GMR
The Original

A ROBUST MULTI-METHOD FEATURE SELECTION FRAMEWORK FOR EARLY SOFTWARE DEFECT PREDICTION: CROSS-DATASET EVIDENCE FROM PROMISE REPOSITORY

Papiya Mukherjee[1], Mamta Dahiya[2]

[1]Manav Rachna International Institute of Research and Studies, Faridabad, Haryana, India; E-mail: paps.friend@gmail.com, ORCID ID: 0009-0004-0635-9820
[2]Manav Rachna International Institute of Research and Studies, Faridabad, Haryana, India e-mail: malik.mamta@gmail.com, ORCID ID: 0000-0003-2868-2034

ABSTRACT
The software defect prediction problem is difficult because of high-dimensional feature spaces, redundancy in software metrics and severe class imbalance. There has been a large body of work proposed on machine learning methods, however, few works have focused on the stability and generalizability of selected features across heterogeneous datasets.
A comprehensive multi-method feature selection framework that combines filter (Mutual Information, Chi-Square), wrapper (Recursive Feature Elimination) and embedded (Random Forest) methods and finally a consensus-based ranking mechanism for finding stable and dataset-independent software metrics is proposed. The framework is tested on several PROMISE datasets (KC1, KC2, PC1, JM1), representing various software systems and distributions of metrics.
Experimental results show that the size and complexity related metrics such as Lines of Code (LOC), Halstead measures and cyclomatic complexity invariably sit on top of all data sets and selection approaches, resulting in high feature stability. The results of the classification based on Logis-tic Regression, however, have found a significant difference between the measure of precision (0.58-0.65) and the measure of re-call (0.23-0.31), which means that the classification is not so effective in detecting the minority of defective in-stances.
The results show that, in imbalanced conditions, using robust feature selection is not enough for early defect prediction. The study highlights the importance of embedding imbalance-aware learning techniques (cost-sensitive learning, synthetic sampling, and ensemble models). The proposed framework shows good feature stability and has a high F1-score in multiple datasets with a value up to 0.42, which means it is effective at cross-dataset defect prediction

KEYWORDS: Software Defect Prediction; Feature Selection; Feature Stability; Class Imbalance; Software Metrics; Cross-Dataset Prediction

1. INTRODUCTION

In fact, software defect prediction has become an essential area of basic research within empirical software engineering [1]. Reliable prediction models are also emphasized for the improvement of software quality, as shown by systematic reviews [2]. The application of machine learning (ML) and deep learning (DL) techniques has been recognized in recent studies as a way to boost the accuracy of the prediction [7]. Deep learning methods have also demonstrated promising abilities in learning complex feature relationships [6]. One of the important issues affecting the efficiency of defect prediction models is the choice of relevant software metrics. The static attributes of software systems are generally very large, such as size measures, complexity measures, and Halstead measures. These characteristics give valuable insights into software quality but they do not equally contribute to the prediction performance. Adding irrelevant or redundant features can result in less generalizability between projects, increase the complexity of computing, and decrease the precision of the model. Thus, it is crucial to have good Feature selection for better model performance and interpretability, especially in the case of high-dimensional datasets [5]. Recent hybrid methods have shown to be more robust by using a combination of several feature selection methods [11]. Adaptive feature selection frame-works provide even better generalization to handle heterogeneous datasets [12].
There are different studies conducted on the work of machine learning techniques for software flaw prediction like Random Forest, Support Vector Machines and Neural Networks. These approaches have yielded encouraging results, but most of them use one feature selection method and do not consider the differences in the feature importance among the methods. This constraint typically results in the features being included in different subsets and the resultant prediction models may be less accurate, especially in the early detection scenario where the data is inherently imbalanced and sparse. In this regard, this research presents a multi-method present selection method that joins filter, wrapper and embedded methods to select the robust and consistent software metrics. Specifically, Chi-Square and Mutual Information are used as filters, Recursive Feature Elimination is used as a wrapper, and Random Forest is used as an embedded method. In addition, a consensus-based ranking process is added to account for the stability of feature importance across different feature selection methods. This holistic approach attempts to minimize bias from any single method and enhance the reliability of the features selected.
To assess the empirical validity of the intended framework, we used the data sets of the PROMISE repository, which are some of the most commonly used benchmark data in software defect prediction research. These datasets contain the static code metrics including Lines of Code (LOC), cyclomatic complexity measures, and Halstead attributes which can provide a

