International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(17s) 2026 pp. 718-740
https://doi.org/10.70917/ijcisim-2026-4795
Article

An Explainable Feature Selection and Stacking Ensemble Framework for Software Fault Prediction

Harsimran Kaur[1], Hardeep Singh[2], Amitpal Singh Sohal[3], Parminder Kaur[4]

[1] Department of Computer Science, Guru Nanak Dev University, Amritsar, Punjab, India.
Email: harsimran.csrsh@gndu.ac.in
[2] Department of Computer Science, Guru Nanak Dev University, Amritsar, Punjab, India.
Email: hardeep.dcse@gndu.ac.in
[3] Department of Computer Engineering and Technology, Guru Nanak Dev University, Regional Campus, Gurdaspur, Punjab, India.
Email: amitpal.csegsp@gndu.ac.in
[4] Department of Computer Science, Guru Nanak Dev University, Amritsar, Punjab, India.
Email: parminder.dcse@gndu.ac.in
Corresponding Author: Harsimran Kaur

Abstract: The increasing complexity of contemporary software systems makes accurate and interpretable software fault prediction a challenging task. Although ensemble learning techniques have demonstrated promising predictive performance, existing approaches continue to face challenges related to feature selection, cross-project consistency, and model interpretability. This study proposes a Hybrid Boosted Stacking Ensemble Framework (HBSEF) for software fault prediction that integrates hybrid feature selection, ensemble learning, statistical validation, and SHAP-based interpretability. The proposed framework incorporates data preprocessing, class-imbalance handling using Random Oversampling, and a hybrid feature selection strategy integrating filter, embedded, and wrapper methods to identify software metrics with consistent predictive importance across multiple software projects. The core predictive model employs CatBoost, XGBoost, and LightGBM as base learners, with XGBoost serving as the meta-learner in the stacking ensemble. The framework is evaluated on an unseen test set using standard classification metrics, while paired t-testing is employed to assess the statistical significance of performance differences. Experimental results identify LOC, CBO, WMC, RFC, LCOM, and MAX_CC as the most influential software metrics. The proposed framework demonstrates statistically significant improvements over the baseline ensemble models, as validated through paired t-testing ($t = 4.53$, $p = 0.0004$), indicating the effectiveness of the proposed approach in enhancing software fault prediction performance. SHAP-based explanations provide both global and local interpretations, revealing the contribution of individual software metrics and base-model predictions to fault classification. Overall, the findings indicate that integrating principled feature selection with a boosting-based stacking ensemble can improve software fault prediction performance while providing greater transparency for software quality management.

Keywords: Feature Selection Methods (FSM), Software, Metrics, Software Fault Prediction, Stack, Ensemble Learning, SHAP, Explainable AI

1. Introduction

Testing is a critical part of the software development life cycle, ensuring high-quality software by evaluating all generated modules, like classes and components [1]. It is the most resource-intensive and expensive activity in the development process. A robust mechanism is essential to guarantee the high quality of the final product while optimizing resource usage in the testing process. Identifying fault-prone modules before testing serves as an effective solution to this challenge. Software Fault Prediction (SFP) models enable the automatic identification of fault-prone

modules before the testing phase. These models mainly rely on software metrics from previous releases and fault data gathered during earlier testing stages[2].

Fault prediction models are utilized to improve software quality and help in software inspection by pinpointing potential faults. The effectiveness of these models is determined by the modeling technique used and the software metrics implemented. Several software metrics have been proposed in existing research for SFP, and identifying the right set of metrics for an SFP is challenging [3]. Many studies in SFP have reported unusually high misclassification rates, which may be due to inherent issues in the project datasets [4]. One such issue is that the datasets often contain unnecessary information, requiring pre-processing before they can be used for fault prediction. Additionally, some metrics may be redundant or, in the worst case, have a confounding effect on the fault proneness results. Existing research has shown that a large number of features (attributes) can reduce classification accuracy and increase misclassification errors [5]. The use of High-dimensional data is challenging for many classification algorithms because of high computational cost and memory usage [6] [7]. One of the major challenges in high-dimensional datasets is the occurrence of redundant features, which have strong connections with other features and can be accurately predicted using current data. Such features provide minimal additional information to the fault prediction process and can affect model performance by extending training time and creating noise, which further leads to overfitting. These challenges not only reduce model efficiency but also compromise generalization capabilities. To address these challenges, feature selection techniques are frequently used to minimize dimensionality by selecting and keeping just the most relevant and non-redundant features [8].

Feature Selection (FS) approaches are used to detect and maintain the essential data elements(features), helpful in developing reliable prediction models. FS is critical in data pre-processing since it addresses the complexity of high-dimensional datasets. This challenge has two major issues: irrelevant variables, in which certain features (independent variables) do not influence the target variable (dependent variable), and redundant variables, in which independent variables are highly correlated and can be removed without loss of information [8]. There are four basic feature selection techniques: filter[9], wrapper[10], embedding [11], and hybrid[12]. Numerous studies[8][13][14][15][16][17][18] have used feature reduction approaches to enhance prediction models' performance but still suffer from some drawbacks i) limited number of classification techniques across a small set of dataset ii) lack of addressing both class imbalance and feature redundancy issues iii) lack of investigation into the impact of various feature selection methods (FSM) on the performance of ensemble classifiers iv) Limited evaluation of individual metric importance within the dataset v) Narrow selection of an ensemble learning classifiers.

To address the aforementioned shortcomings, this research work proposes a robust hybrid framework divided into three key phases: (i) addressing class imbalance to improve model reliability, (ii) assessing the significance of individual metrics using diverse feature selection techniques, and (iii) using the most relevant features to train and evaluate models across classification models using 10-fold cross validation employed to ensure robust and generalizable results. This technique ensures higher prediction accuracy, adaptability, and a better comprehension of feature contributions.

1.1 Contribution

This research provides the following key contributions:

- A novel formulation for metric selection is proposed to assess and rank software metrics in terms of their relevance across datasets and predictive importance, aiming at identifying robust fault-related software metrics.
- A hybrid feature selection framework combining the filter, embedded, and wrapper methods is proposed to improve the robustness of the chosen metrics. This is achieved by using the proposed framework on various PROMISE datasets.
- A boosted stacked ensemble learning architecture is proposed, in which XGBoost is used as a meta-learner to effectively integrate the results of various base models and improve the accuracy of fault prediction.

- A cross-dataset evaluation approach is followed to identify universally significant and consistent software metrics, and the results show that LOC, CBO, WMC, RFC, LCOM, and MAX_CC are stable predictors of software faults.
- Comprehensive experimental analysis is carried out along with statistical validation to confirm the effectiveness of the proposed framework by comparing it with individual ensemble models.
- Explainable AI(XAI) interpretation is incorporated using SHAP to provide global and local explanations to enable the transparent understanding of the impact of software metrics on the results of the predictions.

The research paper is organized to ensure that the research flows logically and is presented clearly. Section 2 reviews the relevant literature and highlights the existing research gaps. Section 3 presents the proposed framework and evaluation methodology and also describes the experimental settings. Section 4 reports the results, detailed analysis of the findings, offers an in-depth discussion, and presents a comparative analysis with previous studies. Section 5 addresses the threats to validity and outlines potential limitations of the study. Finally, Section 6 summarizes the important findings and suggests options for future research.

2. Related Work

Software fault prediction is significant in software engineering because it identifies defect-prone modules, hence increasing program reliability and maintainability. Traditional SFP models generally rely on a wide range of software measures, many of which are redundant or less useful. To increase model performance, several feature selection procedures are attempted to discover which metrics are most important. A diverse set of software measurements helps improve program quality by predicting faults. At the same time, numerous software measures are investigated to determine the most effective combination for predicting software quality. Control. I.H. Laradji et al. [19] demonstrated the efficacy of integrating feature selection and ensemble learning for software defect prediction. The authors cover important concerns such as feature repetition, excessive correlation, irrelevance, and data imbalance. Greedy forward selection (GFS) is more effective than Pearson's correlation at identifying relevant traits. The findings highlight the necessity of careful feature selection and robust ensemble models in enhancing defect detection and software quality control.

S. S. Rathore and S. Kumar [3] tested six fault prediction techniques: genetic programming, multilayer perceptron, linear regression, decision tree regression, zero-inflated Poisson regression, and negative binomial regression for predicting the number of faults in software modules rather than simply classifying them as faulty or non-faulty. The study examined 18 datasets from the PROMISE repository and evaluated model performance using criteria such as average absolute error, average relative error, completeness, and prediction at level *l*l. Statistical tests, such as the Kruskal-Wallis and Dunn's multiple comparison tests, were used to compare methodologies to improve software testing and dependability. The author also examined how the metrics RFC, NPM, DIT, CBO, MOA, and MFA continually surfaced as meaningful across a large number of datasets.

Raed Shatnawi [20] investigated software quality assessment by analyzing software metrics and their correlation with faults. The study indicates that, while there are several metrics, not all of them are highly associated with fault-proneness. In addition, uneven fault data challenge fault prediction and threshold identification. The authors used ROC analysis to accomplish three main goals: (1) establishing threshold values for risky modules, (2) assessing the impact of unbalanced data, and (3) applying ROC for feature selection. Their findings demonstrate that ROC may successfully define thresholds for four critical measures, including WMC, CBO, RFC, and LCOM. Furthermore, even after data sampling, the results are consistent, with ROC consistently selecting WMC, CBO, and RFC across most datasets, whereas other approaches showed considerable variability in metric choice. A similar type of study was reported by C. W. Yohannese and T. Li[21] to enhance Software Fault Prediction (SFP) by addressing key challenges such as feature redundancy, irrelevance, and class imbalance in software datasets. It introduces a new framework that integrates Feature Selection (FS) and Data Balancing (DB) techniques to improve classification performance. The study applies multiple feature selection methods and Synthetic Minority Oversampling Techniques (SMOTE) to refine

the dataset, ensuring that only the most relevant features contribute to model training. The empirical findings highlight that Random Forest (RF) with Information Gain (IG) feature selection achieves the highest ROC (0.993) for Static Code Metrics (SCM), while RF with Correlation-Based Feature Selection (CFS) attains the highest ROC (0.909) for Object-Oriented Metrics (OOM).

