learning Models on Imbalanced Software Defect Data

Model	Ant-1.7	Camel-1.6	Log4j-1.1	Velocity-1.6	Xalan-2.4
KNN	82.55%	79.79%	80.82%	73.91%	83.45%
DT	71.81%	76.17%	81.42%	63.04%	80.00%
RF	84.56%	83.42%	81.82%	71.74%	84.14%
Stacking	85.23%	82.90%	86.36%	80.43%	86.14%

From the results in Table 2, it is evident that different machine learning models handle imbalanced datasets with varying levels of effectiveness. Random Forest generally performed well across datasets, showing higher accuracies. The Ensemble Model, which combines predictions from multiple models, consistently provided robust results, outperforming or matching the best individual models in most cases. The imbalanced nature of the datasets might impact models like Decision Trees, which showed lower accuracy compared to other models. KNN performed relatively well but was slightly outperformed by more sophisticated models such as Random Forest and Ensemble approaches. In practice, while accuracy is a useful metric, for imbalanced data, it is critical to consider additional metrics like precision, recall, and F1-score, especially for applications where the correct prediction of the minority class is crucial. Techniques such as oversampling the minority class, undersampling the majority class, or using class-weighted models can also improve performance when dealing with imbalanced datasets.

Decision Tree generally performed poorly compared to other models, likely due to its tendency to overfit to the majority class. Random Forest demonstrated strong performance by leveraging ensemble learning to mitigate overfitting and enhance generalization. Ensemble Models, which combine the strengths of multiple models, consistently outperformed or matched the best individual models, establishing them as the most robust option for these datasets. Meanwhile, KNN delivered moderate performance, making it more suitable for balanced or smaller datasets.

3.3 Performance Analysis

Table 3 below explains the number of samples from the results of the imbalance handling technique using SMOTE and ADASYN. In SMOTE, the amount of data in each class really shows a comparison of data at a ratio of 50:50. While ADASYN doesn't always generate a perfectly balanced dataset (e.g., 50:50 ratio) because it adaptively creates synthetic samples based on the density distribution of the minority class. In practice, achieving a perfect 50:50 balance is not always necessary or desirable. The key objective is to enhance the model's ability to learn from the minority class, which ADASYN accomplishes by focusing on the most challenging sample.

Table 3. Number of Sample after Imbalance Handling

Datasets	Class	SMOTE	ADASYN
Ant 1.7	Not-Defect	579	579
	Defect	579	587
Camel 1.6	Not-Defect	777	777
	Defect	777	822
Log4j 1.1	Not-Defect	72	72
	Defect	72	69
Velocity 1.6	Not-Defect	151	151
	Defect	151	145
Xalan 2.4	Not-Defect	613	613
	Defect	613	609

Table 4 below compares the performance of two resampling techniques, SMOTE and ADASYN, across multiple datasets and machine learning models. The comparison includes models such as KNN, DT, RF, and an Ensemble Model, for each resampling method. SMOTE focuses on generating synthetic samples by interpolating between minority class instances, ensuring a balanced distribution in the dataset. In contrast, ADASYN adapts the generation of synthetic samples based on the difficulty of classifying instances, giving more emphasis to harder-to-classify cases. The results highlight the differences in model performance and stacking accuracy, showcasing ADASYN's potential advantages in certain datasets, particularly for ensemble stacking accuracy.