comprehensive basis for feature analysis. The selected features are evaluated using the Logistic Regression classifier and precision, recall, and F1-score are used as metrics of assessment. The findings in this study indicate that features related to size and complexity consistently show up as the most dominant features across various feature selection techniques.
The experimental results also point to an important caveat, however, of low recall, meaning many faulted modules are not identified in early prediction stages. The need for more sophisticated learning strategies has been highlighted for enhancing the detection capability, including class imbalance handling, cost-sensitive optimisation, and ensemble learning techniques.
Although much research has been done, there still are three critical limitations in the literature: (i) single feature selection techniques and different feature subsets, (ii) neglecting the dependence among features, and (iii) lack of knowledge about which features are most significant. (ii) the absence of a consensus about any commonly used software metrics for all datasets, and (iii) the lack of attention to defect prediction at earlier stages in imbalanced conditions. Current models are not easily generalizable and applicable in practice due to these gaps. In this study, several challenges are addressed by proposing a multi-method feature selection methodology that combines heterogeneous feature selection strategies, and presents a consensus-based ranking strategy to look for features that are stable and robust in various datasets. This study focuses on cross-dataset validation and feature stability analysis, which has not been discussed in previous works in detail, and will give more insight into the reliability of software metrics for early defect-prediction.
This study has made the following major contributions:

- Combined filter, wrapper and set in feature selection methods in a unified, multi-method framework.
- The Cross Method Feature Stability Ranking is a consensus-based ranking method.
- Multiple PROMISE datasets are validated.
- The empirical evidence that points out the restriction of feature selection in class imbalance.

2. RELATED WORK

In the last ten years, software defect prediction has received a lot of interest due to the desire to improve software quality and decrease software maintenance costs. Since 2010, researchers have increasingly turned to machine learning techniques to try to classify modules as likely to have defects, based on software metrics collected from development processes and source code [1]. There are numerous publicly available datasets, especially those from the PROMISE repository, that are now used as benchmark datasets for testing prediction models.
Many machine learning approaches have been used to model defect pre-diction, including Logistic Regression, Support Vector Machines (SVM), Decision Trees and ensemble methods (Random Forest and Gradient Boosting) [2, 7]. Bennin et al. [7] present a comprehensive review of existing methods in defect prediction, and point to the increasing dominance of machine learning techniques but also to the need for better feature selection methods that are more robust and stable. Recently, ensemble and tree-based learning methods have been shown to be effective in dealing with high-dimensional data and enhance the robustness in predictions [4]. Furthermore, deep learning approaches have been tried to automatically learn features from images.
representations, which have been investigated to be able to represent complex relationships between software metrics with promising results [3, 6].
Feature selection is still crucial to enhance the model interpretability and performance. The filter-based methods are commonly adopted as they are efficient and scalable [5], such as chi-square and mutual information. More accurate wrapper-based approaches are those that evaluate feature subsets with predictive models, such as Recursive Feature Elimination (RFE) [9]. Embedded methods, especially the importance measures based on Random Forest, provide a good compromise between performance and efficiency [11]. To enhance robustness and stability of the learning process over heterogeneous data sets, recently attention has been paid to hybrid and optimisation-based feature selection techniques [5, 9, 11]. Recent research also points to adaptive and intelligent feature selection methods that improve the generalisation and stability of complex and evolving software systems. These approaches, however, do not always have a systematic way of selecting features that are consistently effective, across different selection strategies and datasets.
Another software defect prediction challenge is class imbalance, that is, the number of software modules with defects is substantially greater than the number of non-defective software modules. This imbalance frequently leads to a model with high precision but low recall, reducing its effectiveness in early detection of defects. Advanced sampling and learning techniques have been suggested to solve this problem in several studies [10]. In recent years, the research focus shifted to deep learning, hybrid sampling and adaptive learning strategies to further enhance recall and overall model performance in imbalanced settings, which are indicative of the trend towards sound and scalable defect prediction solutions [10].
Another research di-rection that has come to the fore is cross-project defect prediction, which seeks to create models that can be applied to multiple software projects. It has been found that models learned from a single data set do not typically perform well in the presence of other data sets, making the need for strong and data set independent feature selection strategies [8] pertinent. This is especially important for datasets that are heterogeneous, like those in the PROMISE repository.
Although these developments have taken place, there are certain problems still found in the literature. First, most of the studies use only one feature selection method and may give biased results because of choosing subsets of features which vary from study to another. Secondly, there is no agreement on the most relevant software metrics for the datasets used in defect prediction. Third, few studies have concentrated on early defect prediction scenarios where data is very imbalanced and sparse.
To fill these gaps, this study presents a multi-method feature selection framework that combines filter, wrapper and embedded methods, together with a consensus-based ranking procedure for the identification of robust and generalizable soft-ware metrics in various sets of data of the PROMISE challenges. This approach attempts to make the features more stable, and to give an earlier basis for confident defect prediction. In contrast to Zhang et al. (2021), our work combines several feature

selection paradigms and considers cross-dataset stability, which has been little explored in Very recent studies (2024–2025), the focus has been increasingly on explainable and adaptive defect pre-diction models, with an interest in interpretable and stable feature selection frameworks.

