Library Progress International
Vol.44 No.3, Jul-Dec 2024: P.3286-3306

Print version ISSN 0970 1052
Online version ISSN 2320 317X

Original Article

Available online at www.bpasjournals.com

Optimizing Software Defect Detection using advanced Feature Selection, Ensemble Learning, and Class Imbalance Solutions

Tarunim Sharma[1,*] Shalini Bhaskar[1] Aman Jatain[2], Kavita Pabreja[3]

[1] Amity School of Engineering and Technology, Gurgaon, Haryana 122413, India
[2]School of Engineering & Technology, K.R. Mangalam University,Haryana,India
[3]Lal Bahadur Shastri College, Dwarka,Sector-10, New Delhi 110058, India

How to cite this article: Tarunim Sharma, Shalini Bhaskar, Aman Jatain, Kavita Pabreja, Aman Jatain (2024 Optimizing Software Defect Detection using advanced Feature Selection, Ensemble Learning, and Class Imbalance Solutions. *Library Progress International*, 44(3), 3286-3306.

Abstract

Software Anamoly Prediction (SBP) is a vital process in software development,designed to identify potential software defects early in the development lifecycle. Early detection not only enhances software quality and performance but also significantly reduces development costs. The advent of Machine Learning (ML) algorithms has markedly improved the accuracy of bug prediction, leading to more efficient resource allocation and cost management. However, traditional ML models often struggle with managing non-linear relationships, addressing data imbalances, ensuring adequate feature representation, and handling complex scenarios, resulting in sub-optimal performance.This research proposes a novel approach that optimizes the selection and refinement of classifiers, improving the accuracy and reliability of SAP. A key focus of this study is on addressing class imbalance, a crucial factor that significantly impacts the accuracy of software defect detection, as evidenced by performance metrics. Moreover, feature selection, which involves removing irrelevant features from a dataset, is also identified as essential for building more effective learning models. Recent research have also emphasized the importance of tuning of model parameters in boosting the performance of individual classifiers in SAP tasks. Additionally, Ensemble Learning (EL) techniques have demonstrated superior accuracy and effectiveness when applied to SAP datasets.This research introduces an innovative model that integrates Ensemble Learning with hyperparameter tuning, alongside class imbalance handling and careful feature selection, to predict software bugs more effectively. The study investigates whether Ensemble Learning models outperform individual models in software bug prediction and if integrating hyperparameter optimization, class imbalance handling, and feature selection further boosts their accuracy.The findings underscore the key role of these integrated approaches in improving the predictive power of SAP models . The proposed model, tested using Python software, shows a substantial improvement in accuracy compared to single classifier models on the PROMISE repository's PC1 and CM1 data sets, highlighting the potential of these advanced methods in advancing software bug prediction.

KEYWORD: bug, ensemble, defect prediction, class imbalance, hyperparameter tuning

Introduction

For improving the software quality and reducing testing cost, prediction of software bugs is one solution that ensures only modules which are defective potentially will be sent to the testing phase. The process of coding, testing to fix bugs and then retesting (activity performed by Development team) to ensure high quality before giving it to the client is called defects cycle, which is part of Software Development Life Cycle. By predicting defects accurately, the flawed modules can be detected early preventing the expensive downstream issues which occur post release as illustrated in Fig.1 It helps to maintain software quality and reliability but asks for much resources before time. In order to address

these difficulties, Software Anomaly Prediction (SAP) systems have been proposed with the intent of automating defect identification — helping teams in finding and resolving potential problems as early as possible, saving resources and ensuring that software gets released on time.

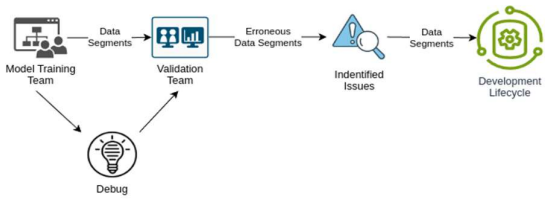

Fig 1. SDLC Workflow: From Code Development to Quality Assurance

Software defect prediction is the technique used to improve the quality of software and also reduce testing effort by sending the potentially defective modules to the testing phase. The testing team and the development team follow an iterative process where coding, testing, and defect fixing continue to happen repeatedly until perfect quality is achieved for that software. Defect prediction enables accurate identification of defect-prone modules at specific phases in SDLC, carryout an earlier detection towards faulty modules which will be undoubtedly reason behind disasters at final cycle as shown in Fig.1. Although it is resource-intensive, this early detection has an important role in the overall endeavor to ensure software quality and reliability. In order to address these problems, several research studies have been proposed that focus on automation bug-detection methods so as to reduce costs and risks early in the software process life cycle.

Software fault prediction can be effectively achieved by training statistical and machine learning models using dependent variables (like fault presence) and independent variables (such as software metrics). Figure 2 illustrates this process, where the model analyzes code segments, predicting faults and guiding the engineering team in fixing them. The updated segments are re-evaluated by quality assurance before re-entering the development cycle. The process continues iteratively with the integration of new data, ensuring continuous improvement in fault detection and resolution.

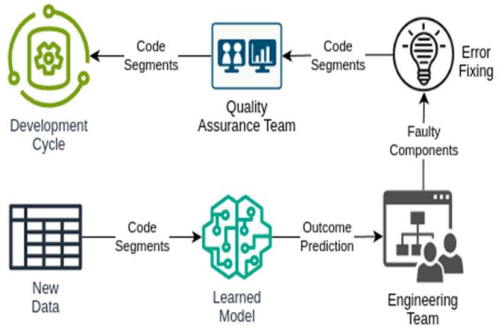

Fig 2.Workflow in SDP

Over the past three decades, a whole number of methods such as logistic regression, decision trees, Naive Bayes, support vector machines or more recently random forest have been used for software anomaly prediction. Machine learning is being used more and more, with which companies proactively predict bugs even before it can happen. Prediction accuracy is vital and raised through classifier models used in supervised learning, which learns from labeled (target) data. Nevertheless, they face practical issues related to overfitting and under fitting. Intuitively, some individual classifiers may not work well at detecting faults in all situations, so researchers have proposed ensemble learning methods which aggregate multiple classifiers to improve the accuracy of fault detection. In the ensemble learning approach, which combines several models to produce a final predictive result with higher prediction accuracy than any of the individual models. However, default settings are used in many studies which could have restricted the possible performance with these models. Types of Ensemble Techniques are:

Homogeneous and Heterogeneous. In the case of homogeneous ensembles, that refers to using the same base learner type throughout dataset subsets with methods such as boosting, rotating forests or bagging proving well in boosting model performance. Heterogeneous Ensembles combines different base learners, utilizing their strengths (where a single type might fail). This way is more flexible and it commonly leads to stronger models that are capable of capturing different patterns in the data. For instance, stacking where predictions are combined from various models via a meta-learner and voting classifiers aggregating predictions using majority votes or probability averaging to boost model accuracy while decreasing the chances of overfitting.

Recent studies emphasize the importance of Hyperparameter Optimization in improving the performance of individual classifiers in Software Bug Prediction (SBP) tasks. Ensemble models, when combined with hyperparameter tuning, show significant accuracy gains over single classifiers. Techniques like tuning Bayesian networks, SVM , and KNN have notably enhanced bug prediction performance.Another key challenge in SBP, as in many real-world datasets, is class imbalance, where one class significantly outnumbers the other. This imbalance causes models to misclassified minority class instances as majority class examples.To mitigate this, data-level techniques like SMOTE(Synthetic Minority Oversampling Technique) are used to balance the dataset by generating synthetic samples for the minority class. Additionally, feature selection plays a critical role by eliminating irrelevant features, reducing dataset complexity, and improving model accuracy.

Further studies have also highlights the advantages of combining re-sampling techniques with feature selection for imbalanced datasets,demonstrating that ensemble feature selection methods, which incorporate multiple techniques, generally outperform individual methods. Our analysis of various feature selection approaches confirmed that addressing class imbalance first, followed by applying ensemble feature selection, greatly improves model accuracy. Two main benefits of heterogeneous ensemble feature selection include:

1. Enhanced Robustness: By combining techniques, these methods address the limitations of individual approaches, resulting in more reliable and stable feature selection.

2. Improved Generalization: They capture a wider range of significant features, allowing the model to generalize better across various datasets and conditions.