To improve Software Defect Prediction (SDP), L. Jia [23] suggested a Hybrid Feature Selection (HFS) technique to address the problem of redundant and unnecessary features in high-dimensional datasets. To find the most significant attributes, the approach employs a combination of ranking approaches such as Chi-Squared, Information Gain, and Pearson Correlation Coefficient. The chosen features are subsequently used to train an RF model for defect prediction. Experimental testing on five NASA datasets revealed that the suggested strategy efficiently minimizes feature redundancy and enhances predictive ability, as seen by improved F-measure scores. The paper identifies several significant benefits of feature selection, including reduced computational complexity, prevention of overfitting, improved generalization, and improved fault prediction accuracy. The feature subset is refined using feature selection methods such as varImp (), Boruta, and Recursive Feature Elimination (RFE). Experimental results show that RF-based feature selection greatly enhances classification performance, making it an effective approach for software defect prediction.

Early works are largely based on the use of traditional machine learning algorithms, including Decision Trees, Naïve Bayes, Support Vector Machines, and Logistic Regression classifiers based solely on static code metrics, though often limited by high dimensionality, imbalance, and dataset dependencies[22][23]. However, to address the limitations of high dimensionality and redundancy, SDP techniques are combined with feature selection methods to eliminate redundant code metrics, hence improving the accuracy of the classifiers[24]. However, the current trend of research has focused on the benefits of meta-heuristic approaches for code feature selection, which has yielded prominent advances when combined with ensemble classifiers for SDP, including within-project and cross-project defect prediction analysis [25][8] Ensemble methods like bagging, boosting, and stacking have often been able to outperform single classifiers due to the utilization of complementary strengths provided by multiple learners. Other works have extended the so-called deep learning models, such as multilayer perceptrons and autoencoder-based architectures, in capturing complex interactions among features. However, the black-box nature of most deep learning models raises several barriers to practical adoption. Recent literature has called for explainable AI methods, including SHAP and LIME, to enhance transparency and trust in SDP models; however, their integration into robust feature selection and ensemble or deep learning frameworks remains poorly explored, hence justifying further research in this direction.

2.1 Research Gap

Among the literature surveyed, there appear to be a number of limitations.

- Techniques for selecting features may fail to provide stability between datasets.
- Ensemble methods and deep learning networks value their performance over interpretability.
- XAI methods are not often incorporated within the SDP pipeline.
- Few studies capture class imbalance, consistency of features, and interpretability at the same time.

These gaps motivate the need for integrated frameworks that combine robust feature selection, advanced learning models, and explainable AI to improve both predictive performance and practical usability.

2.2 Agnostic Feature Selection Process

Fig. 1 depicts a systematic framework for feature optimization and generalization, which is intended to improve the performance and reliability of predictive models. The approach starts with Reducing Dimensionality, which removes superfluous and extraneous features, leaving a cleaner, more efficient dataset that increases model speed and accuracy. The following stage, Top-k Feature Selection, systematically finds the most influential features, allowing

the model to concentrate on the most useful variables and thereby enhancing interpretability and predictive strength. Moving ahead, the approach focuses on Universal Important Features, guaranteeing that the selected qualities are consistently relevant across numerous machine learning models. This not only improves the robustness of the strategy but also reduces model-specific bias.

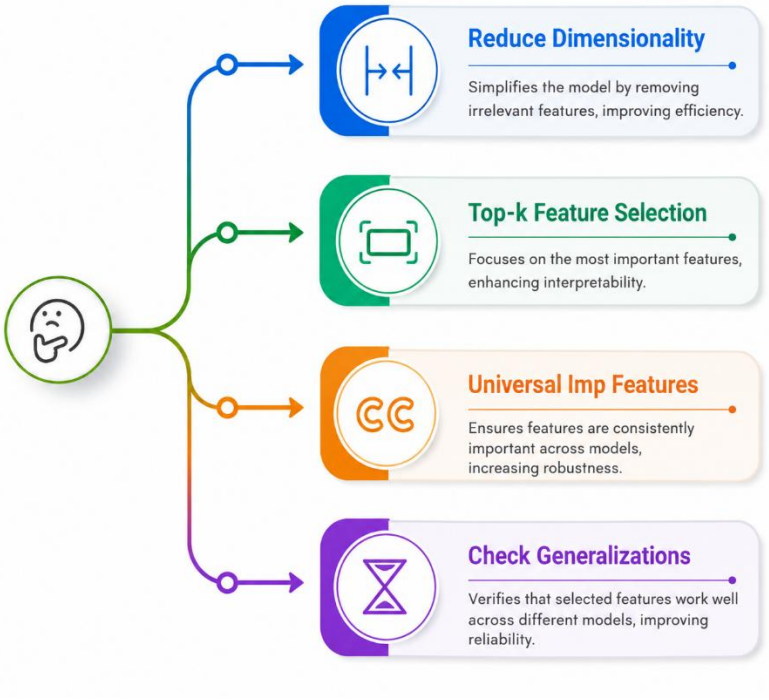

Fig. 1. Feature selection pipeline

Finally, the Check Generalizations phase validates the selected features across diverse models, confirming their reliability and stability in varying experimental conditions. Together, these stages form a powerful and cohesive pipeline that transforms raw features into a refined, interpretable, and generalizable feature subset, ensuring that the final model is both efficient and trustworthy, a key requirement for real-world predictive analytics and software fault prediction studies.

3. Methodology

This section outlines the research methodology adopted to achieve the objectives of the research work. It describes the data sources, pre-processing methodologies, feature selection methods, model implementation, and evaluation procedures utilized in the analysis.

3.1 Proposed Framework

The proposed framework is a three-stage Software Fault Prediction framework designed to improve prediction performance while maintaining statistical reliability and model interpretability. As shown in Fig.2, in the first stage, Data Pre-processing and Feature Engineering, ten PROMISE repository datasets are considered and divided into 80% training and 20% testing data. The training data is preprocessed through missing-value handling, outlier detection and treatment, normalization/standardization, and data cleaning. Feature selection is then performed only on the training data using filter methods (Chi-square and ANOVA), embedded ranking methods (Random Tree, Decision Tree, XGBoost, and Extra Tree), and the wrapper-based RFE method. The rankings obtained from these approaches are used to derive the final selected feature subset. In the second stage, Model Evaluation, Validation, and Statistical Analysis, Random Oversampling is applied to the training data to address class imbalance, followed by 10-fold stratified cross-validation. AdaBoost, CatBoost, LightGBM, and XGBoost are evaluated as candidate classifiers. The

selected base models, CatBoost, XGBoost, and LightGBM, are subsequently combined through a stacking ensemble, with XGBoost acting as the meta-learner to generate the final prediction. The ensemble is evaluated on the independent 20% test set using Accuracy, Precision, Recall, and F1-Score, and a paired t-test is used for statistical significance analysis. In the third stage, Model Interpretability, the importance of the selected software features is examined, while StackSHAP is employed at the meta-level to determine the contribution of each base-model prediction to the final stacking decision. Overall, the framework provides an end-to-end pipeline that combines data preprocessing, multi-method feature selection, class-imbalance handling, ensemble learning, independent evaluation, statistical validation, and explainability for reliable and interpretable Software Fault Prediction.

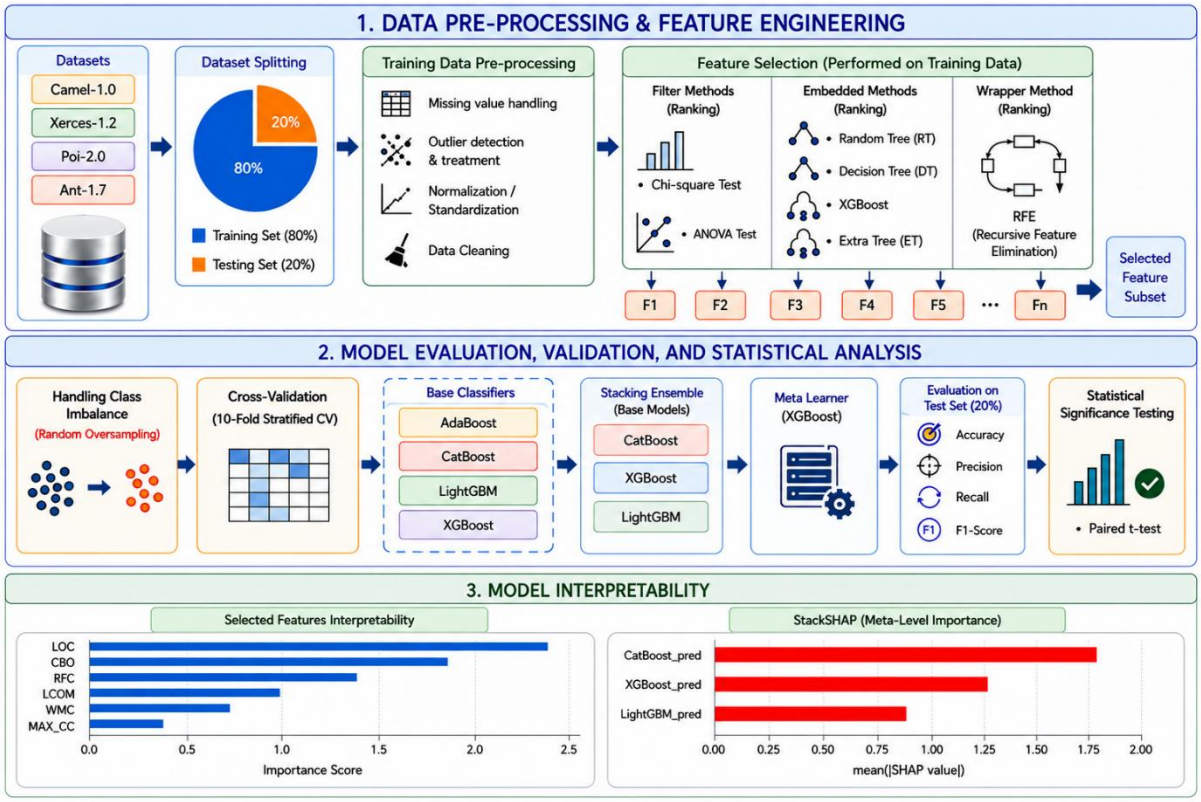

Fig. 2. The Proposed HBSEF Framework

3.2 Materials and Methods

This section describes the datasets, pre-processing steps, feature selection strategy, learning framework, evaluation protocol, and explainability techniques employed in this study.