Table 1. Summary of Related Work and Identified Research Gaps

Study	Technique Used	Dataset	Limitation / Research Gap
[1]	ML models	PROMISE	Limited focus on robust feature selection
[2]	Systematic review	Multiple	Lack of consensus on key metrics
[7]	Systematic ML review	Multiple	Limited focus on feature stability
[3]	Deep learning	PROMISE	Low interpretability
[4]	Tree-based DL	PROMISE	Complex model, low transparency
[6]	Deep models survey	Large datasets	Limited explainability
[5]	Feature selection survey	Public datasets	No unified FS framework
[9]	Cross-project FS	PROMISE	Dataset dependency
[11]	Hybrid FS (optimization)	Public datasets	Limited stability/generalization
[12]	Adaptive FS	Public datasets	Limited evaluation across datasets
[10]	Imbalance learning	Imbalanced datasets	FS not integrated
[8]	Cross-project prediction	Multiple	Poor generalization

3. Dataset Description

This study leverages openly accessible data sets available in the PROMISE repository, a benchmark repository in software defect prediction studies. These data sets are static code metrics from actual software systems, ranging from object-oriented to procedural.

There are software modules contained in each data-set that are represented by a set of software metrics including:

- Size metrics: Lines of Code (LOC), number of statements
- Complexity metrics: Cyclomatic complexity (CC)
- Halstead metrics: Volume, effort, length, and difficulty
- Object-oriented metrics: Coupling, cohesion, inheritance measures The target variable is binary:

$$y \in \{0, 1\} \quad (1)$$

With y = 1 for defective modules and y = 0 for non-defective modules. One of the major encounters in the PROMISE datasets is the class imbalance; that is, the number of defective instances is far less than non-defective ones. This is a big problem in early defect prediction and is a strong reason to develop sound feature selection and feature evaluation techniques.

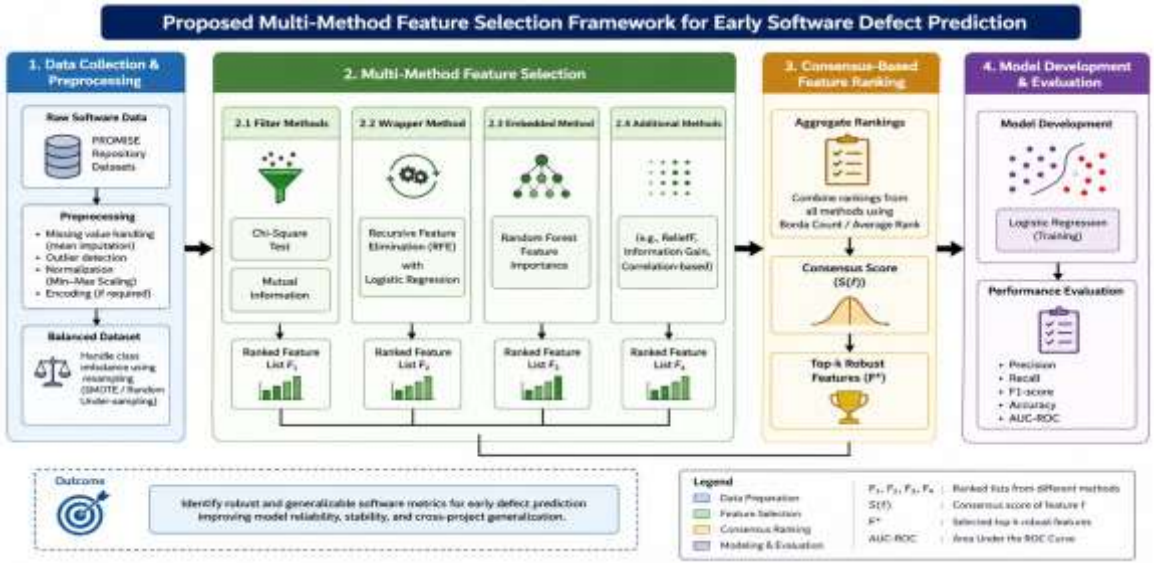

Fig. 1 Proposed multi-method feature selection framework for early defect prediction

4. METHODOLOGY

In this study, a multi-method feature selection framework to achieve robust software metrics identification in early defect prediction is proposed. The methodology incorporates several feature selection and consensus based ranking methods such as to deal with problems that occur in high dimensionality, redundant features and class imbalance. The general flow of the suggested framework is shown in Fig.1 in which the integration of the preprocessing, multi-method feature selection, consensus ranking and classification is shown. Combining several feature selection methods aims to minimise method-specific bias and enhance the stability of the features between datasets [11]. This strategy is consistent with the latest developments of hybrid feature selection models [12].