Despite the modest performance improvement, combining ensemble feature selection with SMOTE yields the best results. This research emphasizes the benefits of integrating class imbalance handling, feature selection, and advanced ensemble methods like stacking and voting to enhance software defect detection accuracy.The contributions of this paper are threefold: (1) It offers a comprehensive comparison of filter, wrapper, and embedded feature selection algorithms, along with ensemble methods; (2) It assesses the effectiveness of combining feature selection (using both single and ensemble methods) with over-sampling in various sequences to re-balance datasets and reduce feature dimensions; (3) It identifies top-performing approaches and algorithms, highlighting the critical role of Hyperparameter Optimization in enhancing the performance of individual classifiers in Software Anomaly Prediction (SAP) tasks.The study explores two primary research questions:

(1) Do Ensemble Learning models outperform individual learning models in predicting software bugs?
(2) Does incorporating hyperparameter optimization into EL models, along with addressing class imbalance and feature selection, significantly enhance their accuracy?

The paper is organized as follows: Section 2 provides review on related work with respect to feature selection, over-sampling and ensembles and highlights the limitations of existing approaches. Section 3 describes the methodological approach used in the research. Section 4 gives the experimental settings and design. Performance evaluation is talked about in section 5, and the experiment results are discussed in section 6.Finally, Section 7 concludes the paper.

1.1 Objective, Innovation, and Impact

To solve software anomaly detection problem, a novel framework is proposed that adjusts the class imbalance and integrates heterogeneous feature selection method with diverse classifiers, utilizing stacking with voting ensemble.As illustrated in Figure 3, the workflow in software defect prediction begins with defect reports that are pre-processed, leading to prediction models that assess software metrics, ultimately guiding performance evaluation and module updates.

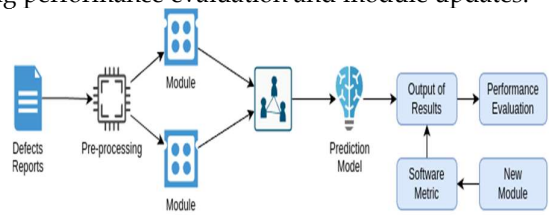

Fig 3. Software Defect Prediction Process

The research work emphasizes on enriching the predictive accuracy, stability and robustness by integrating several machine learning classifiers in an ensemble framework. Here in Figure 4, classification process is shown that illustrates how the data will be taken first and then clean it after this model learn according to its data, predict its results and, produce output by passing through proposed IHSVEM Model workflow. Finally, this model is verified on two NASA MDP datasets where it outperforms the current leading methods. This is achieved by introducing this new framework and demonstrating this in a thorough evaluation of its efficacy, benchmarking against high-level techniques in the field. This result quantifies the benefits of ensemble techniques, significantly improving predictive accuracy and surpassing all previous benchmarks, both academic and practical, in terms of performance scores.

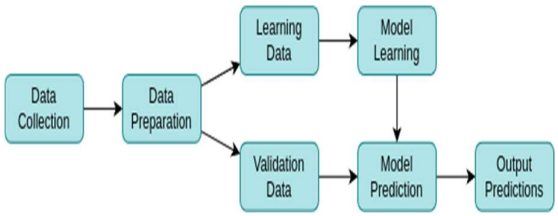

Fig 4. Classification Process

2. Related Work

Data over-sampling techniques are used, creating more synthetic minority samples, in order to deal with the class imbalance in datasets. A prime example of this is the Synthetic Minority Over-sampling Technique (SMOTE) a widely used method created to solve the imbalance. The SMOTE method generates new samples of the minority class within the feature space which is done by taking a random minority instance, finding k-nearest neighbors[1] and generating synthetic points along the lines between these neighboring instances. By doing so, we effectively overs-ample the minority class instances in the dataset. Different approaches either over-sampling methods or new arrivals for the problem of imbalanced data sets have pursued by various research studies. For example, Abdi and Hashemi [2] introduced an algorithm coupled with synthesizing technique based on synthetic data endowing the same Mahalanobis distance to class mean from the other minority class instances. Saez et al.[3] investigated class-dependent characteristics when using over-sampling on multi-class imbalanced datasets.Purnami and Trapsilasiwi [4] proposed the SMOTE-LSSVM, a second-stage classification method for multi-class imbalanced data, that optimizes the LS-SVM parameters with Particle Swarm Optimization (PSO), along with using SMOTE to balance the data.

Feng et al. [5] propose COSTE, a novel oversampling technique for software defect prediction (SDP) that addresses class imbalance by generating synthetic instances from defective instances of similar complexity. It improves defect detection accuracy, enhances data diversity, and accounts for testing effort. In tests on 23 releases from 10 projects, COSTE outperformed existing methods like SMOTE and MAHAKIL, with statistically significant results. Deng et al. [6] proposed an over-sampling method based on classification ranking and weight setting, where data within each class are sorted according to their distance from the hyperplane, and sampling weights consider data density and boundary sorting between adjacent classes. Cai et al.[7] introduce a hybrid model, HMOCS-US-SVM, to tackle class imbalance and SVM parameter optimization in software defect prediction. By simultaneously selecting non-defective samples and tuning SVM parameters, the model improves prediction accuracy. Tested on eight Promise datasets, it outperforms other models in key metrics like false positive rate, probability of detection, and G-mean. Additionally, Bulavas et al. [8] applied SMOTE to balance highly imbalanced network traffic datasets, demonstrating that over-sampling improves classifier performance compared to models without over-sampling. Despite these advancements, the impact of feature selection on over-sampling results remains unexplored. It is unclear whether selecting a feature subset from an imbalanced dataset can enable the over-sampling process to generate more representative synthetic examples for subsequent classifier training. Conversely, it is also uncertain whether performing feature selection after re-balancing the dataset leads to better classifier performance than applying feature selection alone. Feature selection identifies the most important features from a dataset to improve model performance. Starting with a matrix X_{m*n} of m features and n samples, the goal is to select k key features S_{k*n} that best distinguish between classes.Feature selection involves generating candidate subsets, evaluating them, stopping when optimal, and validating results. This process reduces data complexity and enhances model accuracy. Methods are categorized as filter (rank and remove based on statistics), wrapper (use a learning algorithm to optimize selection), and embedded (integrated into model training). While recent research combines feature selection with re-sampling for class imbalances, the impact on binary-class datasets is less explored.

Masanari Kondo et al. [9] recommend using "Correlation-Based (CFS) or Consistency-Based (ConFS)" methods for feature subset selection in supervised defect prediction models, as these approaches outperform original models and other reduction techniques, enhancing model efficiency and accuracy. Dao et al. [10] tackled the issues of high-dimensional features, such as noise and overfitting, with a two-stage feature selection method that reduces dimensionality while optimizing feature selection for better accuracy. Wang et al. [11] proposed an advanced approach to improve feature selection in defect prediction models by integrating a deep learning algorithm, boosting techniques, and the superposition criterion within a Nested-Stacking framework. This method, validated through extensive testing, excels in defect prediction and offers superior adaptability and generalization.Rostami [12] explored Swarm Intelligence techniques for feature selection, emphasizing the effectiveness of methods like Particle Swarm Optimization and Ant Colony Optimization. The study compares these methods, evaluating their strengths and weaknesses, and discusses factors affecting their success in feature selection. Hammouri et al. [13] conducted an in-depth analysis of three supervised ML models—Naive Bayes, Decision Tree, and Artificial Neural Networks—specifically designed for predicting software bugs. Their research, which compared these models with two previously proposed ones, found that the three ML models achieved significantly higher accuracy rates. Yang et al. [14] introduced a two-layer Ensemble Learning (TLEL) approach, integrating decision trees and ensemble techniques to improve just-in-time software defect prediction. Their model demonstrated a substantial performance boost over existing state-of-the-art ML models, identifying over 70% of bugs by reviewing just 20% of the code lines.