3.2.1 Dataset

This research utilizes ten benchmark datasets from the PROMISE repository, namely Ant-1.7, JEdit-4.0, Log4j-1.1, Lucene-2.4, Ivy-2.0, Poi-2.0, Prop1, Camel-1.0, Xalan-2.4, and Xerces-1.2. Each dataset comprises 20 software metrics including Depth of Inheritance Tree (DIT), Weighted Methods per Class (WMC), Coupling Between Object Classes (CBO), Response for a Class (RFC), Lack of Cohesion of Methods (LCOM), Coupling Afferent (CA), Coupling Efferent (CE), Lack of Cohesion in Methods version 3 (LCOM3), Data Abstraction Metric (DAM), Number of Children (NOC), Method Fan-out Average (MFA), Method Overloading Average (MOA), Coupling Among Methods (CAM), Number of Public Methods (NPM), Inheritance Coupling (IC), Class Binary Metric (CBM), Afferent Method Count (AMC), Maximum Cyclomatic Complexity (MAX_CC), and Average Cyclomatic Complexity (AVG_CC). These metrics measure the structural, complexity, and coupling aspects of software modules that determine their fault-proneness.

Table 1: Summary of Software Fault Prediction Dataset

Dataset Name	Total Instances	Features	Faulty Instances	Non-Faulty Instances	Imbalance Ratio (Non-Faulty: Faulty)
Ant-1.7	579	21	166	413	2.49:1
Camel-1.0	326	21	13	313	24.08:1
Jedit-4.0	230	21	79	151	1.91:1
Log4j-1.1	272	21	34	238	7.00:1
Lucene-2.4	337	21	203	134	0.66:1
Ivy-2.0	312	21	16	296	18.50:1
Poi-2.0	277	21	141	136	0.96:1
Prop1	456	21	66	390	5.91:1
Xalan-2.4	613	21	110	503	4.57:1
Xerces-1.2	369	21	77	292	3.79:1

However, not all measurements make equivalent contributions to software fault prediction (SFP). A fundamental challenge associated with these datasets is the class imbalance problem, where the number of faulty and non-faulty instances differs considerably across datasets. As shown in Table 1, the degree of imbalance varies substantially: Camel-1.0 and Ivy-2.0 exhibit severe class imbalance, with imbalance ratios of 24.08:1 and 18.50:1, respectively, while Log4j-1.1 also shows considerable imbalance at 7.00:1. In contrast, Lucene-2.4 and Poi-2.0 have relatively balanced distributions, with ratios of 0.66:1 and 0.96:1, respectively. The remaining datasets, including Ant-1.7, Jedit-4.0, Prop1, and Xalan-2.4, show varying degrees of class imbalance. Such variations may bias model learning toward the majority class and consequently affect the reliable identification of faulty instances. To address this issue, Random Oversampling is applied to the training data to balance the class distribution, ensuring that both faulty and non-faulty instances receive adequate representation during model training.

3.2.2 Data imbalance

Data balancing involves undertaking changes in the class distribution in a given dataset to ensure there is an equal or balanced presentation of each class, which plays a fundamental role in eliminating the issues of classification bias in skewed datasets. Based on the literature, the simplest method to overcome issues of class imbalance in a classification problem would involve the use of the Random Oversampling method to create a balanced class presentation by replicating the instances of the minority class until a balanced presentation is achieved [16]. According to the research in [26], the use of random oversampling increases the classifier's sensitivity to the minority class. Based on the research, therefore, there are valid arguments to prove that despite the deficiency in the method of creating new data, the method remains very effective and easy to compute in terms of the pre-processing technique as a means of dealing with an imbalanced problem. Fig. 3. Illustration of oversampling, where instances belonging to the minority class are duplicated to equalize the class distribution in the original dataset.

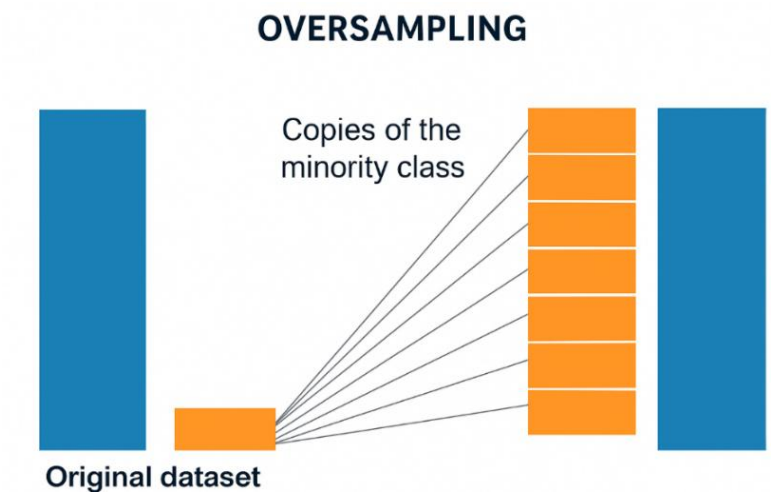

Fig. 3. Oversampling Process for Class Imbalance Handling

3.2.3 Hybrid Feature Selection Process

Feature Selection (FS) is an NP-hard optimization problem [27] that aims to identify the most useful and discriminative features from a dataset while maintaining its predictive integrity. By reducing redundant, noisy, or unnecessary features, FS not only improves classification accuracy and clustering quality but also considerably reduces computing complexity and overfitting risks. [28]. Therefore, researchers use various feature selection techniques to improve the outcomes and reduce computational time for large-scale challenges. Feature selection (FS) techniques typically accomplish two tasks: (i) identifying the smallest subset of features that effectively represents the original dataset, and (ii) evaluating the selected features using a predetermined fitness function.

To achieve these objectives, various FS methods adopt different evaluation strategies. Generally, feature selection is categorized into three major approaches: filter, wrapper, and embedded methods.

- Filtered methods are approaches that choose or extract relevant data, features, or information from a larger set based on predefined criteria or circumstances. These methods use a "filter" to eliminate undesired or insignificant aspects, leaving only the most important or valuable for further analysis or processing[27].
- Wrapper methods are feature selection techniques that employ a prediction model to assess the efficiency of various subsets of features. Each subset is evaluated by training and validating the model, and the subset with the best performance metric is chosen[28].
- Embedded methods are feature selection strategies that involve selecting significant features within the model training process. These methods combine the benefits of filter and wrapper approaches by selecting the most relevant features while developing the predictive model[12].

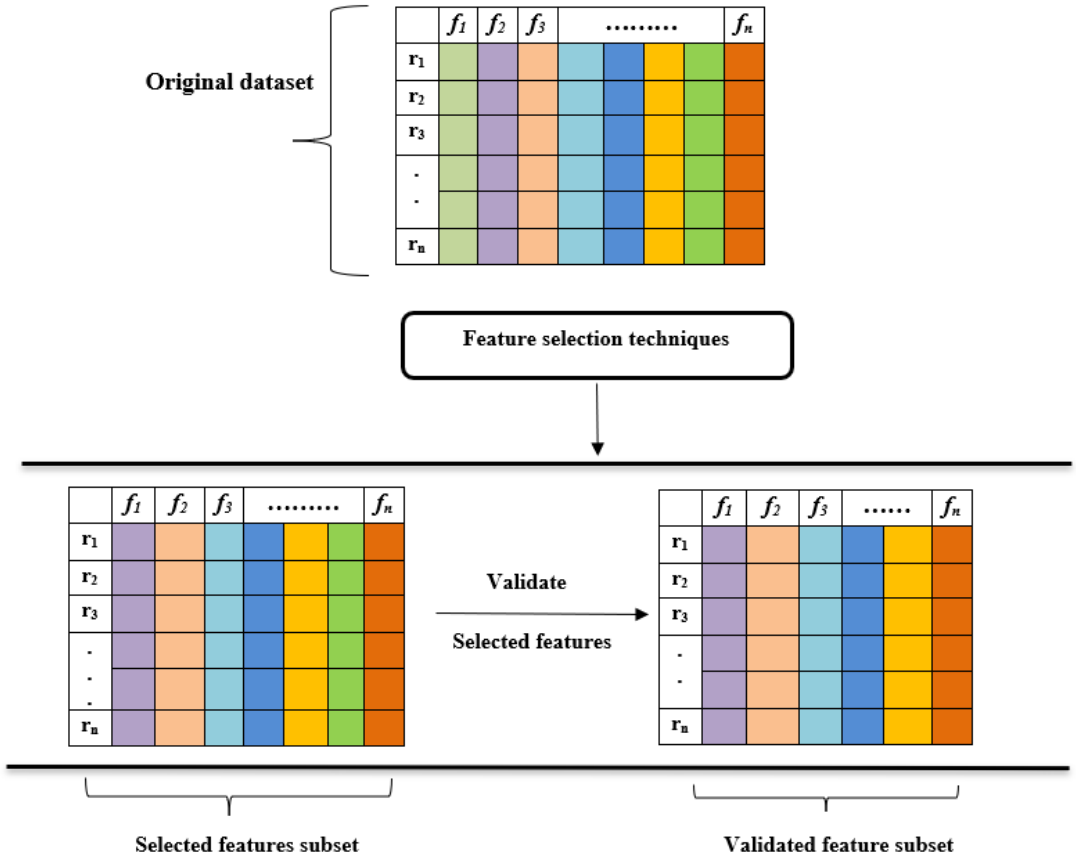

Fig. 4. A pictorial diagram of the Hybrid feature selection process

Filter methods assess the relevance of features by using a ranking procedure that subsequently removes low-ranking features. Several univariate measures like chi-square, ANOVA, information gain, coefficients, and correlation are used in the filter method[3]. Next, the wrapper technique is a greedy search approach that selects a subset of features by comparing a model's performance across different feature subsets. It assesses feature subsets using the performance of a particular machine-learning model. This means that the model used can have an impact on the features chosen. Recursive Feature Elimination (RFE) is a type of wrapper method, and it is used to find and choose the most important features [18]. Embedded methods combine the advantages of both filter and wrapper methods. In an embedded method, feature selection is integrated or built into the classifier algorithms. The goal of embedded methods is to reduce the amount of time that wrapper methods require to compute the reclassification of various subsets. Embedded methods include various types of methods like L1 Regularization (Lasso), decision tree, random forest, XGBoost, Extra Tree, and Elastic Net Regularization[29].