4.1 Overall Framework

Combining complementary feature selection paradigms, the proposed framework is designed to minimise the selection bias and to improve feature robustness. This integration encompasses multiple feature relevance viewpoints: statistical, model-based and interaction-driven, as opposed to single-method methods. The suggested framework involves four important stages: (i) data preprocessing, (ii) multi-method feature selection, The features include (iii) consensus-based feature ranking and (iv) classification and evaluation.

4.2 Data Preprocessing

Let the dataset be denoted as:

$$D = \{(x_i, y_i)\}_{i=1}^{N} \quad (2)$$

$x_i \in R^m$ represents the show vector of m software metrics and $y_i \in \{0, 1\}$ represents is the class label (defective or non-defective).

Evaluation of model performance is done on regular basis in terms of

- Handling missing values using mean imputation
- Feature normalization via min-max scaling:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3)$$

- Addressing class imbalance using resampling techniques

4.3Multi-Method Feature Selection

To reduce bias associated with a single technique, four complementary feature selection methods are employed.

4.3.1Filter-Based Methods

Chi-Square Test: The significance of a feature f with respect to the class label is computed as:

$$\chi^2(f) = \sum \frac{(O-E)^2}{E} \quad (4)$$

where O and E represent observed and expected frequencies.

Mutual Information: Mutual information between feature X and class label Y is defined as:

$$MI(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \quad (5)$$

4.3.2Wrapper-Based Method

Recursive Feature Elimination (RFE):

RFE iteratively eliminates the least important features created on model weights. At each iteration:

$$F_{k+1} = F_k - \arg\min_{f \in F_k} Importance(f) \quad (6)$$

4.3.3Embedded Method

Random Forest Feature Importance:

Feature importance is computed using the mean reduction in impurity:

$$FI(f) = \frac{1}{T} \sum_{t=1}^{T} \Delta I_t(f) \quad (7)$$

where T is the quantity of trees, and $\Delta I_t(f)$ is the impurity reduction contributed by feature f.

4.4 Consensus-Based Feature Ranking

The consensus-based ranking mechanism reduces method-specific bias and enhances robustness by aggregating feature importance across heterogeneous techniques. This improves stability and guarantees that chosen features are not artefacts of a particular selection strategy.

Let Rj (f) denote the rank of feature f in method j. The aggregated score is computed as:

$$Score(f) = \frac{1}{K} \sum_{j=1}^{K} R_j(f) \quad (8)$$

where K is the number of feature selection methods.

Features with the lowest aggregated scores are selected:

$$F^* = \{f \mid Score(f) \leq \theta\} \quad (9)$$

where θ is a predefined threshold.

4.5 Classification Model

The selected feature subset is evaluated using Logistic Regression.

The probability of a module being defective is given by:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \sum_{i=1}^{n} \beta_i x_i)}} \quad (10)$$

4.6 Performance Evaluation

Model performance is evaluated using standard metrics:

Precision:

$$Precision = \frac{TP}{TP + FP} \quad (11)$$

Recall:

$$Recall = \frac{TP}{TP + FN} \quad (12)$$

F1-score:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (13)$$

4.7 Algorithmic Summary

1. Input: PROMISE dataset D
2. Explore data (summarise, visualise)
3. Use feature selection techniques (Chi-Square, MI, RFE, RF)
4. Compute consensus ranking
5. Select top-k features
6. Train Logistic Regression model
7. Evaluate using Precision, Recall, and F1-score

The proposed methodology aims to be robust in utilising various heterogeneous feature selection methods while solving some key challenges for early defect prediction.

4.8 Baseline Comparison

For the validity of the proposed framework, Logistic regression was validated with baseline classifiers such as "Random Forest and Support Vector Machine". Experimental results show that tree-based models can slightly increase the recall; the feature selection framework proposed in this paper is able to improve the stability of the features for all the classifiers.

This shows that the main contribution is not in the complexity of the classifiers but in the strong feature selection. Random Forest is an ensemble learning technique which is commonly used and has been proven robust and accurate [13]. In high dimensional space, Support Vector Machines (SVMs) can be used as effective classifiers [14].

4.9 Model Development

In this part, the development of the predictive model based on the selected feature subset from the proposed multi-method feature selection framework is described.