Di Nucci et al. [15] took an innovative approach by quantifying the dispersal of modifications made by developers on a component and using this data to construct a bug prediction model. Building on prior research focused on the human factor in bug generation, the study found that developers who were less focused were more likely to introduce defects. This model outperformed four competing techniques. Khan et al. [16] developed a model that combined ML classifiers with Artificial Immune Networks (AIN) to optimize hyperparameters, enhancing bug prediction accuracy. Testing seven ML algorithms on a bug prediction dataset, they discovered that hyperparameter tuning using AIN

outperformed classifiers with default settings. Qiu et al. [17] explored 15 imbalanced Ensemble Learning methods across 31 open-source projects. Their study revealed that models combining under-sampling and bagging were more effective in dealing with imbalanced datasets. Additionally, Pandey et al. [18] proposed a classification-based method for software bug prediction using a combination of Ensemble Learning techniques and Deep Representation. This approach showed superior performance compared to other EL techniques and state-of-the-art models across most datasets. The above reviewed literature has provided a comprehensive exploration of methodologies for addressing class imbalance in software defect prediction, homogeneous feature selection, encompassing supervised machine learning, Ensemble Learning, and deep representation techniques. These approaches, including advanced oversampling techniques like SMOTE, COSTE, and HMOCS-US-SVM, have demonstrated improved prediction accuracy by generating synthetic data, optimizing feature selection, and fine-tuning SVM parameters. Comparative analyses underscore the effectiveness of Ensemble Learning models and highlight the importance of hyperparameter optimization in enhancing bug prediction accuracy. Despite these advancements, the impact of feature selection in conjunction with oversampling remains under explored, particularly in multi-class and binary-class scenarios.These insights provide a strong foundation for exploring the key components of Hyperparameter Optimization and Ensemble Learning in software bug prediction in the background section. The literature also highlights the importance of further investigating the integration of deep learning, Ensemble Learning, and Optimization algorithms(swarm intelligence) to enhance prediction accuracy while effectively managing class imbalances. Numerous studies and literature reviews indicate that models incorporating class imbalance handling, effective attribute selection, and ensemble classifiers outperform those relying solely on individual classifier techniques based on these insights, an approach was developed by incorporating multiple classifiers to create a model that is interpretable, generalizable, and highly accurate in predicting anomalies in software.

3. Background

This section takes us through the major components which build the foundation of methodological approach in this research: treating class imbalance, focusing on feature engineering, hyperparameter tuning and harnessing ensemble learning. These factors contribute to an enhancement of the predictive performance of software defect prediction, and a thorough review on how they are fulfilling the objectives that have been set forth within the current study will be discussed in this section.

3.1 Over-sampling Technique

SMOTE, or Synthetic Minority Over-sampling Technique, is commonly employed in software defect prediction to address the challenge of class imbalance. The researchers describes the technique as useful for software defect datasets in which the number of non-defective modules (the majority class) far exceeds that of the defective ones (the minority class). The problem here is that due to this imbalance most predictive models built are dominated with majority class examples and fails most of the time in identifying defects. It generates imbalanced sets by creating new synthetic defective modules using the SMOTE algorithm to help balance training data . It takes the detected defective modules, calculates their nearest neighbors (usually based on a fixed k value) next it samples one of these neighbors and spawns a point between the original defect module and chosen neighbor. The dataset is augmented by the addition of these synthetic defects which helps to improve the representation of the minority class and further balance out the dataset. The major advantage of application of SMOTE in software defect prediction is that it can help the model to detect defects better as it provides a balanced training data. It hooks the model so that it can learn more from the incorrect modules and less prone to missing other defects later.Also, since SMOTE creating new synthetic instances rather than just replicating existing ones, it can reduce overfitting and improve the model generalization to new data. But still there are disadvantages of SMOTE. If the defective modules are not clearly defined, or if the minority class is filled with outliers, there is a risk of noise being introduced into the dataset. However, it performs best with numerical software metrics so it may not be most effective especially when you have categorical or textual features. SMOTE [18] is used as an over-sampling method in this study. Although many different over-sampling algorithms exist (some of which are simply modifications of SMOTE), it is

frequently referred to in the literature as a common over-sampling technique, and often used as a benchmark against which other re-sampling methods are compared.

3.2 Feature Engineering

Feature selection is a critical component in machine learning, particularly in the domain of predictive modeling for software defect prediction. The process involves selecting a subset of relevant features from a larger set to improve model performance, reduce complexity, and prevent overfitting. The significance of feature selection is more evident when examining various approaches and their effects on classifier performance.In the subsequent sections, a comparison is provided between individual feature selection techniques and hybrid methods.

Tables 1 and 2 present the performance metrics of various classifiers (Logistic Regression, Random Forest, Decision Tree, SVM, Naive Bayes, and KNN) on the CM1 and PC1 datasets, respectively, using different feature selection techniques such as Chi2, ANOVA, CFS, FFS, BFS, and RFE. Across both datasets, high values of precision, recall, and F1 scores are observed, particularly with Random Forest and Naive Bayes, where ensemble methods and regularization techniques like Lasso and Ridge maintain consistent performance. While the PC1 dataset generally shows slightly higher metrics across classifiers, both datasets highlight the effectiveness of using feature selection techniques in improving model accuracy and robustness.

CM1																		
Feature Selection Techniques	Chi2			ANOVA			CFS			FFS			BFS			RFE		
Classifier	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
LR	.77	.88	.82	.77	.88	.82	.95	.97	.96	.77	.88	.82	.77	.87	.82	.77	.87	.82
RF	.85	.88	.85	.77	.87	.82	.95	.97	.96	.82	.87	.83	.77	.87	.82	.77	.85	.81
DT	.83	.80	.81	.81	.84	.83	.95	.96	.95	.77	.87	.82	.81	.86	.83	.81	.83	.82
SVM	.77	.88	.82	.77	.88	.82	.95	.97	.96	.77	.87	.82	.77	.88	.82	.84	.87	.85
NB	.84	.87	.85	.84	.87	.85	.96	.95	.96	.77	.86	.81	.77	.86	.81	.84	.87	.85
KNN	.77	.88	.82	.82	.87	.83	.95	.97	.96	.77	.88	.82	.77	.86	.81	.77	.85	.81

Table 1: Performance measure of best-reduced feature subsets using the filter, wrapper, and embedded methods[28] using a variety of performance metrics such as Precision, Recall, and F1 score on the CM1 dataset

LASSO			RIDGE		
P	**R**	**F**	**P**	**R**	**F**
.77	.88	.82	.77	.88	.82
.77	.87	.82	.84	.88	.84
.79	.81	.80	.83	.85	.84
.77	.88	.82	.77	.88	.82
.85	.88	.85	.83	.86	.84
.77	.88	.82	.77	.88	.82

PC1																		
Feature Selection Techniques	Chi2			ANNOVA			CFS			FFS			BFS			RFE		
Classifier	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
LR	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.96	.95	.98	.98	.97
RF	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.97	.96
DT	.95	.94	.94	.95	.96	.95	.95	.96	.95	.95	.97	.96	.95	.97	.96	.95	.95	.95
SVM	.95	.97	.96	.95	.97	.96	.95	.97	.96	.97	.97	.96	.95	.97	.96	.95	.97	.96
NB	.96	.96	.96	.96	.95	.96	.96	.95	.96	.95	.97	.96	.95	.97	.96	.96	.94	.95
KNN	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.97	.96	.95	.97	.96

Table 2: Performance measure of best-reduced feature subsets using the filter, wrapper, and embedded methods[28] using a variety of performance metrics such as Precision, Recall, and F1 score on the PC1 dataset

LASSO			RIDGE		
P	**R**	**F**	**P**	**R**	**F**
.95	.96	.95	.95	.97	.96
.95	.97	.96	.95	.97	.96
.96	.97	.96	.95	.95	.95
.95	.97	.96	.95	.97	.96
.96	.93	.94	.96	.95	.95
.95	.97	.96	.95	.97	.96

Table 3 below demonstrates the enhanced performance achieved by combining class imbalance handling (SMOTE), hybrid feature selection techniques (Recursive Feature Elimination, Sequential Feature Selection), and ensemble methods like Random Forest, Decision Tree, Adaboost, Bagging, Voting, and Stacking. Compared to previously discussed tables focusing on individual feature selection methods, these hybrid and ensemble approaches show significant improvements in performance metrics. For example, SMOTE, RFE , SFS and Random Forest delivers 98% in accuracy, precision, recall, and F1-score, surpassing the typical 95-96% seen with standalone techniques. Additionally, methods such as Bagging, Stacking, and Voting consistently maintain high performance with 94% across all metrics, indicating the reliability of ensemble approaches. The advantages of these combinations include better generalization, reduced overfitting by leveraging multiple models, and improved handling of class imbalance, leading to more accurate and consistent results. These hybrid techniques are particularly beneficial for imbalanced datasets, providing stronger overall performance than individual methods. By combining multiple feature selection methods, ensemble approaches capitalize on the strengths of different algorithms, enhancing both the precision and recall across classifiers. This highlights the benefit of hybrid approaches, which result in better overall accuracy and more robust performance.