I.Tumar et al.[30] used a binary representation to address feature selection (FS) problems, where 0 indicates that a feature is not selected and 1 indicates that a feature is selected. However, this approach has some limitations. Specifically, the authors only determine whether a feature is selected or not, without evaluating the relative importance or ranking of each feature. In other words, they did not assess how important each metric is based on its contribution. Fig. 4 depicts a pictorial diagram for an original dataset of n features. Feature selection (FS) techniques are applied to select s features from the original set of n features, satisfying $s \leq n$. From these selected features, a subset of v−validated features ($v < s$) is further chosen based on their ranking scores. The desired result is achieved using the following formulas:

1. Calculate the frequency of selection (fi) for each feature across all datasets.

$$fi = \sum_{j=1}^{D} Sij \qquad (1)$$

- *fi* = frequency of selection of feature *i*
- S*ij* =1 if feature i is selected in dataset j, and S*ij* = 0 otherwise.
- D = Total number of datasets (D=10).

2. Calculate the percentage of selection for each feature:

$pi = (fi / D) \times 100$ (2)

Where: *pi* = percentage of times feature *i* is selected.

3. Select features that meet the 90% threshold

Feature i is selected if $pi >= 90\%$ (3)

3.2.4 Ensemble Methods

An algorithm that performs classification, especially in practical implementations, is commonly referred to as a classifier. In this research work, SFP models incorporated sixteen well-established classification algorithms from Extra Tree [31], XGBoost[32], Adaboost[33], CatBoost[34], LightGBM[35], and Random Forest[36]. Extra Trees improves prediction accuracy by generating a set of randomized decision trees. To improve performance, XGBoost combines scalable gradient boosting with advanced regularization algorithms. All of these algorithms are implemented in Jupyter Notebook, which enabled efficient testing and comparison of their performance in the context of software fault prediction.

- ***XGBoost***

XGBoost is an efficient and scalable gradient boosting method that generates decision trees in a sequential order, with each tree rectifying the faults of the preceding one. It uses regularisation (L1 and L2) to reduce overfitting, allows parallel and distributed computing for performance, and can automatically manage missing variables.XGBoost is commonly used for classification and regression problems due to its high accuracy, adaptability, and superior performance on large-scale and complicated datasets[32].

- ***AdaBoost***

AdaBoost is the best-known dependent algorithm for creating ensemble models. AdaBoost aims to identify misclassified instances when training a new inducer. Each instance in the training set is assigned a weight that determines its level of attentiveness. It addresses a variety of categorization issues, including multi-class settings. AdaBoost generates a strong classifier by minimizing errors in each iteration[33].

- ***LightGBM***

LightGBM (Ke et al., 2017) is a gradient-boosting framework created by Microsoft that is designed for efficiency and scalability. It employs a leaf-wise tree growth strategy with depth limitations, resulting in more accuracy than typical level-wise methods while preserving speed. LightGBM also supports approaches for efficiently handling large-scale, multidimensional data, such as Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). LightGBM is popular in classification, regression, and ranking problems due to its quick training speed, minimal memory utilization, and great performance[35].

- ***Cat Boost***

CatBoost (Dorogush, Ershov, & Gulin, 2018) is a gradient boosting technique created by Yandex to efficiently handle categorical information without considerable preprocessing. It employs ordered boosting and oblivious decision trees to minimize overfitting and improve generalization. CatBoost offers quick training, enables GPU acceleration, and requires little parameter adjustment, making it ideal for real-world applications. It is commonly used in classification, regression, and ranking applications, especially for datasets with a large number of categorical variables[34]

3.2.5 Hybrid Boosted Stacked Ensemble Model (HBSEF)

This approach combines both boosting and stacking techniques that leverage the complementary capabilities of different ensemble learning algorithms for software fault prediction. Through stacking a set of different boosted learning models, this approach captures complex defect data while reducing model bias and variance. This hybrid approach aims to improve robustness, increase generalization capabilities over different datasets, and provide highly accurate results for software quality evaluation through reliable fault prediction. The predictions of each base learner are stacked and fed to a higher-level layer in a stacked learning layer. The stacking allows the model to learn optimal combinations of base-level outputs without depending on the individual classifier or predictors. To improve predictive performance, boosting within the architecture of stacking is utilized to iteratively correct misclassified instances and improve discrimination on fault-prone modules.

3.2.6 STACKSHAP: Explainability for Boosted Stacked Ensembles

STACKSHAP has been proposed as an explanation framework to interpret the decision-making process for boosted stacked ensemble methods. In stacked architectures, existing methods for computing feature importance may fail to properly interpret the complex interactions between the base learners and the stacked ensemble. STACKSHAP takes advantage of the stacked ensemble architecture to properly apply the SHAP technique, which stands for Shapley Additive explanations.

In STACKSHAP, SHAP explanations are derived independently for individual base learners. The SHAP explanations are then aggregated and transferred to the stacked layer, where the contribution of each base learner's prediction to the final output is determined. By combining base learner SHAP explanations and stacked layer explanations, STACKSHAP delivers an explanation that describes both importance and interactions.

3.2.7 Performance measures

To assess the effectiveness of the proposed framework for software fault prediction, several performance measures are employed. These metrics quantitatively evaluate the predictive capability of the models in identifying faulty and non-faulty modules. Therefore, additional evaluation measures such as Precision, Recall, and F1-score are considered to provide a comprehensive understanding of model performance. The following section presents the definitions and formulas of the performance metrics used in the experimental analysis.

- **Accuracy:** Accuracy is one of the common measures of assessing the performance of an AI system on its predictions or classification results. Accuracy is calculated as the quotient of the correct predictions divided by the total number of predictions. Though accuracy works well in a class-balanced dataset, its use could be problematic in the case of datasets that are class-imbalanced.[37].

$$\textit{Accuracy} = \frac{TN+TP}{TN+TP+FN+FP} \qquad (4)$$

- **Precision and Recall**: Precision and Recall are significant evaluation measures for classification problems, particularly when there is a class imbalance. Precision is measured as the ratio of correctly predicted positive instances to all positive instances [38]. Recall is measured as the ratio of correctly predicted positive instances to all positive instances [38]. Precision and recall measure how well the model distinguishes positive instances from negative instances and suppresses false positives [39].

$$\textbf{\textit{Precision}} = \frac{TP}{TP+FP} \quad (5)$$

$$\textbf{\textit{Recall}} = \frac{TP}{TP+FN} \quad (6)$$

- **F1 Score:** It is a measure of a statistic that merges the values for precision and recall into a single score. It is the harmonic mean value between precision and recall, offering a well-rounded assessment criterion that takes into account both false positives and false negatives. The F1 measure is particularly helpful when dealing with an imbalanced data set, where both precision and recall are necessary [40].

$$\textbf{\textit{F1 Score}} = \beta * \frac{Precision * Recall}{Precision + Recall} \quad (7)$$

3.2.8 Parameters Configuration

Reproducibility is one of the basic principles of this study. To improve reproducibility, all aspects of the experiment pipeline, including pre-processing and the final model deployment, are governed by a set of parameters that are predefined and specified. This includes not only certain parameters of the experiment, such as the usage of Random Oversampling during the training of models and ensuring that the random state is preserved across all ensemble models, but also all aspects of the experiment have been described in Table 2.

Table 2: Experimental Setup and Parameter Configuration

Parameters	Specific settings
Data Pre-processing	Handling missing values, normalization/standardization, and categorical encoding
Class Balancing	Random Oversampling
Cross-Validation	Stratified K-Fold (n_splits=10, shuffle=True, random state=42)
Training–Testing Split	80% Training, 20% Testing (per fold)
Classifier Settings	random state=42, n_jobs=-1, class_weight=balanced
Ensemble Classifiers	XGBoost, AdaBoost, CatBoost, LightGBM
Base Classifiers	XGBoost, CatBoost, LightGBM
Meta learner	XGBoost
Evaluation Metrics	Accuracy, Precision, Recall, F1-score
Model Interpretability	StackSHAP

3.3 Data validation

In this case, the Stratified K-Fold cross-validation technique (with K=10) has been employed to thoroughly assess the performance of the models on an unseen dataset. Although randomly splitting the entire dataset into the training and testing datasets is common and accepted best practice, it often leads to biased results, particularly when the dataset is small or unbalanced. Consequently, the application of the K-fold cross-validation technique has been considered, which represents the best technique to effectively obtain reliable performance estimates [38]. This technique allows the dataset to be split evenly into K folders, and K-1 of the folders are used for the training dataset, with the lone folder serving as the testing dataset. The technique has been repeated K times to ensure that every dataset instance has been tested only once. The technique of stratification has been employed to preserve the original representation of the number of faulty and non-faulty modules in the 10 folders to mitigate the issue of class imbalance and make the entire testing process of the model more efficient and effective[39].

4. Results and Discussion

The selection of relevant software metrics helps in improving the prediction capabilities of the models in software development. The selection of the relevant metrics for the prediction task helps in developing the models. In addition, the availability and quality of data related to the metrics also affect the performance of the models. To determine the relevant metrics for the models, different techniques have been employed, and these are presented in Fig. 2. The main goal of the research study is to find the software metrics that have the largest effect on the software fault prediction task. Software metrics such as WMC, MOA, CAM, NPM, IC, CBM, AMC, MAX_CC, NOC, CBO, RFC, LCOM, CA, CE, LCOM3, LOC, DAM, MFA, and AVG_CC are considered in terms of the importance to the software metrics in the ten different datasets including Ant-1.7, Camel-1.0, Jedit-4.0, Log4j-1.1, Lucene-2.4, Ivy-2.0, Poi-2.0, Prop1, Xalan-2.4, and Xerces-1.2 from the PROMISE repository. To make the study credible and reliable in terms of the metrics' performance, the first five metrics in terms of the rank scores in each technique have been considered.