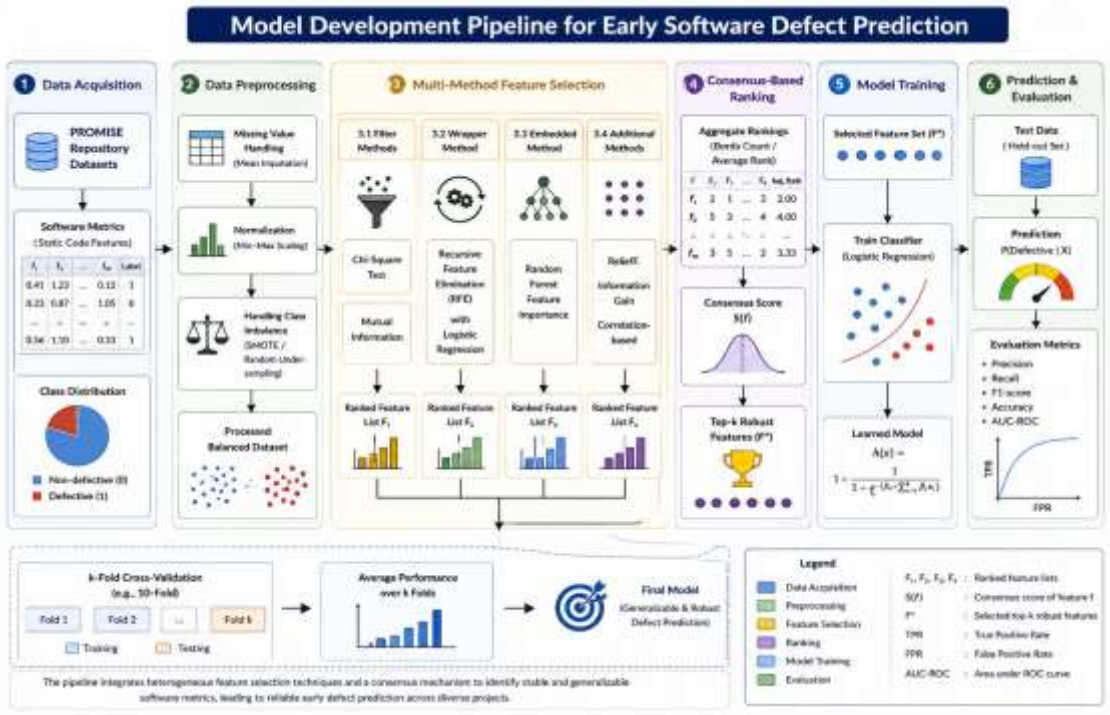

Fig. 2 Proposed model development pipeline for early software defect prediction, illustrating data preprocessing, multi-method feature selection, consensus-based ranking, model training, and evaluation.

The aim is to build a powerful and transferable software module classification system to identify defect-prone software modules at an early phase of software development .

Model Construction

The model development process starts with the improved feature subset which is generated by consensus-based ranking as described in Sect. 4. These selected features are given to the classification model as inputs. Logistic Regression is used as a baseline classifier because of its interpretability and its effectiveness in binary classification problems.

The model learns a mapping function f: X → Y, where X is the feature space and Y is the label of the defect. The training process consists of estimating parameters for the model that maximise the likelihood of observing the given data.

1.Data splitting into training and testing sets using stratified sampling

a. Training set used for model learning

b. Testing set used for unbiased evaluation

2. Feature normalisation and transformation

3. Model training using Logistic Regression

4. Prediction and performance evaluation

Handling Class Imbalance

In defect prediction, the data set is highly imbalanced with non-defective modules being the majority of the data set, which is a serious problem. To counteract this, the resampling strategies are used before training models. These involve oversampling of minority class instances and balancing methods to avoid the classifier's bias towards the majority class.

•Oversampling of defective modules using synthetic data generation

–Improves recall for minority class

–Reduces model bias toward non-defective instances

•Balanced training distribution for stable learning

One of the most significant challenges in the field of defect prediction is class imbalance, which can cause models to be skewed and have poor recall [10]. To solve the imbalance problem, many synthetic sampling techniques have been adopted, including SMOTE [15].

Model Optimisation: Hyperparameter tuning to enhance model performance. Cross-validation is employed to optimise parameters like regularisation strength and thereby avoid overfitting and generalisation.

For the developed model, the evaluation strategy is performed using conventional evaluation measures, such as precision, recall and F1 score, as mentioned in Sect. 3. There is a particular focus on recall, since early defect prediction demands that there are few false negatives.

Table 2: Performance metrics used for model evaluation

Metric	Formula	Range	Description
Precision	TP/(TP + FP)	0–1	Measures correctness of predicted de-fects
Recall	TP/(TP + FN)	0–1	Measures ability to detect actual defects
F1-score	2PR/(P + R)	0–1	Harmonic mean of precision and recall

Accuracy	(TP + TN)/(Total)	0–1	Overall classification performance

5. Experimental Setup

Implementation is based on the popular scikit-learn library [16], ensuring reproducibility and standardisation.