Class Imbalance +Feature Selection Techniques	Accuracy	P	R	F1
Smote + RFE + SFS + Random Forest	98	98	98	98
Smote + RFE + SFS + Decision Tree	80	82	80	81
Smote + RFE +SFS +KNN	88	94	88	91
Smote + RFE + SFS + ADABOOST	92	94	92	93
Smote + RFE +SFS + Bagging	94	94	94	94
Smote + RFE + SFS + Voting	92	93	92	92
Smote + RFE + SFS + Stacking	94	94	94	94

Table 3. Class Imbalance and Feature Selection Models Performance

Acknowledging the strength of a hybrid feature selection method in enhancing model performance, three critical feature selection approaches namely two filter methods — Mutual Information (MI) and Principal Component Analysis (PCA), along with Recursive Feature Elimination with Cross-Validation (RFECV)— were implemented simultaneously, forming a pipeline in this study. This section discusses all these feature selection techniques as well as the benefits of combination of them into one hybrid approach. In this study, a systematic feature selection criterion was executed over the pre-processed dataset to improve the prediction accuracy of software defect prediction models. All of these feature selection strategies together played vital role in reducing the feature set as much as possible and thus creating a resilient efficient model.

Mutual Information (MI) was the first technique applied to the dataset. MI was used to measure the dependency between each feature and the target variable, allowing for the identification of features that provide significant information about the target. This method is particularly effective because it captures non-linear relationships between features and the target variable, and it does not assume any specific distribution of the data. By applying MI, features with the highest relevance to the prediction of software defects were retained, forming an initial, refined set of features. Following the application of MI, Recursive Feature Elimination with Cross-Validation (RFECV) was employed to further narrow down the feature set. RFECV works by iteratively removing the least important features based on model performance. At each iteration, the model was trained, and features were ranked according to their importance scores. The process continued until an optimal subset of features was identified. Cross-validation within this method ensured that the selected features were not only relevant but also generalized well to unseen data. This step was crucial in avoiding overfitting and ensuring the robustness of the model.

Finally, Principal Component Analysis (PCA) was applied to the feature set resulting from the RFECV process. PCA is a dimensionality reduction technique that transforms the original features into a new set of uncorrelated components, known as principal components. These components are linear combinations of the original features and are ranked based on the amount of variance they explain in the data. By applying PCA, the dimensionality of the feature space was further reduced while preserving the most critical information. This reduction not only helped in speeding up the training process but also improved the overall performance of the model by eliminating any remaining noise in the data.

The combination of these three techniques allowed for a comprehensive and effective feature selection process. The feature selection pipeline with Mutual Information (MI), Recursive Feature Elimination with Cross-Validation (RFECV) and Principal Component Analysis (PCA) is introduced to maximize model performance by selecting the relevant features through MI, refining them iteratively via

elimination while generalizing through RFECV and dimensionality reduction for computational efficiency and noise reduction in PCA. By harnessing this hybrid, the predictive performance and model stability get a liftoff. Each step progressively refined the feature set, ensuring that the final model was both efficient and highly accurate in predicting software defects. This approach was particularly beneficial in addressing the complexity and variability inherent in the dataset, ultimately leading to more reliable predictions.

So to enhance the accuracy of a classification model, especially when managing imbalanced datasets and numerous features, the following procedure is used to balance the data and identify the most important variables for training and testing.

1. Begin with the original dataset Z, which consists of n observations, each described by m variables.

2. Partition the dataset Z into a learning subset P and an evaluation subset Q.

3. Apply an oversampling method to the learning subset P to balance the class distribution, resulting in a modified subset P_balanced.

4. Execute a feature selection process on P_balanced to reduce the number of variables from m to r (where r < m).

5. Create a refined learning subset P_selected, containing only the r chosen variables.

6. Use P_selected to train the classification algorithm.

7. Implement the hybrid feature selection process on the evaluation subset Q, producing Q_selected with the identical r variables.

8. Test the trained model using the Q_selected subset to assess its performance.

To evaluate the proposed methods, two binary-class imbalanced datasets – the NASA MDP (Metric Data Program) are used in this work. This Table 4 represents some basic information of these datasets such as the number of features, instances, and classes which are sorted by the number of features in ascending manner. The imbalance ratio is calculated for each dataset as the number of the largest class divided by that of the smallest class. Datasets were iteratively divided into training datasets (80% of the data) and testing dataset (20% of the data), using 10-fold cross-validation. To remove the bias from the experiment, and to show a homogeneous result, the original imbalance ratio of the dataset was kept as it is while dividing into different

Dataset	Number of Features		Number of Instances	Number of classes	Imbalance Ratio	Evaluation Metrics
CM1	21		1105	2(Defective, Non-Defective)	5.2	Precision,Recall, F1-Score, AUC ROC
PC1	21		498	2(Defective, Non-Defective)	9.2	Precision,Recall, F1-Score, AUC ROC

Table 4. Dataset Description

training sets which also been imbalanced.

3.3 Aggregated Learning

Ensemble Learning (EL) algorithms stand out from traditional single-hypothesis approaches by utilizing multiple hypotheses instead of depending on a single, optimal solution. Rather than focusing on one best explanation, EL algorithms generate a variety of hypotheses and combine them to classify new data points more effectively and with greater diversity. Studies have consistently shown that ensemble methods are generally more dependable than individual models. Fraihat et al. [21] conclude that Software Bug Prediction (SBP) is crucial for improving software quality and reducing costs by predicting bugs early. Their study shows that using Machine Learning (ML) with Ensemble Learning (EL) and Hyperparameter Optimization significantly boosts prediction accuracy.In a comparison using NASA's dataset (10,885 instances, 20 attributes) and WEKA, EL models outperformed single classifiers, with further accuracy gains from hyperparameter tuning. This confirms EL with optimization as an effective strategy for enhancing SBP performance highlighting the robustness and adaptability of EL in diverse scenarios.

An "ensemble" is a set of base learners (Individual models) whose generalization may be better than an individual model. Therefore, EL is specifically useful as it can upscale the performance of weak learners — i.e. learners only slightly better than random guessing process— to an excellent prediction accuracy. Theoretical studies are biased toward weak learners; in practice, however, EL applications often include strong base learners which lead to an even better performance from the ensemble. Bagging, Adaboost, Stacking and Voting are some of the frequently used EL methods.

Bagging, or bootstrap aggregation, is a foundational ensemble technique that boosts model performance by creating diversity among base learners. It works by generating multiple bootstrapped samples from the original dataset, where each sample is a random subset of the data with replacement. Different classifiers are trained on these subsets, and their predictions are aggregated using majority voting. This approach reduces variance, making Bagging particularly effective for high-variance models like decision trees, which are prone to overfitting. Adaboost is another popular ensemble method that improves accuracy by training a sequence of models, where each model focuses on correcting the errors of its predecessors. During this process, the algorithm increases the weight of misclassified examples, making them more influential in subsequent rounds. This iterative process helps Adaboost gradually refine its ability to handle difficult cases, significantly enhancing the ensemble's overall performance. Stacking, or stacked generalization, is an ensemble technique that combines the predictions of multiple base models using a meta-model to improve accuracy. It allows for the use of diverse models, capturing a wider range of patterns in the data. There are several approaches to stacking, including simple stacking, where a single meta-model combines base models' outputs; blending, which uses a holdout set to train the meta-model and reduce overfitting; multi-layer stacking, which adds complexity by layering meta-models; and cross-validation stacking, which uses cross-validation to generate unbiased predictions for the meta-model. Voting is another ensemble technique that aggregates predictions from multiple classifiers to reach a final decision. The three primary voting strategies are unanimous, majority, and plurality. Unanimous voting requires all classifiers to agree on the decision, majority voting selects the class agreed upon by more than half of the classifiers, and plurality voting chooses the class with the most votes, even if it doesn't represent an absolute majority. These strategies enhance the ensemble's robustness and accuracy by combining the strengths of different models. Random Forests extend the Bagging method by not only using random bootstrap samples for each tree but also selecting a random subset of features for each node split. This added randomness increases diversity among the trees, improving the ensemble's robustness. The trees in a Random Forest are fully grown and unpruned, capturing detailed patterns. By combining the outputs of these diverse, fully grown trees, Random Forests achieve high accuracy and stability, making them a widely used and effective model.The following methodology section will delve into the practical application of class imbalance, ensemble feature selection and classifier with Hyperparameter Optimization within the context of software bug prediction, demonstrating how these techniques can be effectively utilized to enhance predictive accuracy.