Table 4. Comparison of SMOTE and ADASYN in Software Defect Prediction

Datasets	Model	SMOTE				ADASYN			
		Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy
Ant 1.7	KNN	88,50	89,50	87,25	87.50	82,10	83.40	83.60	83.33
	DT	81,75	84,40	82,27	82.33	75,10	76,90	76,40	76.50
	RF	88,80	89,80	89,50	89.22	89,50	89,60	88,80	89.74
	Stacking	89,30	89,80	89,50	**89.32**	90,50	90,68	90,70	**90.60**
Camel 1.6	KNN	79,20	79,10	78,40	78.14	81,45	81,50	82,10	81.56
	DT	79,40	79,70	79,90	79.74	81,20	81,50	81,30	81.25
	RF	91,50	91,20	91,10	91.00	90,50	91,50	90,80	90.94
	Stacking	91,50	91,40	91,30	**91.32**	90,50	91,70	91,50	**91.56**
Log4j 1.1	KNN	85,40	87,80	86,40	86.21	74,90	76,10	75,80	75.86
	DT	90,45	89,70	89,60	89.66	61,70	63,60	62,10	62.07
	RF	86,10	86,15	86,30	86.21	74,60	74,70	75,10	75.86
	Stacking	89,50	89,70	89,75	**89.66**	82,50	82,60	82,90	**82.76**
Velocity 1.6	KNN	74,40	75,10	75,40	75.41	71,70	71,60	71,80	71.67
	DT	72,10	73,10	72,50	72.13	66,50	66,70	66,70	66.67
	RF	77,00	77,20	77,15	77.05	71,20	70,20	70,59	70.00
	Stacking	73,50	74.10	73,80	**73.77**	75,00	75,00	75,00	**75.00**
Xalan 2.4	KNN	85,10	84,70	84,00	84.15	82,60	83,10	84,10	83.67
	DT	81,20	82,50	82,15	82.11	84,20	84,90	85,70	84.90
	RF	87,50	87,40	87,60	87.40	88,50	88,50	88,70	88.57
	Stacking	90,10	91,10	90,70	**90.61**	90,62	90,70	90,65	**90.61**

The results from the table highlight the differences between SMOTE and ADASYN in terms of their impact on various machine learning models and ensemble stacking performance. SMOTE generally provided slightly higher or comparable accuracy for Decision Tree models compared to ADASYN, suggesting that SMOTE’s interpolation-based method is better suited for simpler models that rely on linear or rule-based decision boundaries. Conversely, ADASYN demonstrated competitive performance in more complex models like Random Forest and stacking ensembles, where its adaptive sample generation could better capture the intricacies of imbalanced data. Across most datasets, ensemble stacking with ADASYN slightly outperformed SMOTE, as seen in datasets like Ant-1.7 (90.60% for ADASYN vs. 89.32% for SMOTE) and Camel-1.6 (91.56% for ADASYN vs. 91.32% for SMOTE). This indicates that ADASYN's adaptive focus on harder-to-classify instances can enhance the generalization capabilities of ensemble stacking, making it a stronger option for combining multiple models.

For specific datasets, SMOTE performed better for Log4j-1.1, with a stacking accuracy of 89.66%, compared to 82.76% for ADASYN, suggesting that SMOTE’s approach of uniformly generating synthetic samples might be more effective for smaller or simpler datasets. For Xalan-2.4, both methods achieved the same stacking accuracy of 90.61%, indicating that for some datasets, the choice of resampling technique may not significantly affect ensemble performance. In Velocity-1.6, ADASYN achieved slightly better stacking accuracy (75.00% vs. 73.77%), showing its adaptability in generating synthetic samples tailored to complex decision boundaries. Regarding model-specific observations, Random Forest consistently achieved high accuracy for both SMOTE and ADASYN, reflecting its robustness to data imbalance. However, Decision Tree models were more sensitive to the resampling method, with SMOTE often outperforming ADASYN due to their reliance on simpler decision boundaries. These results underline the importance of choosing the appropriate resampling method based on the dataset and the complexity of the model.

3.4 Results Analysis

The analysis focuses on the performance comparison between SMOTE and ADASYN as resampling techniques for addressing class imbalance in software defect prediction datasets. Table 3 highlights the results across various machine learning models, including KNN, Decision Tree (DT), Random Forest (RF), and an Ensemble Stacking approach. Each technique's impact on model performance is evaluated using datasets with differing characteristics, offering insights into their relative strengths and limitations.

From the results, it is evident that SMOTE and ADASYN exhibit varying levels of effectiveness depending on the dataset and the model used. SMOTE generally performed better or was comparable to ADASYN for simpler models like decision trees, where its uniform interpolation method aligns well with the linear or rule-based decision boundaries of these models. For example, in the Log4j-1.1 dataset, SMOTE achieved a stacking accuracy of 89.66%, outperforming ADASYN's 82.76%. Similarly, for Velocity-1.6, SMOTE demonstrated higher performance in DT (72.13% vs. 66.67%).