From Table 3, the Chi-Square test from the camel-1.0 dataset revealed that CBO (710.4), LCOM (1049.6), NOC (150.40), LOC (448.83), and CA (1217.3) had high scores, thus showing strong relations with fault-prone modules. These are important in assessing code complexity and coupling criteria. ANOVA, which measures variation within groups, identified CBO with (29.117), WMC with (8.115), NPM with (8.044), CA with (26.74), and NOC with (12.92) as influential metrics. Using Random Forest, which measures variation within groups, CBO is (0.1154), CA (0.1173), CAM (0.08381), LOC (0.07651), and LCOM (0.06625) are the most influential metrics. NOC is (0.2083), followed by CA (0.1510), MFA (0.1671), CAM (0.1455), and MAX_CC (0.1263) by the Decision Tree Classifier. Extra Tree Classifier identifies CA (0.1183), CBO (0.1012), NOC (0.0904), LCOM (0.0586), and NPM (0.0647). On all occasions, this approach identifies metrics explaining the code dimension, coupling, and cohesion. Boost identifies NOC (0.1243), CBO (0.1301), LCOM (0.0785), NPM 01128, and AVG_CC 0.0804. Lastly, Recursive Feature Elimination (RFE) ranks the selection criteria using their ranks, and the selection criteria selected based on their ranks include RFC (rank 1), LOC (rank 1), AMC (rank 1), DAM (rank 1), and MFA (rank 1), which all belong to the top-ranked criteria. The final column represents how often each selection criterion is selected by the implemented feature selection methods for the Camel-1.0 dataset. From the study, it is observed that WMC and MAX_CC are selected by 1 method, NOC by 6 methods, CBO and LCOM by 5 methods, CA by 6 methods, LOC, CAM, AVG_CC, and MFA by 2 methods, and NPM by 3 methods. Since the selected criteria for the study demonstrate persistence and stability in their selection across various feature selection methods, they would be selected for further study, as they are also relevant for software fault prediction. Additionally, criteria such as DIT, RFC, CE, LCOM3, DAM, MOA, IC, CBM, and AMC, which all have zero frequency, would not be selected by any feature selection method and would thus not be selected for consideration in this research work. A similar set of analyses is performed on the rest of the datasets: Ant-1.7, Jedit-4.0, Log4j-1.1, Lucene-2.4, Ivy-2.0, Poi-2.0, Prop1, Xalan-2.4, and Xerces-1.2. Only the results on Camel-1.0 will be presented here for brevity.

Table 3. Analysing Metric Relevance through Feature Selection Methods on the Camel-1.0 Dataset

Methods / **Metrics**	**Chi-Square**	**ANOVA**	**RT**	**DT**	**ET**	**XGBoost**	**RFE**	**Frequency of metrics selected by methods**
DIT	1.488	2.013	0.01955	0.000	0.0357	0.0351	15	0
WMC	67.63	**8.115**	0.04813	0.000	0.0495	0.0052	16	**1**

NOC	**150.40**	**12.92**	0.06525	**0.2083**	**0.0904**	**0.1243**	**1**	6
CBO	**710.4**	**29.117**	**0.11546**	0.0505	**0.1012**	**0.1301**	14	5
RFC	48.80	2.675	0.05926	0.000	0.0543	0.0277	12	**0**
LCOM	**1049.6**	2.200	**0.06625**	0.0585	**0.0586**	**0.0785**	**1**	**5**
CA	**1217.3**	**26.74**	**0.1173**	**0.1510**	**0.1183**	0.0401	**1**	6
CE	3.664	0.718	0.04863	0.000	0.0448	0.0227	10	**0**
LCOM3	1.029	1.991	0.04352	0.000	0.0369	0.0461	8	**0**
LOC	**448.83**	2.615	**0.07651**	0.000	0.0527	0.0647	11	**2**
DAM	0.2243	0.6822	0.01550	0.0215	0.0177	0.0000	4	0
MFA	2.051	5.471	0.02510	**0.1671**	0.0276	0.07038	**1**	**2**
MOA	1.849	1.091	0.03142	0.000	0.03417	0.0043	13	0
CAM	0.3073	2.427	**0.08381**	**0.1455**	0.0499	0.0435	2	**2**
NPM	65.52	**8.044**	0.03974	0.000	**0.0647**	**0.1128**	9	**3**
IC	0.657	0.8417	0.00426	0.000	0.0148	0.0000	7	0
CBM	1.651	0.885	0.008654	0.000	0.0113	0.0000	6	**0**
AMC	8.057	0.831	0.05286	0.000	0.0403	0.0570	5	**0**
MAX_CC	0.6417	0.2956	0.03075	**0.1263**	0.0495	0.0566	3	**1**
AVG_CC	0.510	1.203	0.04795	0.0710	0.0461	**0.0804**	**1**	**2**

The highlighted numbers indicate selected metrics (IMPORTANT), and the non-highlighted depict non-selected metrics (UNIMPORTANT) based on their importance. (Threshold _value=5)

Table 4 illustrates the datasets selected for this research work. These datasets are pre-processed and analyzed with the help of FSMs to identify the most relevant software metrics. These metrics significantly contribute to the SFP accuracy. The summary of selected metrics for each dataset is as follows: For Ant-1.7, 12 metrics are chosen, including WMC, CBO, RFC, LCOM3, LOC, and others such as DAM, MFA, and NPM. MAX_CC AND AVG_CC are also included due to their significance in predicting fault-prone modules. The Camel-1.0 dataset included 11 selected metrics, such as WMC, NOC, CBO, LCOM, and CA. Metrics like MFA, Cohesion Among Methods, and cyclomatic complexity measures MAX_CC AND AVG_CC are also prioritized. For Jedit-4.0, there are 15 important metrics: WMC, NOC, RFC, LCOM, CA, LCOM3, and LOC. There are other metrics such as AMC, MOA, CAM, Inheritance

IC, and CBM. The cyclomatic complexity MAX_CC is also chosen as a metric. For Log4j-1.1, there are 10 important metrics: LOC, WMC, CBO, RFC, LCOM, and CE. There are other metrics such as MOA, NPM, MAX_CC, AND AVG_CC. There are 14 metrics for the dataset of Lucene-2.4. These are DIT, CBO, RFC, LCOM, CA, CE, AMC, and LOC. There are other metrics such as CAM, NPM, IC, CBM, MAX_CC, and AVG_CC. For Ivy-2.0, there are 14 important metrics: DIT, CBO, RFC, LCOM, CA, CE, LCOM3, AND LOC. There are other metrics such as CAM, AMC, NPM, IC, and MAX_CC.

The POI-2.0 dataset had 12 metrics, including AMC, WMC, CBO, RFC, LCOM, CE, LCOM3, and LOC. Metrics such as MFA, IC, AVG_CC, and MAX_CC are also considered. For Prop1, a total of 13 metrics are considered, including DIT, WMC, CBO, RFC, LCOM, CE, and LOC. Metrics such as AMC, MFA, IC, CBM, and MAX_CC & AVG_CC are also given high priority. The Xalan-2.4 data set had a total of 12 metrics, including WMC, CBO, RFC, LCOM, AMC, CE, LCOM3, and LOC. Metrics such as CAM, IC, MAX_CC, and AVG_CC are also considered. Finally, for Xerces-1.2, a total of 15 metrics are considered, including WMC, AMC, NOC, RFC, CA, LCOM3, and LOC. Metrics such as DAM, MFA, CAM, IC, CBM, CBO, DIT, and AVG_CC are also considered.

Table 4. Important Metrics Selected by Feature Selection Methods

Dataset	Selected metrics	Selected by the FSM method are important
Ant-1.7	WMC, CBO, RFC, LCOM3, LOC, DAM, MFA, NPM, AMC, MAX_CC, AVG_CC, LCOM	12
Camel-1.0	WMC, NOC, CBO, LCOM, CA, LOC, MFA, CAM, NPM, MAX_CC, AVG_CC	11
Jedit-4.0	WMC, NOC, RFC, LCOM, CA, LCOM3, LOC, DAM, MFA, MOA, CAM, IC, CBM, AMC, MAX_CC	15
Log4j-1.1	WMC, CBO, RFC.LCOM, CE, LOC, MOA, NPM, MAX_CC, AVG_CC	10
Lucene-2.4	DIT, CBO, RFC, LCOM, CA, CE, LOC, CAM, NPM, IC, CBM, AMC, MAX_CC, AVG_CC	14
Ivy-2.0	DIT, CBO, RFC, LCOM, CA, CE, LCOM3, LOC, CAM, NPM, IC, AMC, WMC, MAX_CC	14
Poi-2.0	WMC, CBO, RFC, LCOM, CE, LCOM3, LOC, MFA, IC, AMC, AVG_CC, MAX_CC	12
Prop1	DIT, WMC, CBO, RFC, LCOM, CE, LOC, MFA, IC, CBM, AMC, MAX_CC, AVG_CC	13
Xalan-2.4	WMC, CBO, RFC, LCOM, CE, LCOM3, LOC, CAM, IC, AMC, MAX_CC, AVG_CC	12

Xerces-1.2	WMC, NOC, RFC, CA, LCOM3, LOC, DAM, MFA, CAM, IC, CBM, AMC, AVG_CC, DIT, CBO	15

As presented in Table 5, it analyses the different software metrics on ten different software systems: Ant-1.7, Camel-1.0, Jedit-4.0, Log4j-1.1, Lucene-2.4, Ivy-2.0, Poi-2.0, Prop1, Xalan-2.4, and Xerces-1.2. Each metric is identified by a checkbox (□) and an (✗) to identify whether it is suitable and not suitable for each software system. The "Count" column expresses the number of software systems for which each metric is suitable, and the "Selection%" column expresses the percentage for which each metric is suitable. As presented in the table, it provides the selection rate of the most important software metrics on the ten datasets of the PROMISE repository, highlighting the significant role they have in software defect prediction problems. Lines of Code (LOC) is the most frequently selected metric on all ten datasets (100%), which expresses its universally significant role in predicting software defects. Other significant measures are Weighted Methods per Class (WMC), Coupling Between Objects (CBO), Response for a Class (RFC), Lack of Cohesion in Methods (LCOM), and the Maximum Cyclomatic Complexity (MAX_CC) because they are presented in 9 out of the 10 datasets (90%). These metrics are closely associated with software faults and are important for training models. The Efferent Couplings (CE), Lack of Cohesion in Methods (LCOM3), Measure of Functional Abstraction (MFA), and Cohesion Among Methods (CAM) are moderately chosen metrics, which appear in 60% of the datasets. The other metrics are Inheritance Coupling (IC) and Number of Public Methods (NPM), showing that they have a moderate influence on fault prediction; they are selected in 70% and 50% of the datasets, respectively. On the other hand, the remaining metrics are less chosen: the Measure of Aggregation (MOA) is chosen in just two datasets (20%), while the Number of Children (NOC) and Data Access Metric (DAM) appeared in 30% of the datasets. It seems that these measures have less impact than the generally used ones. In brief, the table provides significant information regarding the relative importance of different software metrics to predict bugs, which helps researchers and practitioners concentrate on the most significant phase related to training the model and the improvement of software quality. Based on the above study, the metrics that lie within the selection range of (90%-100%) represent LOC (100%), and CBO, WMC, RFC, LCOM, and MAX_CC with 90%, which indicate the significant importance of the respective metrics.