3.4 Hyperparameter Optimization

In machine learning, "optimization" involves fine-tuning hyperparameters—such as regularization, kernels, and learning rates—to enhance model performance. These adjustments are crucial for improving classifier accuracy throughout the learning, model construction, and evaluation phases. Several methods are commonly employed for hyperparameter optimization. Manual Search relies on expert intuition to iteratively select, evaluate, and refine hyperparameters, although it may not fully explore the parameter space. Grid Search, in contrast, systematically examines all possible combinations of predefined hyperparameters, offering thorough coverage but at a high computational cost. Random Search provides a more efficient alternative by randomly selecting combinations, often yielding effective solutions with less computational effort. Bayesian Optimization further refines this process by predicting and focusing on the most promising hyperparameter configurations, achieving near-optimal results with fewer iterations. Evolutionary Algorithms take a different approach by mimicking natural selection to evolve and combine the best hyperparameter settings, making them particularly useful for complex optimization tasks. Typically, these methods are paired with cross-validation to ensure the chosen hyperparameters generalize well to new data, ultimately helping to maximize model accuracy and overall performance. Nevendra et al.[20] study highlights the importance of hyperparameter tuning in improving defect prediction accuracy in software systems. Through analysis of 15 software defect datasets, the research shows that optimizing hyperparameters significantly boosts model performance. The study underscores the effectiveness of thorough hyperparameter exploration, particularly through grid search, in achieving substantial accuracy gains, especially for parameter-sensitive models.As suggested by Haidar Osman et al.[19] explored the impact of hyperparameter tuning on bug prediction accuracy in machine learning models. They studied k-nearest neighbors (IBK) and support vector machines (SVM) across five open-source Java systems. The study found that tuning hyperparameters significantly improved accuracy for IBK and maintained or enhanced it for SVM, showing that models often benefit from customized settings over default ones. The authors recommend hyperparameter tuning as essential for optimizing bug prediction models.Having established the importance of hyperparameter optimization in boosting classifier performance, it is now crucial to explore how these optimized models are implemented in real-world scenarios. The next section presents an advanced ensemble model specifically tailored for software defect prediction, utilizing multiple heterogeneous supervised machine learning classifiers. This approach capitalizes on the benefits of hyperparameter tuning to further improve accuracy and efficiency in detecting software defects.

4. Experimental Design and Approach

The paper puts forward an advanced ensemble model for software defect prediction utilizing multiple heterogeneous supervised machine learning classifiers to provide higher accuracy. This approach solves the problems predicting software defects more effectively. Through a stacking with voting ensemble approach that aggregates outputs of single base classifiers, IHSVEM model improves the predictive strength in more accurate defect detection. As depicted in Figure 5 this method involves data preprocessing, base classification, ensemble classification and subsequently, testing on the entire dataset to obtain a final prediction. Figure 5 illustrates the structure of the IHSVEM model, which has two main layers — training and testing; its training phase includes three major steps — data preprocessing, base classification, and ensemble classification.

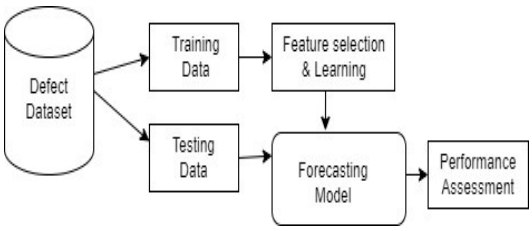

Fig 5. Training and Testing Data

In the training layer, the dataset is cleansed and formatted for accurate classification. The results from the base classifiers are then combined using an ensemble classifier, forming the model-based ensemble. The testing layer targets exclusively the prediction, using the trained model to predict defect in software modules that have never been seen before. This work was conducted using a Python-based tool, which helped in efficient data preprocessing, advanced statistical analysis and precise machine learning modeling which eventually led to the discovery of important insights from the research data. The following steps were performed to locate faulty modules of the software:

1. Data Collection: Initially, datasets comprising various software metrics were gathered and reused [22].

2. Data Preprocessing:The datasets were subjected to preprocessing, which involved dataset splitting, cleaning, and normalization and handling class imbalance using smote [23], [24].

3. Feature Selection: A hybrid feature selection method was used to identify the most relevant features, reducing the dataset's dimensionality.

4. Model Training: The IHSVEM model was trained using a diverse combination of base classifiers.

5. Ensemble Integration:The base classifiers were combined using an ensemble learning technique, which aggregated their accuracy to produce unbiased and more accurate results.

6.Defect Prediction:Finally, the preprocessing technique was applied to new modules, and the dataset was fed into the trained model to predict defective modules.

Stacking with Voting Intelligent Hybrid Ensemble model can be defined as a mix of two main ensemble techniques: Stacking and Voting. Being an ensemble, the representation here takes its own process in which the outputs from base models are either stacked or boosted and later gets optimized by a voting mechanism. Mathematically, it is found as follows:

Base Models: Suppose you have n base models M_1, M_2, …, M_n. Each model M_i takes the input vector X and produces an output prediction $Y_i = M_i(X)$

Stacking Layer: The predictions from all base models $Y_1, Y_2, \ldots, Y_n$ are used as input features for a meta-model M_{meta}, which produces the final stacked prediction:

$$Y_{stack} = M_{meta}(Y_1, Y_2, \ldots, Y_n)$$

Here, Y_{stack} represents the output after stacking the base models.

Voting Mechanism: In the final step, the output from the stacking model Y_{stack} is combined with the predictions from the base models through a voting mechanism (e.g., majority voting or weighted voting) to produce the final prediction Y_{final}:

$$Y_{final}=V(Y1,Y2,\ldots,Yn,Y_{stack})$$

Where V represents the voting function that aggregates the predictions. The primary goal of the proposed approach is to accurately predict defective software modules. The overall mapping of the Stacking with Voting Intelligent Hybrid Ensemble model can be represented as follows:

$$Y_{final}=V(M_1(X),M_2(X),\ldots,M_n(X),M_{meta}(M_1(X),M_2(X),\ldots,M_n(X)))$$

This equation illustrates that the final prediction Y_{final} is obtained by first stacking the predictions from multiple base models and then applying a voting mechanism to combine these stacked predictions with the original base model outputs. This hybrid approach leverages both stacking and voting to enhance

the model's predictive performance. Each software module X is characterized by multiple attributes: $X = x_1, x_2, x_3, ….., x_n$ where $x_1, x_2, x_3, …, x_n$ denote the attributes associated with the software module. This research aims to discover this mapping through ensemble-based machine learning techniques.

4.1 Dataset Selection and Preprocessing

The first step in this study involved selecting historical software defect datasets from NASA's MDP repository, specifically choosing benchmark datasets like CM1, PC1, and PC4. These datasets were selected to ensure alignment with previous research. Each dataset corresponds to a unique software component, where each instance represents a software module. These modules contain various software quality metrics, such as LOC_COMMENT, LOC_TOTAL_CALL_PAIRS, HALSTEAD_LENGTH, and HALSTEAD_CONSTANT, all documented during the Software Development Life Cycle (SDLC). To evaluate classifier performance, the datasets were split into 80% for training and 20% for testing. This method maintained the original dataset's imbalance ratio within each training fold, thereby minimizing the risk of bias in the results. Each software module includes multiple independent variables (e.g., $x_1, x_2, x_3,…$) for prediction, while the dependent variable, or target, indicates whether a module is defective (Y) or non-defective (N).Pre-processing, the second phase in the training layer, includes three essential steps: dataset splitting, cleaning, and normalization. In the first step, the pre-processed dataset is split into training and testing datasets with a 80:20 ratio using a class-based approach to ultimately assure the segregation of relevant and irrelevant data for the purpose of training and testing the model[22]. Second, cleaning is essential for improving the model's robustness and accuracy. Cleaning is performed through the elimination of the outliers and undesired data points, reduction of noise, and imputation of missing values using the mean technique to improve prediction and data quality[23]. Third, normalization is applied to scale the input features within a range of 0 to 1[24], enhancing the model's convergence, stability, and processing speed. It prevents the model form feature domination due to limited range; thus, contributing to the balanced learning and consistency of the model across all datasets[25]. This step is essential for overall model robustness and generality in the IHSVEM.