Conversely, ADASYN showcased its adaptive strength in more complex models such as Random Forest and Stacking Ensembles. ADASYN’s emphasis on generating synthetic samples for harder-to-classify instances helped it capture the nuanced relationships in the data, leading to competitive or superior performance in several cases. For example, in the Ant-1.7 dataset, ADASYN outperformed SMOTE in stacking accuracy (90.60% vs. 89.32%) and delivered comparable results in Camel-1.6 (91.56% for ADASYN vs. 91.32% for SMOTE). Notably, ADASYN achieved slightly better results in Velocity-1.6 for stacking (75.00% vs. 73.77%), reflecting its adaptability to datasets with complex decision boundaries. Random Forest consistently delivered high accuracy for both resampling methods, emphasizing its robustness to data imbalance. For instance, RF achieved 89.74% accuracy with ADASYN and 89.22% with SMOTE in Ant-1.7, showcasing the marginal impact of the resampling method on RF's performance. However, Decision Tree models were more sensitive to the choice of resampling technique, with SMOTE often outperforming ADASYN due to its simpler and uniform sample generation approach.

Overall, the results indicate that the dataset characteristics and the complexity of the model should guide the choice between SMOTE and ADASYN. While SMOTE is suitable for simpler models and datasets with relatively uniform data distribution, ADASYN provides an advantage in capturing the complexities of imbalanced data when used with ensemble models and advanced algorithms. ADASYN generates synthetic samples adaptively based on the density distribution of minority class samples, focusing more on difficult-to-learn samples. SMOTE generates synthetic samples by interpolating between minority class samples without considering their density distribution.

Ensemble stacking particularly benefited from ADASYN’s adaptive sample generation, making it a strong choice for combining multiple models to enhance generalization. The ensemble method using stacking shows better results for both SMOTE and ADASYN. This can happen because the stacking method uses a combination of prediction results from several base models, which are then used as data for the meta model to get the results. This way, the stacking method is successful in getting the best performance. By balancing the dataset with ADASYN, the base learners in the stacking ensemble can learn more effectively from the minority class, leading to better overall performance, and stacking leverages the strengths of different models, making the final prediction more robust and less prone to errors from any single model. The combination of these methods helps in reducing overfitting, as the meta-learner can correct biases from the base learners.

4. Conclusion

This study highlights the importance of integrating imbalance handling techniques with ensemble learning methods to address the challenges posed by class imbalance in software defect prediction datasets. Our evaluation shows that ADASYN, with its adaptive focus on harder-to-classify instances, can positively influence ensemble stacking performance, especially when combined with advanced classification models such as Random Forest and stacking ensembles. Compared to SMOTE, ADASYN demonstrates competitive or superior performance in datasets with complex decision boundaries, showcasing its potential for enhancing model generalization and predictive accuracy. The incorporation of stacking as an ensemble technique in the analysis further underscores its robustness and effectiveness in leveraging the strengths of multiple base models. The results indicate that stacking combined with ADASYN consistently outperformed or matched traditional methods like SMOTE, particularly in terms of accuracy and the ability to handle imbalanced datasets. These findings reinforce the value of combining adaptive resampling methods with ensemble strategies, offering a practical approach for improving software defect prediction outcomes. Future work could explore extending the evaluation to more diverse datasets and classifiers, as well as investigating hybrid techniques that combine the strengths of both SMOTE and ADASYN for further performance improvements. By addressing these areas, this paper aims to contribute to the development of more reliable and efficient strategies for handling imbalanced data in machine learning applications.