Table 5. Feature Selection Across Multiple Datasets

Dataset Metrics	Ant -1.7	Camel -1.0	Jedit -4.0	Log4j -1.1	Lucene-2.4	Ivy - 2.0	Poi - 2.0	Prop 1	Xalan -2.4	Xerces-1.2	Count	Selection %
DIT	✗	✗	✗	✗	□	□	✗	□	✗	□	4	40%
WMC	□	□	□	□	✗	□	□	□	□	□	**9**	**90%**
NOC	✗	□	□	✗	✗	✗	✗	✗	✗	□	3	30%
CBO	□	□	✗	□	□	□	□	□	□	□	**9**	**90%**
RFC	□	✗	□	□	□	□	□	□	□	□	**9**	**90%**
LCOM	□	□	□	□	□	□	□	□	□	✗	**9**	**90%**
CA	✗	□	□	✗	□	□	✗	✗	✗	□	5	50%
CE	✗	✗	✗	□	□	□	□	□	□	✗	6	60%

LCOM3	□	✗	□	✗	✗	□	□	✗	□	□	6	60%
LOC	□	□	□	□	□	□	□	□	□	□	**10**	**100%**
DAM	□	✗	□	✗	✗	✗	✗	✗	✗	□	3	30%
MFA	□	□	□	✗	✗	✗	□	□	✗	□	6	60%
MOA	✗	✗	□	□	✗	✗	✗	✗	✗	✗	2	20%
CAM	✗	□	□	✗	□	□	✗	✗	□	□	6	60%
NPM	□	□	✗	□	□	□	✗	✗	✗	✗	7	50%
IC	✗	✗	□	✗	□	□	□	□	□	□	7	70%
CBM	✗	✗	□	✗	□	✗	✗	□	✗	□	5	40%
AMC	□	✗	□	✗	□	□	□	□	□	□	8	80%
MAX_CC	□	□	□	□	□	□	□	□	□	✗	**9**	**90%**
AVG_CC	□	□	✗	□	□	✗	□	□	□	□	8	80%

As the research moves to the next stage, there will be an emphasis on the training of models based on the selected metrics LOC, CBO, WMC, RFC, LCOM, and MAX_CC. The metrics will be applied due to the high frequency of their selection among different methods of feature selection, symbolizing the impact they have on software failure prediction. These metrics represent different aspects of software architecture and complexity. They will therefore enable the identification of possibly defective modules. For instance, LOC will measure the size of the program. Other metrics, such as CBO and WMC, will reveal the dependence of the classes. Additionally, RFC will reveal the reactivity of the classes to external events. On the other hand, the cohesiveness of the methods in the class will be measured by the metrics LCOM and MAX_CC, which represent cyclomatic complexity and evaluate a class's logical complexity. These metrics are used as input features in a variety of ensemble classifiers such as XGBoost, AdaBoost, LightGBM, and CatBoost. The performance of these models is evaluated in the form of Accuracy, recall, precision, and F1 score. Then we evaluate the mean performance percentage of each ensemble model in the individual ensemble models; the highest value for accuracy (93.93%) is observed for XGBoost, followed by LightGBM with 93.49%, and CatBoost with 93.25%, which reflects the best predictive power, while the lowest accuracy of 78.66% is observed for AdaBoost due to the model's vulnerability to noise and imbalanced samples as shown in Table 6. The performance comparison of the classifiers is visually represented in Fig. 5. For further improvement, the models having the highest performance (%) are chosen as base models, namely LightGBM, XGBoost, and CatBoost, for the stacking hybrid ensemble technique, and XGBoost is chosen as the meta-learning algorithm due to its robustness to noisy data, regularization property, and efficiency for processing non-linear relationships.

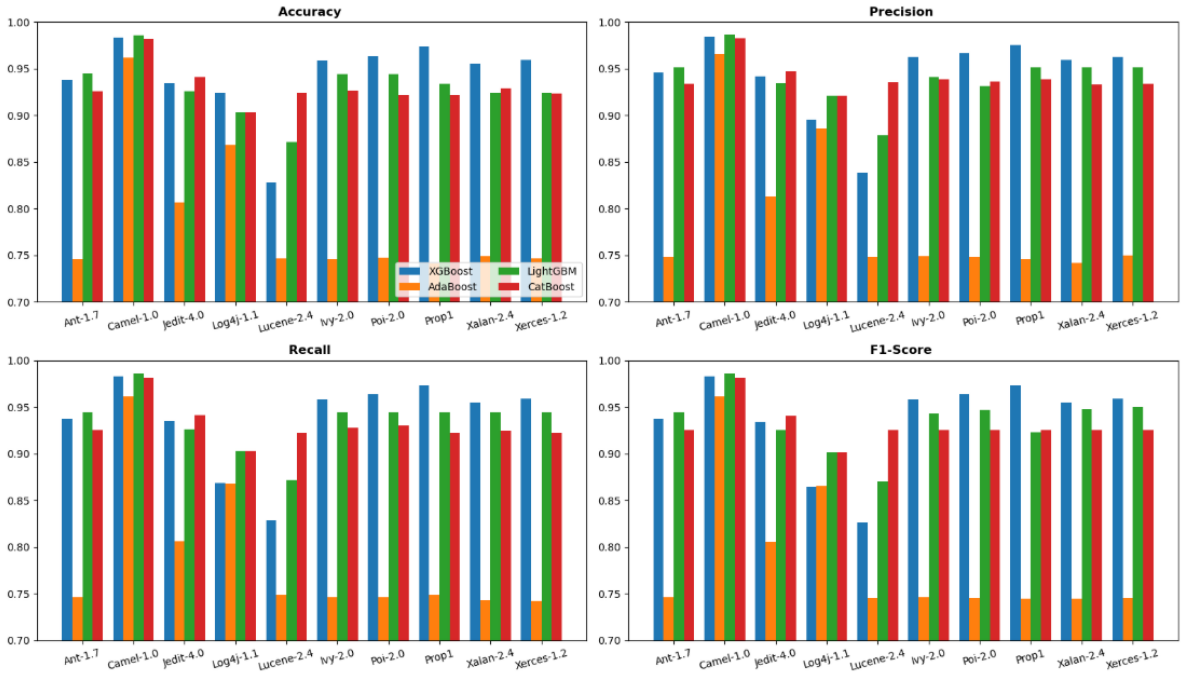

Fig. 5. Performance comparison of ensemble classifiers across benchmark datasets

Table 6. Comparative Evaluation of Classifiers for SFP

Ensemble Classifiers	Performance percentage (%)
AdaBoost	78.66%
CatBoost	**93.25%**
LightGBM	**93.49%**
XGBoost	**93.93%**

4.1 Paired t-test

To further confirm the efficacy of the proposed Boosted Stacked Ensemble model, a paired t-test is performed on the proposed technique against the best ensemble-based models on all datasets as shown in Table 7. The result of the statistical analysis shows a t-value of 4.53 and a corresponding p-value of 0.0004, which is significantly smaller compared to the significance level of $\alpha = 0.05$.

Table 7. Statistical significance analysis of ensemble and Boosted Stacked Ensemble models

Comparison	T-statistic	P-value	Cohen's d	95% CI	Significant ($\alpha = 0.05$)	Interpretation
Base Ensemble model vs Stacked Ensemble	4.53	0.0004	1.71	[0.015,0.052]	Yes	Stacking Hybrid Ensemble performs well.

Additionally, its large effect size is evident from its high value of Cohen's d of 1.71. The 95% confidence interval [0.015, 0.052] of the experiment is within a safe zone and does not contain a value of 0, and so on. The result of this experiment proves the proposed technique to be significantly better than traditional ensemble techniques and is a scalable and efficient technique for software fault prediction.

4.2 Model Explainability

Finally, Fig. 6 shows the importance of the parameters after the combination of the base models using the meta-learned approach, where LOC, followed by the importance of CBO and RFC parameters, plays a significant role, whereas LCOM, WMC, and MAX_CC have the lowest importance.