4.2 Classification

Through optimization, it was determined that the Support Vector Machine (SVM) performs optimally with a regularization parameter (C = 1), an RBF kernel, and a class weight setting of {0: 1, 1: 5}, effectively handling class imbalance. SVM is particularly strong in managing non-linear data due to its kernel functions. Adaboost, known for enhancing model performance by focusing on difficult-to-classify instances, is paired with the XGBoost classifier, which showed the best results with 200 estimators, a learning rate of 0.1, and a maximum depth of 7. Similarly, CatBoost, which efficiently handles categorical variables and reduces overfitting, performed best with 200 iterations, a learning rate of 0.1, and a depth of 7.

Gradient Boosting, recognized for its sequential error correction capabilities, was optimized with 200 estimators, a learning rate of 0.1, and a maximum depth of 7. Logistic Regression, valued for its simplicity and effectiveness in binary classification tasks, was fine-tuned with a regularization parameter (C = 1) and a class weight setting of {0: 1, 1: 10} to better manage class imbalance. Finally, Random Forest, which excels in capturing complex data relationships, was optimized with 200 estimators, a maximum depth of 20, and a minimum sample split of 2 . After optimizing these six classifiers, they were integrated into a stacking model, where their predictions were combined using a Random Forest as the meta-model. Following the stacking process, a voting ensemble was implemented, incorporating the stacked model along with XGBoost, Adaboost, and Gradient Boosting classifiers. This final step leveraged the complementary strengths of the stacked ensemble, which combines diverse model outputs for greater accuracy, and the individual models, which excel in specific aspects like handling non-linear relationships, correcting errors sequentially, and dealing with class imbalances. As a result, this approach not only ensures robustness and accuracy across various datasets but also enhances the model's adaptability and resilience, making it well-suited for complex and varied predictive tasks.

4.3 Ensemble modelling

Ensemble modeling, as the fourth step in the training process, involves the integration of multiple individual models to enhance prediction accuracy and classification performance. This approach leverages the power of ensemble-learning, as different models perform well in certain tasks may be able to mine a more diverse set of patterns and knowledge than merely thrashing on one model[26]. In this work voting ensemble method is used to achieve reliable and accurate predictions. By combining the strengths of each base model, more dependable predictions are obtained through voting in the ensemble. Due to the diversity of the ensemble, biases and errors that may exist in any one model are mitigated; this leads to a more complete view on software quality attributes. This diversity also makes the model more resistant to outliers and noise in the data which helps make better and stable predictions. As an added bonus, this ensemble approach effectively mitigates the risk of overfitting, which is typical in machine learning and pertains to models accommodating themselves too closely to particular patterns in the training data.

Ensemble models are particularly useful for software defect prediction, in which we need higher precision and reliability to predict a model against bugs by merging predictions from multiple sources contribute improving the accuracy overall. The above model improves the overall accuracy by using ensemble method to aggregate predictions of multiple stream based machine or data learning models. This fact is well suited in certain types of tasks, one of which is anomaly prediction, requiring high precision and reliability for security and quality assurance. This model combined the predictive outputs produced by six different base learners into a ensemble stacking model, with the meta-model being of the form of Random Forest. After stacking all the models, a Voting Ensemble was employed with these predicted probabilities as inputs for the stacked model along with XGBoost, Adaboost and Gradient Boosting classifiers. The former is deep in the sense that it uses stacking, whereby high-level outputs from multiple models are combined to create one meta-model; and using of voting which refines predictions by considering collective opinion among all models. This hybrid approach allows each technique to learn and perform better than they would individually. Experiments on datasets demonstrate that the IHSVEM model significantly outperforms single base classifiers by utilizing this ensemble strategy over the original base models.

4.4 Advancing Defect Detection with the IHSVEM Framework

To overcome such challenges this study proposes a novel stacking with voting ensemble based software defect prediction model[27]. In the testing phase, this model is deployed to serve its purpose that requires only one task of making real-time predictions for a different software package which has never been seen before. At this phase, the function f(X) is applied to unlabeled data for appropriate labelling of each module. The obtained results indicate that the proposed approach has a minimum error rate ε compared to state-of-the-art software defect prediction methods. The predicted output Y by the function f(X) is then given to the development team in the Software Development Life Cycle (SDLC). This enables the team to find and rectify defective modules before they are passed to the testing team that streamlines the quality assurance process and reduces costs for the organization.

5. Performance evaluation

In this study, the performance of the proposed software defect prediction model SIHVE is assessed using several metrics, including accuracy, recall, precision, F1 score, AUC-ROC, and PR AUC. Because the dataset is imbalanced, accuracy may not be a reliable indicator of performance, as it might not adequately reflect how well the model identifies instances from the minority class. Even if accuracy appears high, it may fail to capture the model's effectiveness in detecting the minority class instances as it is the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances. The formula for accuracy is:

$$\text{Accuracy} = \text{Correct Predictions} / \text{Total Predictions}$$

Thus, additional metrics which can be considered for a more comprehensive evaluation are:

1. Precision is the ratio of true positive predictions to the total number of positive predictions, highlighting the model's ability to accurately identify relevant instances. The precision formula is:

Precision=Correct Positive Predictions/All Positive Predictions

2. Recall measures the proportion of actual positive instances that the model correctly identifies, ensuring that the minority class is adequately represented. The recall formula is:

Recall= Correct Positive Predictions/Correct Positive Predictions +Missed Positive Instances

3. F1 Score is the harmonic mean of precision and recall, balancing both metrics to provide a single measure of the model's performance. This is particularly useful in imbalanced datasets. The F1 score is calculated as:

F1 Score= 2* (Precision*Recall)/Sum of Precision and Recall

Another important metric for evaluating model performance is AUC-ROC (Area Under the Receiver Operating Characteristic Curve). An ROC curve shows the model's ability to distinguish between classes by plotting the true positive rate (recall) against the false positive rate (1-Specificity). The AUC measures the probability that, given a randomly chosen positive and negative instance, the model will correctly rank them. A higher AUC indicates a better model.

Additionally, PR AUC (Precision-Recall Area Under the Curve) is particularly useful for imbalanced datasets. The PR curve plots precision as a function of recall, focusing on the quality of the positive class predictions. For imbalanced datasets where the rarer class is of greater interest, a higher PR AUC score indicates the model's ability to correctly identify positive instances, which are typically fewer in number.Together, these metrics provide a comprehensive assessment of the model's performance, ensuring that evaluation covers various aspects of prediction accuracy, especially when dealing with imbalanced data.

6. Result and Discussion

The performance comparison between PC1 and CM1 datasets without addressing class imbalance presents several notable findings, as analyzed in Table 5 . The Intelligent Heterogeneous Stacked Voting Model achieves high accuracy (92%) on PC1, showing balanced precision, recall, and F1-score. However, its AUC-ROC (86%) and PR Curve AUC (39%) reveal limitations in handling minority class instances, as reflected by its moderate MCC (0.37). On CM1, the model's performance declines, with accuracy reduced to 87% and an MCC of -0.057, indicating difficulty in managing class imbalance. Similarly, Random Forest performs well on PC1 (92% accuracy) but struggles on CM1 with a negative MCC (-0.04). The Logistic Regression model displays variability, with reasonable performance on PC1 (81% accuracy) but lower scores on CM1 (83% accuracy, 58% AUC-ROC). Models like SVM , Naive Bayes , and KNN maintain high accuracy on PC1 (91%-92%) but show a significant drop in AUC-ROC and MCC on CM1, with SVM particularly under-performing with an MCC of -0.018. Leaving the class imbalance unaddressed may reduce performance, particularly on the CM1 dataset. Metrics such as MCC, PR Curve AUC, and AUC-ROC—especially on models—are significantly worse at detecting minority class instances. For instance, SVM MCC shows a negative value (-0.018), highlighting the difficulty in predicting minority class instances even when examining specific cases in isolation. Failing to account for class imbalance can cause models to be biased towards the majority class, resulting in inflated accuracy while performing poorly in terms of recall (ability to correctly identify) and precision (accuracy of classifications) for minority classes. This leads to a biased overall performance, diminishing the effectiveness of the models for defect prediction in software. There are techniques like SMOTE or undersampling, and applying one of these would likely make the results a lot better by generating better classifiers system that needs to detect software defects as well as balance the class representation in favor of the smallest ones. Figure 6 below also shows a comparison of precision, recall and F1-score with regard to PC1 and CM1 datasets which helps visualize the performance difference amongst the datasets.