References

[1] E. Izquierdo-Verdiguier and R. Zurita-Milla, “An evaluation of Guided Regularized Random Forest for classification and regression tasks in remote sensing,” Int. J. Appl. Earth Obs. Geoinformation, vol. 88, p. 102051, Jun. 2020. https://doi.org/10.1016/j.jag.2020.102051
[2] A. Kurani, P. Doshi, A. Vakharia, and M. Shah, “A Comprehensive Comparative Study of Artificial Neural Network (ANN) and Support Vector Machines (SVM) on Stock Forecasting,” Ann. Data Sci., vol. 10, no. 1, pp. 183–208, Feb. 2023. https://doi.org/10.1007/s40745-021-00344-x
[3] A. Antoniadis, S. Lambert-Lacroix, and J.-M. Poggi, “Random forests for global sensitivity analysis: A selective review,” Reliab. Eng. Syst. Saf., vol. 206, p. 107312, Feb. 2021. https://doi.org/10.1016/j.ress.2020.107312
[4] Y. Tang, Q. Dai, M. Yang, T. Du, and L. Chen, “Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm,” Int. J. Mach. Learn. Cybern., vol. 14, no. 6, pp. 1967–1987, Jun. 2023. https://doi.org/10.1007/s13042-022-01740-2

[5] Y. Zakariyau Bala, P. Abdul Samat, K. Yatim Sharif, and N. Manshor, “Cross-project software defect prediction through multiple learning,” Bull. Electr. Eng. Inform., vol. 13, no. 3, pp. 2027–2035, Jun. 2024. https://doi.org/10.11591/eei.v13i3.5258
[6] M. Aria, C. Cuccurullo, and A. Gnasso, “A comparison among interpretative proposals for Random Forests,” Mach. Learn. Appl., vol. 6, p. 100094, Dec. 2021. https://doi.org/10.1016/j.mlwa.2021.100094
[7] K. K. Bejjanki, J. Gyani, and N. Gugulothu, “Class Imbalance Reduction (CIR): A Novel Approach to Software Defect Prediction in the Presence of Class Imbalance,” Symmetry, vol. 12, no. 3, p. 407, Mar. 2020. https://doi.org/10.3390/sym12030407
[8] N. A. A. Khleel and K. Nehéz, “Software defect prediction using a bidirectional LSTM network combined with oversampling techniques,” Clust. Comput., vol. 27, no. 3, pp. 3615–3638, Jun. 2024. https://doi.org/10.1007/s10586-023-04170-z
[9] N. S. Thomas and S. Kaliraj, “An Improved and Optimized Random Forest Based Approach to Predict the Software Faults,” SN Comput. Sci., vol. 5, no. 5, p. 530, May 2024. https://doi.org/10.1007/s42979-024-02764-x
[10] M. N. M. Rahman, R. A. Nugroho, M. R. Faisal, F. Abadi, and R. Herteno, “Optimized multi correlation-based feature selection in software defect prediction,” TELKOMNIKA Telecommun. Comput. Electron. Control, vol. 22, no. 3, p. 598, Jun. 2024. https://doi.org/10.12928/telkomnika.v22i3.25793
[11] Y. Sun et al., “Unsupervised Domain Adaptation Based on Discriminative Subspace Learning for Cross-Project Defect Prediction,” Comput. Mater. Contin., vol. 68, no. 3, pp. 3373–3389, 2021. https://doi.org/10.32604/cmc.2021.016539
[12] T. Zhang et al., “Intelligent fault diagnosis of machines with small & imbalanced data: A state-of-the-art review and possible extensions,” ISA Trans., vol. 119, pp. 152–171, Jan. 2022. https://doi.org/10.1016/j.isatra.2021.02.042
[13] W. Huang et al., “Railway dangerous goods transportation system risk identification: Comparisons among SVM, PSO-SVM, GA-SVM and GS-SVM,” Appl. Soft Comput., vol. 109, p. 107541, Sep. 2021. https://doi.org/10.1016/j.asoc.2021.107541
[14] Y. Tang, Q. Dai, M. Yang, T. Du, and L. Chen, “Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm,” Int. J. Mach. Learn. Cybern., vol. 14, no. 6, pp. 1967–1987, Jun. 2023. https://doi.org/10.1007/s13042-022-01740-2