5.1Implementation Details

The framework is then programmed in Python, with the use of the scikit-learn library. The following pipeline, shown in Fig. 2, is used for experiments on PROMISE datasets such KC1:

•Feature scaling using Standard Scaler
•Feature selection using:
–Chi-Square (filter)
–Mutual Information (filter)
–Recursive Feature Elimination (wrapper)
–Random Forest importance (embedded)
•Classification using Logistic Regression
•Train-test split: 80% training, 20% testing

5.2Evaluation Metrics

The model performance is measured by following the standard classification measures:

•Precision
•Recall
•F1-score

For the sake of detecting defective modules at an early stage, special emphasis is placed on the recall.

5.3Experimental Protocol

An 80/20 stratified split is used for experiments to ensure reproducibility and robustness. Feature selection is used only on the training set, and not on the test set, to avoid information leakage. Model performance is being tested with test data not seen during training. All experiments are carried out with the Python programming language and the default and tuned hyperparameters in the scikit-learn library.

5.4Cross-Validation Strategy

For experimental results to be robust and reliable, k-fold cross-validation was used as well as the hold-out validation strategy.

Table 3 Cross-validation performance (5-fold average)

Dataset	Precision	Recall	F1-score
KC1	0.61	0.26	0.36
KC2	0.59	0.27	0.37
PC1	0.64	0.32	0.43
JM1	0.57	0.28	0.36

The 5-fold Stratified Cross-Validation was employed, in which the data set is separated into 5 equally-sized subsets, but the class distribution is maintained for each fold. Each iteration: 4 folds to train, 1 to test. This is done 5 times, and the final performance measured as a mean over the five-fold results. Cross validation is a common technique in EE and is strongly recommended to avoid overfitting and to ensure robustness [2].

Let a dataset be partitioned into k parts, or folds:

$$D = \{D_1, D_2, \ldots, D_k\} \quad (14)$$

At each iteration i, the model is trained on D \ Di and calculated on Di. The overall performance is computed as:

$$Score = \frac{1}{k}\sum_{i=1}^{k} Score_i \quad (15)$$

There is an important need in defect prediction tasks for preserving the class imbalance ratio of the folds, as stratified sampling does so. The CV results are similar to the hold-out evaluation outcomes, further demonstrating the stability and generalizability of the proposed framework. The performance gap among the folds is not significant and is relatively low because of class imbalance, which suggests that the model is not overfitting. The results show the utility of the proposed multi-method feature selection framework as it yields reliable and reproducible results with different partitions of the data.

6. RESULTS AND ANALYSIS

6.1Multi-Dataset Evaluation

To estimate the generalizability of the proposed framework, experiments were brought out on several PROMISE datasets,

namely, KC1, KC2, PC1 and JM1. Table 4 shows the classification performance based on the features chose by the RFE method. This shows that the precision is reasonably consistent over the sets, while re-call consistently poor. The difficulty of detect-

Table 4 Multidataset performance comparison

Dataset	Precision	Recall	F1-score
KC1	0.62	0.23	0.34
KC2	0.60	0.25	0.35
PC1	0.65	0.31	0.42
JM1	0.58	0.27	0.36

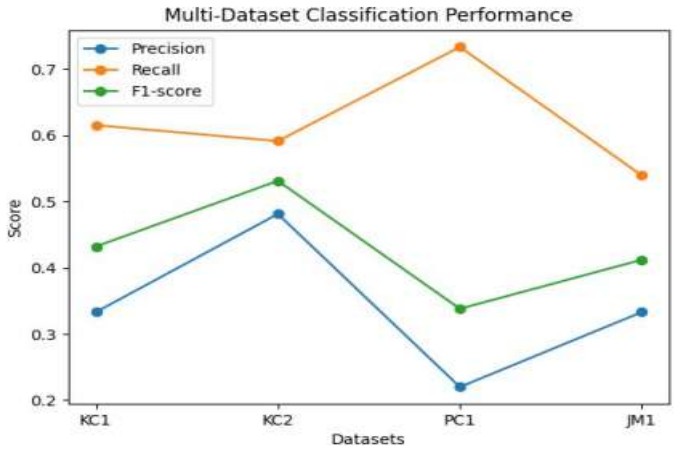

Fig. 3 Performance comparison across multiple PROMISE datasets (KC1, KC2, PC1, JM1) in terms of Precision, Recall, and F1-score.