.**Table 5.** Performance Comparison of Optimized Model Frameworks without handling Class Imbalance on PC1 and CM1 Datasets

Datasets	Optimized Model Framework	Accuracy	Precision	Recall	F1-SCORE	AUC ROC	PR Curve AUC	MCC
PC1	**IHSVEM** MI RFE CV PCA + Stacked Voting ('SVM', 'XGBoost', 'CatBoost', 'Gradient Boosting', 'Logistic Regression') + Soft Voting ('Stacking', 'XGBoost', 'Adaboost', 'Gradient Boosting')	92	92	92	92	86	39	0.37
CM1		87	80	87	83	75	20	-0.057
PC1	MI RFE CV PCA + Random Forest + Optimized Hyperparameters + K-Fold Cross Validation	92	91	92	91	91	37	0.31
CM1		88	80	88	84	72	17	-0.04
PC1	MI RFE CV PCA + Logistic Regression + Optimized Hyperparameters + K-Fold Cross Validation	81	91	81	85	71	35	24
CM1		83	85	83	84	58	16	19
PC1	MI RFE CV PCA + SVM + Optimized Hyperparameters + K-Fold Cross Validation	92	86	92	89	73	19	-0.018
CM1		89	80	89	84	54	16	0
PC1	MI RFE CV PCA + Naive Bayes + Optimized Hyperparameters + K-Fold Cross Validation	91	90	91	90	80	29	24
CM1		87	83	87	85	59	20	11
PC1	MI RFE CV PCA + KNN + Optimized Hyperparameters + K-Fold Cross Validation	92	91	92	91	80	32	0.31
CM1		89	80	89	84	51	11	-0.028

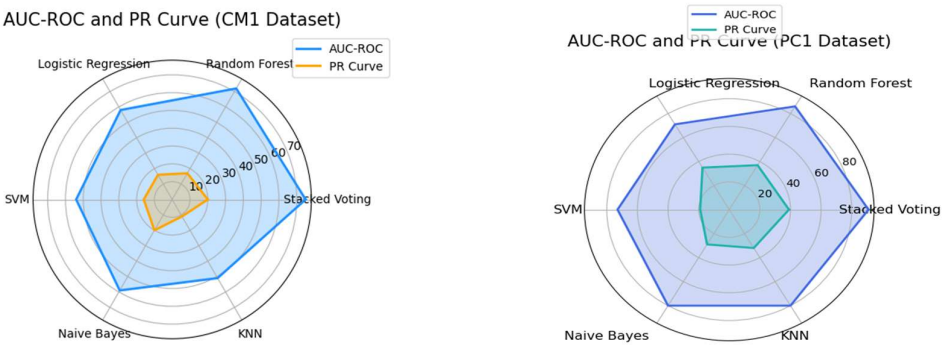

Fig 6. Performance Metrics(AUC ROC & PR Curve) on CM1and PC1 testing Dataset without implementing SMOTE Across Optimized Model Frameworks

The Smart Ensemble Stack consistently stands out for its ability to manage class imbalances and handle complex datasets like PC1 and CM1 , as shown in Table 6. It achieves superior accuracy, maintaining 95% on PC1 and 85% on CM1, while balancing precision, recall, and F1-scores effectively. Its AUC-ROC values—93% on PC1 and 77% on CM1—demonstrate its capability to differentiate between classes in challenging scenarios, outperforming individual models like Random Forest, Naive Bayes, and Adaboost. Even though CM1 is more complex, the Smart Ensemble Stack remains highly

adaptable, showing minimal performance decline compared to other models, as further visualized in Figure 7.

What makes the Smart Ensemble Stack exceptional is its integration of diverse techniques, such as stacking and voting mechanisms, along with advanced feature selection and class balancing strategies like SMOTE. It achieves superior performance metrics, particularly on the PC1 dataset, and maintains strong results on the more challenging CM1 dataset.This ensures high stability and generalization across different datasets, making it well-suited for real-world software defect prediction tasks. The combination of multiple classifiers, including SVM, XGBoost, CatBoost, and Logistic Regression, allows it to leverage the strengths of each model, leading to better accuracy, robustness, and adaptability. This ability to handle complex, imbalanced datasets effectively is what makes the Smart Ensemble Stack a powerful tool for software defect prediction as highlighted in the performance comparison in Figure 7. The analysis underscores the importance of using ensemble models like this, which are better suited to handle the intricacies of real-world datasets, especially those with imbalanced class distributions.

Table 6. Performance Comparison of Optimized Model Frameworks handling Class Imbalance using SMOTE on PC1 and CM1 Datasets

Datasets	Optimized Model Framework	Accuracy	Precision	Recall	F1-SCORE	AUC ROC	PR Curve AUC	MCC
PC1	**IHSVEM** Smote +MI RFE CV PCA + Stacked Voting ('SVM', 'XGBoost', 'CatBoost', 'Gradient Boosting', 'Logistic Regression') + Soft Voting ('Stacking', 'XGBoost', 'Adaboost', 'Gradient Boosting')	95	95	95	95	93	62	0.5
CM1		85	82	85	83	77	22	-0.014
PC1	Smote+MI RFE CV PCA + Random Forest + Optimized Hyperparameters + K-Fold Cross Validation	90	93	90	91	90	44	42
CM1		85	82	85	83	72	19	0.068
PC1	Smote+MI RFE CV PCA + Logistic Regression + Optimized Hyperparameters + K-Fold Cross Validation	84	90	84	87	70	23	0.23
CM1		79	85	79	81	69	17	0.18
PC1	Smote+MI RFE CV PCA + SVM + Optimized Hyperparameters + K-Fold Cross Validation	83	91	83	86	73	19	0.16
CM1		75	82	75	78	60	14	0.043
PC1	Smote+MI RFE CV PCA + Naive Bayes + Optimized Hyperparameters + K-Fold Cross Validation	91	90	91	90	77	28	0.24
CM1		87	83	87	85	73	21	0.11
PC1	Smote+MI RFE CV PCA + KNN + Optimized Hyperparameters + K-Fold Cross Validation	82	92	82	86	81	46	0.35
CM1		75	84	75	79	61	24	0.13

The analysis of the performance metrics for various models applied to the PC1 and CM1 datasets demonstrates the effectiveness of the Optimized Model Framework, which integrates SMOTE for class imbalance, feature selection techniques (MI, RFECV, PCA), hyperparameter tuning, and multiple

classifiers. Overall, the Smart Ensemble Stack leads in most metrics, particularly on PC1, showcasing its robustness and ability to handle class imbalances effectively. While the CM1 dataset presents more difficulty for all models, the Optimized Model Framework continues to demonstrate adaptability and strong performance across varying conditions. Each model displays unique strengths depending on the dataset, emphasizing the importance of tailored approaches in software defect prediction.