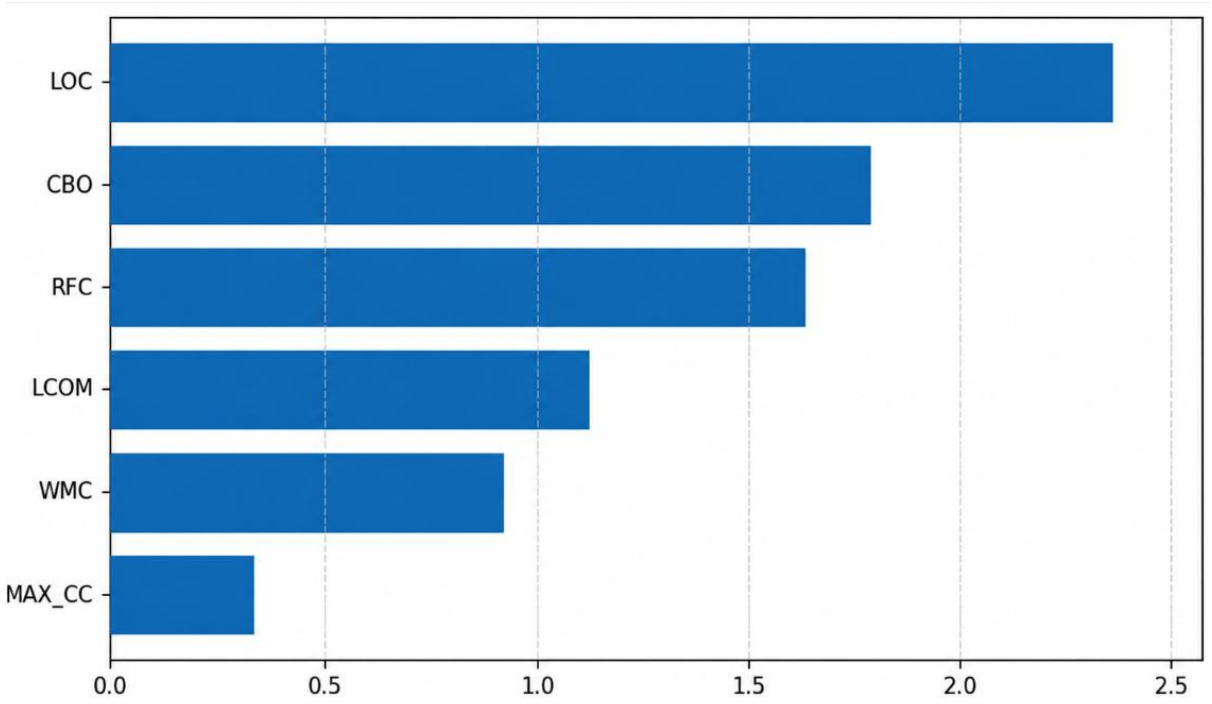

Fig. 6. Effective feature contribution of validated feature using SHAP

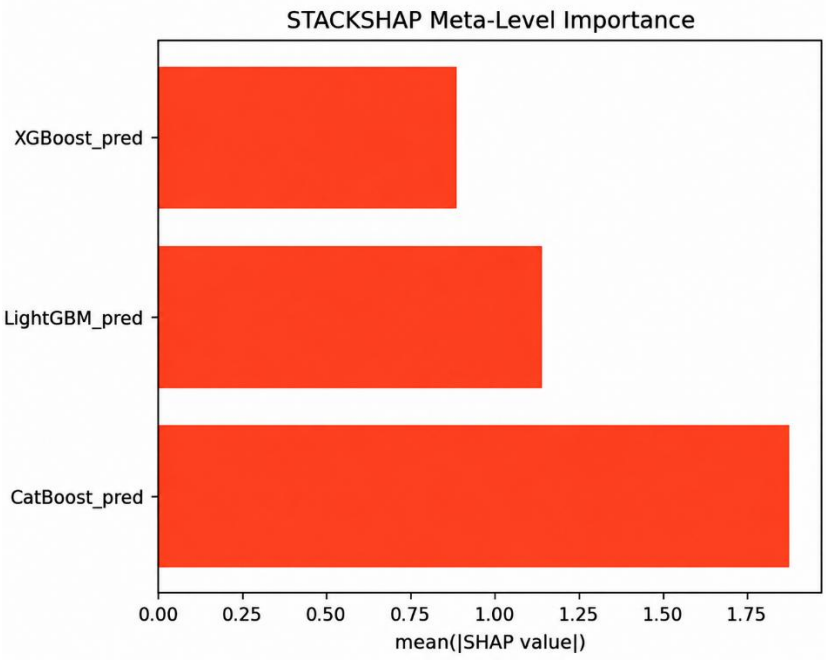

Fig. 7. Mean absolute SHAP values of the stacked model.

Fig. 7 represents the comparison between base-level and meta-level importance carried out by the STACKSHAP technique on the boosted stacked model. The base-level analysis indicates that the three ensemble models are important contributors to the prediction task, with LightGBM having the highest average SHAP importance value, followed by CatBoost and then XGBoost. On the meta-level analysis, it is found that CatBoost_pred has the greatest contribution towards making the prediction, followed by LightGBM_pred, with a relatively lower contribution by XGBoost_pred. The relative importance difference between the meta and base levels indicates that the stacked model has the flexibility to assign weights to the predictions made by the base learners instead of giving importance solely to the models with good prediction outcomes. The STACKSHAP analysis has been able to establish the dependencies among models and has explained how the base predictions are used by the stacked models with diverse learning capabilities to generate improved fault prediction results.

4.3 A Comparative Evaluation Against Existing Studies

Table 8 offers a comparison between the proposed Boosted Stacked Hybrid Ensemble Model (HBSEF) and existing state-of-the-art research works on software defect predictions. Existing research works mostly concentrate on stacking ensemble models or on boosting models, but lack integration of multi-feature selection and are not based on explainable AI. For example, Chen et al. (2022) use nested stacking models along with heterogeneous feature selections but lack the incorporation of interpretability techniques, while Alazba and Aljamaan (2022) use tree-based stacking models based on optimization but lack feature selection. Similar research works from other authors, such as S. P. Sahu et al. (2021) and H.Wang et al. (2018), lack comprehensive feature selection techniques and hybrid ensemble models. In fact, the proposed HBSEF offers its innovations in combining multi-features, hybrid boosting and stacking models, and STACKSHAP-based techniques for explainability. This comprehensive design enables HBSEF to achieve the highest predictive performance, with an accuracy of 0.988, outperforming all compared approaches. The comparison shows that most of the existing studies have focused on ensemble learning techniques without considering the cross-dataset consistency of software metrics and interpretability of models. The proposed HBSEF framework considers the cross-dataset consistency of software metrics and interpretability of models, thus improving the predictive capabilities of the models.

Table 8. Comparison of Existing Feature Selection Approaches with the Proposed Framework

Study	Core Methodology	Cross-Dataset Metric Consistency	Explainable AI	Ensemble Strategy	Best Accuracy
Chen et al.[40]	Nested-Stacking with MLP meta-classifier	No	No	Stacking	~0.92–0.95
Alazba & Aljamaan [41]	Optimized Tree-Based Stacking	No	No	Stacking	~0.93–0.96
S. P. Sahu et al. [42]	Hybrid Ensemble Learning	No	No	Bagging / Boosting	~0.91
H.Wang et al. [15]	Feature Selection Stability Analysis	No	No	No	~0.88–0.92
The Proposed study	**Boosted Stacked Ensemble**	**Yes**	**Yes**	**Yes (Boosting + Stacking)**	**0.988**

4.4 Discussion

The results of the above study make it clear that the proposed Boosted Stacked Hybrid Ensemble Model (HBSEF) outperforms the existing approaches remarkably well in software fault prediction. The proposed approach comprehensively uses the strengths of multiple gradient boost learners combined in a stacked manner. Thus, the approach efficiently identifies the highly non-linear patterns simultaneously associated with software features that usually remain unnoticed by individual classifiers. The robust performance of the proposed approach over all datasets manifests the outstanding generalization ability of the approach. The most important thing that has been noticed is that the gradient boost learning-based ensembles outperform the traditional ensemble models. The proposed stacking approach further boosts the advantages of the above aspect by comprehensively using the decision boundaries of different classifiers that are likely to be complementary. The result of the paired t-test, plus the large effect size, makes the proposed approach more trustworthy by minimizing the possibility of observing a significant improvement due to randomness. In conclusion, from the above discussion, it is clear that the synergistic integration of hybrid feature selection methods, boosted stacking, statistical validation, and explainable AI results in a robust, accurate, and interpretable fault prediction system. It can be stated that the proposed method represents an advancement of state-of-the-practice because it deals with some of the most critical issues, such as model instability, redundancy, and a lack of interpretability, prevailing in software defect prediction techniques.

5. Threats to validity

External validity: External validity is the measure by which the applicability of the result obtained from the experiment to other datasets can be ascertained. Even though the PROMISE dataset is very diversified, it may not be possible to understand the exact nature and depth of the projects that can be measured using industrial standards. In order to overcome such difficulties, multiple datasets belonging to various domains, application sizes, and distributions, which can contribute towards bias, are considered for evaluating the new method. Hence, multiple dataset evaluation increases the validity of the method, making it applicable to similar projects. However, validation is recommended on industrial projects for ascertaining the exactness of the method.

Internal validity: Internal validity is the extent to which the results are affected only by the variables and not by any extraneous factors. One of the most significant internal threats to fault prediction problems is imbalance among the classes because the number of samples not classified as faulty often outnumbers those that are faulty. This often results in models biased towards the majority class because the model is not able to capture as many of the actual faulty examples. To address these problems, oversampling techniques are employed by the research study to balance the results among all the classes. Additionally, a 10-fold cross-validation approach is applied throughout all the experiments conducted to guarantee the validity of the results.

Construct validity: A possible threat to the construct validity of the study could be the use of software measures such as LOC, RFC, LCOM, WMC, CBO, and MAX_CC, which may not be able to account for every different aspect of the software. To tackle this, the study makes use of a combination of filter methods (ANOVA, Chi-Squared), embedded methods (Random Forest, XGBoost, Extra Trees), and wrapper methods (Recursive Feature Elimination) to make sure that only the most important features are selected, which will further help make sure of the construct validity of the study and will not be biased.

6. Conclusion and Future Work

This study presented an experimentally validated framework for identifying important software metrics for software fault prediction across diverse benchmark datasets. A hybrid feature-selection strategy integrating filter, embedded, and wrapper methods identified six important software metrics: Lines of Code (LOC), Coupling Between Objects (CBO), Response for a Class (RFC), Lack of Cohesion of Methods (LCOM), Weighted Methods per Class (WMC), and Maximum Cyclomatic Complexity (MAX_CC). These metrics represent key dimensions of software size, coupling, cohesion, and complexity. The selected features were used following an 80:20 training-testing split, with class imbalance addressed through random oversampling. 10-fold stratified cross-validation is employed for model training and validation. Comparative experiments showed that CatBoost, LightGBM, and XGBoost outperformed AdaBoost across the evaluated datasets. Based on these results, a Hybrid Boosted Stacking Ensemble

Framework (HBSEF) is developed using CatBoost, XGBoost, and LightGBM as base learners and XGBoost as the meta-learner. The proposed HBSEF achieved better predictive performance than the best individual ensemble model, with an improvement of up to 6.4 percentage points in accuracy. Statistical analysis using the paired t-test further supported the observed performance differences. In addition, SHAP-based explainability provided insights into global and local feature contributions and further highlighted the importance of the six selected software metrics. Despite these contributions, the study has some limitations. It primarily focuses on static and object-oriented software metrics and does not incorporate process, change, or developer metrics. Moreover, the use of public benchmark datasets may not fully represent the complexity of industrial software systems. The framework is also evaluated under conventional batch-learning and cross-validation settings rather than continuously evolving development environments. Future work will extend the framework by incorporating process, change, and developer metrics, as well as investigating deep learning and metaheuristic optimization techniques to further improve feature selection and fault-prediction performance. Cross-project, cross-version, and industrial-scale validation will also be considered to assess the generalizability and practical applicability of the proposed framework.