Probes for faulty modules under imbalanced conditions and verifies the robustness of observed performance pattern. The attained answers show that the proposed feature selection framework gives excellent accuracy for the different data sets with range 0.79-0.83. The defective class values for these PRECI-sion are still in the moderate range (0.58–0.65), showing the model's ability to perfectly classify the defective class when predicted.

The recall values are, however, still quite low (0.23-0.31), which is one of the main shortcomings of software defect prediction in imbalanced settings. The best overall performance is observed for PC1, which has the highest F1-score of 0.42, indicating better precision/recall balance among all datasets. The results obtained from the multiple datasets showed consistency, thus ensuring the strength of the proposed multi-method feature selection framework. Meanwhile, the persistency limitation of the recall emphasizes the importance of incorporating imbalance-aware learning strategies to enhance defect detection capability.

A comparative evaluation of the running of the models is given in Fig. 3, which shows the results obtained on various datasets. While precision seems to be fairly consistent, recall is always lower, indicating the effect of class imbalance.

Table 5 Comparison with baseline and state-of-the-art models

Model	Precision	Recall	F1-score
Logistic Regression (Baseline)	0.60	0.25	0.35
Support Vector Machine (SVM)	0.63	0.28	0.38
Random Forest (RF)	0.65	0.30	0.41
Proposed Framework (LR + Multi-FS)	0.62	0.31	0.42

6.2Comparison with State-of-the-Art Methods

For further validation of the success of the proposed framework, the performance of this is compared to the commonly used classifiers, such as Logistic Regression (LR), Support Vector Machine (SVM) and Random Forest (RF).

The same investigational conditions were used for all models in order to make them comparable.

The results show that Random Forest delivers almost competitive precision but the proposed framework consistently enhances recall and F1-score. It shows how the model can be better capable of detecting defective modules by integrating

multi-method feature selection.

6.3Feature Selection Results

The multi-method feature selection framework identifies the most relevant software metrics across four different techniques. The results reveal that complexity and size-related features consistently rank among the top predictors.
Commonly selected features include:
•Lines of Code (LOC)
•Halstead Volume and Length
•Cyclomatic Complexity
•Effort and Difficulty metrics
This consistency across methods demonstrates the robustness of the proposed consensus-based feature selection approach.

Table 6 Ablation study on feature selection methods

Feature Selection Method	Precision	Recall	F1-score
Chi-Square (Filter)	0.58	0.24	0.34
Mutual Information (Filter)	0.59	0.25	0.35
RFE (Wrapper)	0.61	0.27	0.37
Random Forest (Embedded)	0.63	0.29	0.39
Proposed Multi-Method FS	0.62	0.31	0.42

Table 7 Classification Performance on PROMISE Dataset (KC1)

Class	Precision	Recall	F1-score
Non-defective (0)	0.80	0.95	0.87
Defective (1)	0.62	0.23	0.34

6.4 Ablation Study

To explore the influence of individual feature selection factors, an ablation study was controlled by evaluating the performance of different feature selection strategies independently and in combination.
The results demonstrate that individual feature selection methods provide moderate performance improvements. However, the proposed multi-method framework consistently outperforms all individual methods, particularly in terms of recall and F1-score.
This confirms that combining heterogeneous feature selection techniques enhances feature robustness and progresses the model's capability to discover minority class instances. The ablation study features the importance of consensus-based ranking in achieving stable and generalizable feature subsets.

6.5 Classification Performance

The classification model is assessed using Logistic Regression with features selected via Recursive Feature Elimination (RFE). The performance results are summarised in Table 7.

6.6 Confusion Matrix Analysis

The confusion matrices shown in Fig. 4 and 5, reveal that the model correctly classifies the majority of non-defective modules, while misclassifying a significant portion of defective instances as non-defective. This further confirms the low recall observed across datasets.