[15] A. Antoniadis, S. Lambert-Lacroix, and J.-M. Poggi, “Random forests for global sensitivity analysis: A selective review,” Reliab. Eng. Syst. Saf., vol. 206, p. 107312, Feb. 2021. https://doi.org/10.1016/j.ress.2020.107312
[16] M. Hammad, M. H. Alkinani, B. B. Gupta, and A. A. Abd El-Latif, “Myocardial infarction detection based on deep neural network on imbalanced data,” Multimed. Syst., vol. 28, no. 4, pp. 1373–1385, Aug. 2022. https://doi.org/10.1007/s00530-020-00728-8
[17] J. Zheng, X. Wang, D. Wei, B. Chen, and Y. Shao, “A Novel Imbalanced Ensemble Learning in Software Defect Predication,” IEEE Access, vol. 9, pp. 86855–86868, 2021. https://doi.org/10.1109/ACCESS.2021.3072682
[18] K. Vos, Z. Peng, C. Jenkins, M. R. Shahriar, P. Borghesani, and W. Wang, “Vibration-based anomaly detection using LSTM/SVM approaches,” Mech. Syst. Signal Process., vol. 169, p. 108752, Apr. 2022. https://doi.org/10.1016/j.ymssp.2021.108752
[19] M. Nevendra and P. Singh, “Empirical investigation of hyperparameter optimization for software defect count prediction,” Expert Syst. Appl., vol. 191, p. 116217, Apr. 2022. https://doi.org/10.1016/j.eswa.2021.116217
[20] X. Pu et al., “Improving Lower Limb Function and Frailty in Frail Older Patients with Acute Myocardial Infarction After Percutaneous Coronary Intervention: A Randomized Controlled Study of Neuromuscular Electrical Stimulation,” Clin. Interv. Aging, vol. Volume 19, pp. 1163–1176, Jul. 2024. https://doi.org/10.2147/CIA.S460805
[21] M. N. M. Rahman, R. A. Nugroho, M. R. Faisal, F. Abadi, and R. Herteno, “Optimized multi correlation-based feature selection in software defect prediction,” TELKOMNIKA Telecommun. Comput. Electron. Control, vol. 22, no. 3, p. 598, Jun. 2024. https://doi.org/10.12928/telkomnika.v22i3.25793
[22] M. Mohammadi et al., “A comprehensive survey and taxonomy of the SVM-based intrusion detection systems,” J. Netw. Comput. Appl., vol. 178, p. 102983, Mar. 2021. https://doi.org/10.1016/j.jnca.2021.102983
[23] Y. Wang, Z. Pan, and J. Dong, “A new two-layer nearest neighbor selection method for kNN classifier,” Knowl.-Based Syst., vol. 235, p. 107604, Jan. 2022. https://doi.org/10.1016/j.knosys.2021.107604
[24] M. Rizky Pribadi, H. Dwi Purnomo, and H. Hendry, “A three-step combination strategy for addressing outliers and class imbalance in software defect prediction,” IAES Int. J. Artif. Intell. IJ-AI, vol. 13, no. 3, p. 2987, Sep. 2024. https://doi.org/10.11591/ijai.v13.i3.pp2987-2998
[25] M. K. Suryadi, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, “Comparative Study of Various Hyperparameter Tuning on Random Forest Classification With SMOTE and Feature Selection Using Genetic Algorithm in Software Defect Prediction,” J. Electron. Electromed. Eng. Med. Inform., vol. 6, no. 2, pp. 137–147, Mar. 2024. https://doi.org/10.35882/jeeemi.v6i2.375
[26] A. W. Dar and S. U. Farooq, “Handling class overlap and imbalance using overlap driven under-sampling with balanced random forest in software defect prediction,” Innov. Syst. Softw. Eng., Jun. 2024. https://doi.org/10.1007/s11334-024-00571-4
[27] S. Goyal, “Heterogeneous Stacked Ensemble Classifier for Software Defect Prediction,” in 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India: IEEE, Nov. 2020, pp. 126–130. https://doi.org/10.1109/PDGC50313.2020.9315754