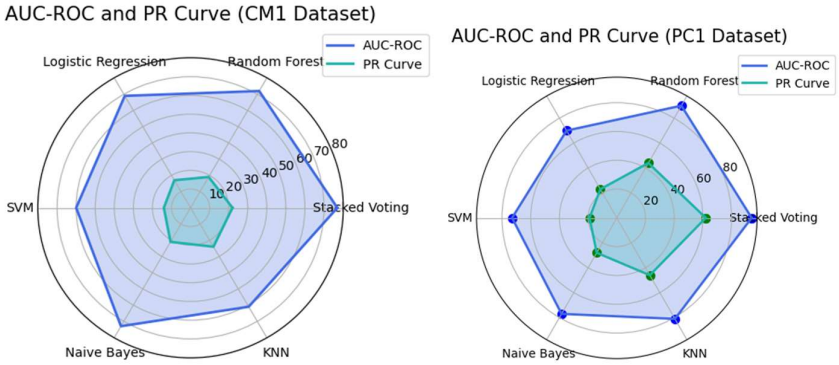

Fig 7. Performance Metrics(AUC ROC & PR Curve) on CM1and PC1 testing Dataset Across Optimized Model Frameworks with SMOTE

7. Conclusion

The study highlights the effectiveness of the Smart Ensemble Stack (Intelligent Heterogeneous Stacked Voting Ensemble Model) for software defect prediction, integrating SMOTE to address class imbalance, advanced feature selection methods (MI, RFECV, PCA), and optimized hyperparameters across multiple classifiers. The model achieves an impressive 95% in accuracy, precision, recall, and F1-score on the PC1 dataset, along with an AUC-ROC of 93%, outperforming individual models such as Random Forest and Naive Bayes. Even on the more complex CM1 dataset, it maintains strong performance, with 85% accuracy and an AUC-ROC of 77%. This approach demonstrates superior class distribution handling and predictive stability, setting a new benchmark for software defect prediction by surpassing the limitations of single classifiers and showcasing the advantages of ensemble learning for managing imbalanced and complex data. However, an ensemble model trained on historical data may face challenges when adapting to unforeseen changes
in project dynamics or new development approaches. Since the model generates predictions based on past data patterns, its performance may degrade if the current task significantly diverges from the training data so this need to be handled in future. We will enhance our methodology by applying statistical tests like the t-test, Wilcoxon test, ANOVA, and Mann-Whitney U test to compare classifier performance. These tests ensure that performance differences are statistically significant and not due to random chance, providing a more robust and reliable evaluation. Additionally, future work will involve extending the approach to multiclass datasets to further validate the models across more complex classification tasks.

References

[1] D. Elreedy, A.F. Atiya, A comprehensive analysis of synthetic minority oversampling technique (smote) for handling class imbalance, Inf. Sci. 505 (2019) 32–64”

[2] L. Abdi, S. Hashemi, To combat multi-class imbalanced problems by means of over- sampling techniques, IEEE Trans. Knowl. Data Eng. 28 (1) (2016) 238–251

[3] J. Saez, B. Krawczyk, M. Wozniak, Analyzing the oversampling of different classes and types of examples in multi-class imbalanced datasets, Pattern Recognit. 57 (2016) 164–178.

[4] S.W. Purnami, R.K. Trapsilasiwi, SMOTE-least square support vector machine for classification of multiclass imbalanced data, Int. Conf. Mach. Learn. Comput. (2017) 107–111.

[5] Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., & Zhang, M. (2021). COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. Information and Software Technology, 129, 106432.

[6] M. Deng, Y. Guo, C. Wang, F. Wu, An oversampling method for multi-class imbalanced data based on composite weights, PLoS One 16 (11) (2020) e0259227

[7] Cai, X., Niu, Y., Geng, S., Zhang, J., Cui, Z., Li, J., & Chen, J. (2020). An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. Concurrency and Computation: Practice and Experience, 32(5), e5478.

[8] V. Bulavas, V. Marcinkevicius, J. Ruminski, Study of multi-class classification algorithms performance on highly imbalanced network intrusion datasets, Informatica 32 (3) (2021) 441–475.

[9] Kondo, Masanari, Cor-Paul Bezemer, Yasutaka Kamei, Ahmed E. Hassan, and Osamu Mizuno. "The impact of feature reduction techniques on defect prediction models." Empirical Software Engineering 24 (2019): 1925-1963.

[10] Dao, Fu-Ying, Hao Lv, Fang Wang, Chao-Qin Feng, Hui Ding, Wei Chen, and Hao Lin. "Identify origin of replication in Saccharomyces cerevisiae using two-step feature selection technique." Bioinformatics 35, no. 12 (2019): 2075-2083.

[11] Chen, Li-qiong, Can Wang, and Shi-long Song. "Software defect prediction based on nested-stacking and heterogeneous feature selection." Complex & Intelligent Systems 8, no. 4 (2022): 3333-3348.

[12] Rostami, Mehrdad, Kamal Berahmand, Elahe Nasiri, and Saman Forouzandeh. "Review of swarm intelligence-based feature selection methods." Engineering Applications of Artificial Intelligence 100 (2021): 104210.

[13] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, ''Software bug prediction using ML approach,'' Int. J. Adv. Comput. Sci. Appl., vol. 9, no. 2, pp. 78–83, 2018.

[14] X. Yang, D. Lo, X. Xia, and J. Sun, ''TLEL: A two-layer ensemble learning approach for just-in-time defect prediction,'' Inf. Softw. Technol., vol. 87, pp. 206–220, Jul. 2017.

[15] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia, ''A developer centered bug prediction model,'' IEEE Trans. Softw. Eng., vol. 44, no. 1, pp. 5–24, Jan. 2018.

[16] F. Khan, S. Kanwal, S. Alamri, and B. Mumtaz, ''Hyper-parameter optimization of classifiers, using an artificial immune network and its application to software bug prediction,'' IEEE Access, vol. 8, pp. 20954–20964, 2020.

[17] S. Qiu, L. Lu, S. Jiang, and Y. Guo, ''An investigation of imbalanced ensemble learning methods for cross-project defect prediction,'' Int. J. Pat- tern Recognit. Artif. Intell., vol. 33, no. 12, Nov. 2019, Art. no. 1959037.

[18] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, ''BPDET: An effective soft- ware bug prediction model using deep representation and ensemble learn- ing techniques,'' Expert Syst. Appl., vol. 144, Apr. 2020, Art. no. 113085.

[19] H. Osman, M. Ghafari and O. Nierstrasz, "Hyperparameter o ptimization to improve bug prediction accuracy," 2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), Klagenfurt, Austria, 2017, pp. 33-38, doi: 10.1109/MALTESQUE.2017.7882014.

[20] Nevendra, M., & Singh, P. (2022). Empirical investigation of hyperparameter optimization for software defect count prediction. Expert Systems with Applications, 191, 116217.

[21] Al-Fraihat, D., Sharrab, Y., Al-Ghuwairi, A. R., Alshishani, H., & Algarni, A. (2024). Hyperparameter Optimization for Software Bug Prediction Using Ensemble Learning. IEEE Access.

[22] M. Shepperd, Q. Song, Z. Sun, and C. Mair, ''Data quality: Some comments on the NASA software defect datasets,'' IEEE Trans. Softw. Eng., vol. 39, no. 9, pp. 1208–1215, Sep. 2013, doi: 10.1109/TSE.2013.11.

[23] J. Shi, X. Li, L. Li, C. Ouyang, and C. Xu, ''An efficient deep learningbased troposphere ZTD dataset generation method for massive GNSS CORS stations,'' IEEE Trans. Geosci. Remote Sens., 2023.

[24] W. Du, C. Wu, H. Yu, Q. Kong, Y. Xu, and W. Zhang, ''Determination of multicomponents in Rubi Fructus by near-infrared spectroscopy technique,'' Int. J. Anal. Chem., vol. 2023, pp. 1–9, Nov. 2023, doi: 10.1155/2023/5575944.

[25] Aftab, S. Abbas, T. M. Ghazal, M. Ahmad, H. A. Hamadi, C. Y. Yeun, and M. A. Khan, ''A cloud-based software defect prediction system using data and decision-level machine learning fusion,'' Mathematics, vol. 11, no. 3, p. 632, Jan. 2023, doi: 10.3390/math11030632

[26] Ali, M., Mazhar, T., Arif, Y., Al-Otaibi, S., Ghadi, Y. Y., Shahzad, T., ... & Hamam, H. (2024). Software defect prediction using an intelligent ensemble-based model. IEEE Access.

[27] Agrawalla, B., & Reddy, B. R. (2024). Software Fault Prediction Using Optimal Classifier Selection: An Ensemble Approach. Procedia Computer Science, 235, 2965-2974.

[28] Sharma, T., Jatain, A., Bhaskar, S., & Pabreja, K. (2024). Original Research Article An empirical analysis of feature selection techniques for Software Defect Prediction. Journal of Autonomous Intelligence, 7(3).