REFERENCES

1. A. Iqbal and S. Aftab, "A classification framework for software defect prediction using multi-filter feature selection technique and MLP," *Int. J. Mod. Educ. Comput. Sci.*, vol. 12, no. 1, pp. 18–25, 2020, doi: 10.5815/ijmecs.2020.01.03.
2. C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Inf. Sci. (Ny).*, vol. 179, no. 8, pp. 1040–1058, 2009, doi: 10.1016/j.ins.2008.12.001.
3. S. S. Rathore, "A Comparative Study of Feature-Ranking and Feature-Subset Selection Techniques for Improved Fault Prediction," 2014.
4. K. Gao, T. M. Khoshgoftaar, and N. Seliya, "Predicting high-risk program modules by selecting the right software measurements," *Softw. Qual. J.*, vol. 20, no. 1, pp. 3–42, 2012, doi: 10.1007/s11219-011-9132-0.
5. N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *Mach. Learn. Appl. Softw. Eng.*, vol. 25, no. 5, pp. 72–86, 2005.
6. H. Liu, "Toward Integrating Feature Selection Algorithms for Classification and Clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 0826, no. 4, p. 126, 2005, doi: 10.1117/12.942023.
7. H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction," *Expert Syst. Appl.*, vol. 122, pp. 27–42, 2019, doi: 10.1016/j.eswa.2018.12.033.
8. M. Ali *et al.*, "Analysis of Feature Selection Methods in Software Defect Prediction Models," *IEEE Access*, vol. 11, no. December, pp. 145954–145974, 2023, doi: 10.1109/ACCESS.2023.3343249.
9. G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 16–28, 2014, doi: 10.1016/j.compeleceng.2013.11.024.
10. T. Thaher, M. Mafarja, B. Abdalhaq, and H. Chantar, "Wrapper-based Feature Selection for Imbalanced Data using Binary Queuing Search Algorithm," *2019 2nd Int. Conf. New Trends Comput. Sci. ICTCS 2019 - Proc.*, 2019, doi: 10.1109/ICTCS.2019.8923039.
11. V. K. C. Mula, L. Kumar, L. B. Murthy, and A. Krishna, "Software Sentiment Analysis using Deep-learning Approach with Word-Embedding Techniques," *Proc. 17th Conf. Comput. Sci. Intell. Syst.*, vol. 30, pp. 873–882, 2022, doi: 10.15439/2022f138.
12. W. Jerbi, A. Ben Brahim, and N. Essoussi, "A hybrid embedded-filter method for improving feature selection stability of random forests," *Adv. Intell. Syst. Comput.*, vol. 552, no. His 2016, pp. 370–379, 2017, doi: 10.1007/978-3-319-52941-7_37.
13. L. Kumar, S. Misra, and S. K. Rath, "An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes," *Comput. Stand. Interfaces*, vol. 53, no. December 2016, pp. 1–32, 2017, doi: 10.1016/j.csi.2017.02.003.
14. P. Wang, C. Jin, and S. W. Jin, "Software defect prediction scheme based on feature selection," *Proc. 2012 4th Int. Symp. Inf. Sci. Eng. ISISE 2012*, pp. 477–480, 2012, doi: 10.1109/ISISE.2012.114.
15. H. Wang, T. M. Khoshgoftaar, and Q. Liang, "A study of software metric selection techniques: Stability analysis and defect prediction model performance," *Int. J. Artif. Intell. Tools*, vol. 22, no. 5, 2013, doi: 10.1142/S0218213013600105.
16. S. C. Rathi, S. Misra, R. Colomo-Palacios, R. Adarsh, L. B. M. Neti, and L. Kumar, "Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction," *Expert Syst. Appl.*, vol. 223, no. March, 2023, doi: 10.1016/j.eswa.2023.119806.
17. A. A. Saifan and L. A. Abuwardih, "Software defect prediction based on feature subset selection and ensemble classification," *ECTI Trans. Comput. Inf. Technol.*, vol. 14, no. 2, pp. 213–228, 2020, doi: 10.37936/ecti-cit.2020142.224489.
18. C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on Pareto-based multi-objective feature selection for software defect prediction," *J. Syst. Softw.*, vol. 152, pp. 215–238, 2019, doi: 10.1016/j.jss.2019.03.012.

19. I. H. Laradji, M. Alshayeb, and L. Ghouti, “Software defect prediction using ensemble learning on selected features,” *Inf. Softw. Technol.*, vol. 58, pp. 388–402, 2015, doi: 10.1016/j.infsof.2014.07.005.
20. R. Shatnawi, “The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction,” *Innov. Syst. Softw. Eng.*, vol. 13, no. 2–3, pp. 201–217, 2017, doi: 10.1007/s11334-017-0295-0.
21. C. W. Yohannese and T. Li, “A Combined-Learning based framework for improved software fault prediction,” *Int. J. Comput. Intell. Syst.*, vol. 10, no. 1, pp. 647–662, 2017, doi: 10.2991/ijcis.2017.10.1.43.
22. H. Gong, Y. Li, J. Zhang, B. Zhang, and X. Wang, “A new filter feature selection algorithm for classification task by ensembling Pearson correlation coefficient and mutual information,” *Eng. Appl. Artif. Intell.*, vol. 131, no. August 2023, p. 107865, 2024, doi: 10.1016/j.engappai.2024.107865.
23. D. Sharma and P. Chandra, “Software fault prediction using machine-learning techniques,” *Smart Innov. Syst. Technol.*, vol. 78, pp. 541–549, 2018, doi: 10.1007/978-981-10-5547-8_56.
24. S. Mehta, “Improved prediction of software defects using ensemble machine learning techniques,” *Neural Comput. Appl.*, vol. 33, no. 16, pp. 10551–10562, 2021, doi: 10.1007/s00521-021-05811-3.
25. X. Chen, D. Zhang, Z. Q. Cui, Q. Gu, and X. L. Ju, “DP-Share: Privacy-Preserving Software Defect Prediction Model Sharing Through Differential Privacy,” *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 1020–1038, 2019, doi: 10.1007/s11390-019-1958-0.
26. K. Bhandari, K. Kumar, and A. L. Sangal, *Data quality issues in software fault prediction: a systematic literature review*, no. 0123456789. Springer Netherlands, 2022. doi: 10.1007/s10462-022-10371-6.
27. S. Singh and T. U. Haider, “Selection of best feature reduction method for module-based software defect prediction,” *J. Phys. Conf. Ser.*, vol. 2273, no. 1, 2022, doi: 10.1088/1742-6596/2273/1/012002.
28. B. A. Oluwagbemiga, B. Shuib, S. J. Abdulkadir, and A. Sobri, “A Hybrid Multi-Filter Wrapper Feature Selection Method for Software Defect Predictors,” vol. 8, no. 2, pp. 916–922, 2019.
29. J. L. Potharlanka and M. Nirupama Bhat, “Feature importance feedback with Deep Q process in ensemble-based metaheuristic feature selection algorithms,” *Sci. Rep.*, vol. 14, no. 1, pp. 1–30, 2024, doi: 10.1038/s41598-024-53141-w.
30. I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, “Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction,” *IEEE Access*, vol. 8, pp. 8041–8055, 2020, doi: 10.1109/ACCESS.2020.2964321.
31. F. Matloob *et al.*, “Software defect prediction using ensemble learning: A systematic literature review,” *IEEE Access*, vol. 9, pp. 98754–98771, 2021, doi: 10.1109/ACCESS.2021.3095559.
32. T. Zivkovic, B. Nikolic, V. Simic, D. Pamucar, and N. Bacanin, “Software defects prediction by metaheuristics-tuned extreme gradient boosting and analysis based on Shapley Additive Explanations,” *Appl. Soft Comput.*, vol. 146, p. 110659, 2023, doi: 10.1016/j.asoc.2023.110659.
33. C. Anjali, J. P. M. Dhas, and J. Amar Pratap Singh, “Moth Flame Optimization Based FCNN for Prediction of Bugs in Software,” *Intell. Autom. Soft Comput.*, vol. 36, no. 2, pp. 1241–1256, 2023, doi: 10.32604/iasc.2023.029678.
34. H. Aljamaan and A. Alazba, “Software defect prediction using tree-based ensembles,” *PROMISE 2020 - Proc. 16th ACM Int. Conf. Predict. Model. Data Anal. Softw. Eng. Co-located with ESEC/FSE 2020*, pp. 1–10, 2020, doi: 10.1145/3416508.3417114.
35. F. Min *et al.*, “Fault prediction for distribution network based on CNN and LightGBM algorithm,” *2019 14th IEEE Int. Conf. Electron. Meas. Instruments, ICEMI 2019*, pp. 1020–1026, 2019, doi: 10.1109/ICEMI46757.2019.9101423.
36. A. Kaur and I. Kaur, “An empirical evaluation of classification algorithms for fault prediction in open source projects,” *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 30, no. 1, pp. 2–17, 2018, doi: 10.1016/j.jksuci.2016.04.002.
37. C. L. Prabha and N. Shivakumar, “Software Defect Prediction Using Machine Learning Techniques,” *Proc. 4th Int. Conf. Trends Electron. Informatics, ICOEI 2020*, no. Icoei, pp. 728–733, 2020, doi: 10.1109/ICOEI48184.2020.9142909.
38. S. Hussain, J. Keung, A. A. Khan, and K. E. Bennin, “Performance evaluation of ensemble methods for software fault prediction: An experiment,” *ACM Int. Conf. Proceeding Ser.*, vol. 28-Septemb, pp. 91–95, 2015, doi: 10.1145/2811681.2811699.
39. W. Afzal and R. Torkar, “Towards benchmarking feature subset selection methods for software fault prediction,” *Stud. Comput. Intell.*, vol. 617, pp. 33–58, 2016, doi: 10.1007/978-3-319-25964-2_3.
40. L. Qiong Chen, C. Wang, and S. Long Song, “Software defect prediction based on nested-stacking and heterogeneous feature selection,” *Complex Intell. Syst.*, vol. 8, no. 4, pp. 3333–3348, 2022, doi: 10.1007/s40747-022-00676-y.
41. O. T. Ensembles and A. Alazba, “Applied Sciences Software Defect Prediction Using Stacking Generalization of,” 2022.
42. S. P. Sahu, B. R. Reddy, D. Mukherjee, D. M. Shyamla, and B. S. Verma, “A hybrid approach to software fault prediction using genetic programming and ensemble learning methods,” *Int. J. Syst. Assur. Eng. Manag.*, vol. 13, no. 4, pp. 1746–1760, 2022, doi: 10.1007/s13198-021-01532-x.