[28] S. Feng, J. Keung, X. Yu, Y. Xiao, and M. Zhang, “Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction,” Inf. Softw. Technol., vol. 139, p. 106662, Nov. 2021. https://doi.org/10.1016/j.infsof.2021.106662
[29] A. Kurani, P. Doshi, A. Vakharia, and M. Shah, “A Comprehensive Comparative Study of Artificial Neural Network (ANN) and Support Vector Machines (SVM) on Stock Forecasting,” Ann. Data Sci., vol. 10, no. 1, pp. 183–208, Feb. 2023. https://doi.org/10.1007/s40745-021-00344-x
[30] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, “Ensemble Machine Learning Paradigms in Software Defect Prediction,” Procedia Comput. Sci., vol. 218, pp. 199–209, 2023. https://doi.org/10.1016/j.procs.2023.01.002
[31] Y. Z. Bala, P. Abdul Samat, K. Y. Sharif, and N. Manshor, “Improving Cross-Project Software Defect Prediction Method Through Transformation and Feature Selection Approach,” IEEE Access, vol. 11, pp. 2318–2326, 2023. https://doi.org/10.1109/ACCESS.2022.3231456
[32] M. H. D. M. Ribeiro, R. G. Da Silva, S. R. Moreno, V. C. Mariani, and L. D. S. Coelho, “Efficient bootstrap stacking ensemble learning model applied to wind power generation forecasting,” Int. J. Electr. Power Energy Syst., vol. 136, p. 107712, Mar. 2022. https://doi.org/10.1016/j.ijepes.2021.107712
[33] T. Wu, W. Zhang, X. Jiao, W. Guo, and Y. Alhaj Hamoud, “Evaluation of stacking and blending ensemble learning methods for estimating daily reference evapotranspiration,” Comput. Electron. Agric., vol. 184, p. 106039, May 2021. https://doi.org/10.1016/j.compag.2021.106039
[34] Z. Ding and L. Xing, “Improved software defect prediction using Pruned Histogram-based isolation forest,” Reliab. Eng. Syst. Saf., vol. 204, p. 107170, Dec. 2020. https://doi.org/10.1016/j.ress.2020.107170
[35] W. Huang et al., “Railway dangerous goods transportation system risk identification: Comparisons among SVM, PSO-SVM, GA-SVM and GS-SVM,” Appl. Soft Comput., vol. 109, p. 107541, Sep. 2021. https://doi.org/10.1016/j.asoc.2021.107541
[36] M. Rashid, J. Kamruzzaman, T. Imam, S. Wibowo, and S. Gordon, “A tree-based stacking ensemble technique with feature selection for network intrusion detection,” Appl. Intell., vol. 52, no. 9, pp. 9768–9781, Jul. 2022. https://doi.org/10.1007/s10489-021-02968-1
[37] N. A. A. Khleel and K. Nehéz, “Software defect prediction using a bidirectional LSTM network combined with oversampling techniques,” Clust. Comput., vol. 27, no. 3, pp. 3615–3638, Jun. 2024. https://doi.org/10.1007/s10586-023-04170-z

[38] N. Kardani, A. Zhou, M. Nazem, and S.-L. Shen, “Improved prediction of slope stability using a hybrid stacking ensemble method based on finite element analysis and field data,” J. Rock Mech. Geotech. Eng., vol. 13, no. 1, pp. 188–201, Feb. 2021. https://doi.org/10.1016/j.jrmge.2020.05.011

[39] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren, “StackGenVis: Alignment of Data, Algorithms, and Models for Stacking Ensemble Learning Using Performance Metrics,” IEEE Trans. Vis. Comput. Graph., vol. 27, no. 2, pp. 1547–1557, Feb. 2021. https://doi.org/10.1109/TVCG.2020.3030352

[40] R. Lazzarini, H. Tianfield, and V. Charissis, “A stacking ensemble of deep learning models for IoT intrusion detection,” Knowl.-Based Syst., vol. 279, p. 110941, Nov. 2023. https://doi.org/10.1016/j.knosys.2023.110941