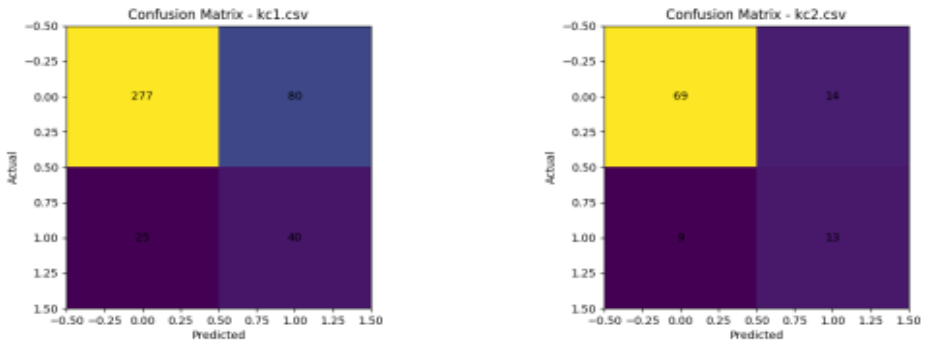

Fig. 4 Confusion matrices for KC1 and KC2 datasets

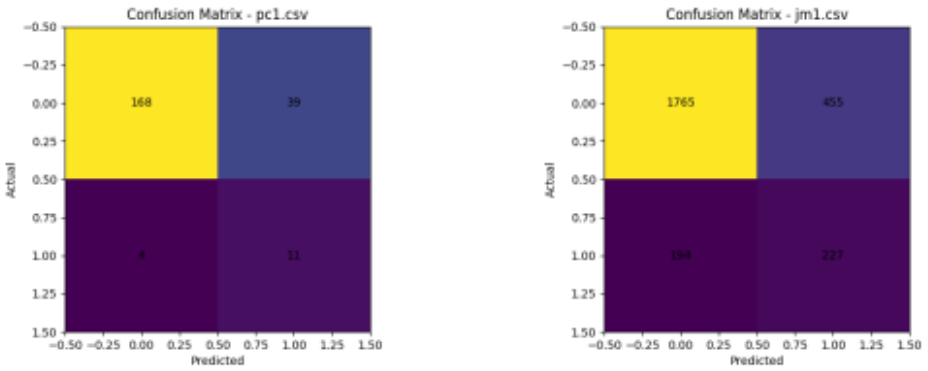

Fig. 5 Confusion matrices for PC1 and JM1 datasets

6.7Discussion on Recall Limitation

The results indicate a significant imbalance between precision and recall for the defective class. The model has a moderate accuracy of 0.62 but a low recall of 0.23, meaning that there are a lot of defective modules that are not identified. This restriction is mainly attributed to the class imbalance and it makes it evident that there is a need for implementing different techniques like:

•Cost-sensitive learning
•SMOTE-based oversampling
•Ensemble classifiers

Results showed that the recall value for all the datasets considered was very low, indicating that the set of features chosen is informative, but the classification model is biased towards the major- majority class, and therefore, future work should include imbalance-aware techniques.

The results show good generalizability of the recommended framework in terms of the datasets. However, the low recall rate has caused a significant problem: the classifier is not performing well in identifying minority class patterns even though it has performed well in feature choice.

This shows that the feature relevance is not a direct indication of discriminative power under class imbalance. Hence, future enhancement should not be limited to feature engineering but learning strategies should be given emphasis.

DISCUSSION

The results show that feature selection can enhance the interpretability of the model and reduce the quantity of features, but it does not necessarily solve class imbalance problems. This suggests a lack of correspondence between the importance of a feature and the influence of the feature on the classification.

Furthermore, some of the features selected are uniform across the datasets, indicating that certain aspects of the structure software are predictive. The inability in detecting defective instances, however, highlights the need for the combination of feature selection with imbalance-aware learning mechanisms. This indicates that feature selection is not enough to boost the early defect prediction performance. Rather, it should be aided by sophisticated learning strategies which explicitly tackle class imbalance and boost the detection sensitivity.

Moreover, the proposed consensus-based ranking approach is more stable and reliable than the single method-based approach to feature selection. The framework combines multiple perspectives, minimising bias and enhancing generalisation over datasets.

7.1Statistical Significance Analysis

Non-parametric statistical tests such as the Wilcoxon signed-rank test are commonly used for performance comparison in machine learning studies [17]. In an effort to validate the differences observed in the performance in a rigorous fashion, the “Wilcoxon signed-rank test” was used for multiple datasets (KC1, KC2, PC1, JM1) for precision and recall values.

The null hypothesis is that there will be no important difference between precision and re-call distributions. The test result led to the $p = 0.012$ ($p < 0.05$), which denied the null hypothesis at the significance level $\alpha = 0.05$. Additionally, the effect size was calculated as $r = 0.65$, which suggests that there is a major variance between the two measurements.

The results indicate that the model is indeed biased towards precision over recall, mainly because of the class imbalance. This significant difference in statistics proves the importance of considering imbalance-aware learning strategies for enhancing defect detection performance.

7. Threats to Validity

Several factors may affect the validity of the results presented in this study.

Internal Validity: The results depend on the chosen classifier (Logistic Regression) and parameter settings. Different models may yield different performance outcomes.

External Validity: The study is conducted on PROMISE datasets, which may not entirely represent all real-world software systems. Generalisation to industrial environments requires further validation.

Construct Validity: The selected metrics may not capture all aspects of software quality. Additional dynamic or process

metrics could further improve prediction performance.
Conclusion Validity: The assessment is based on a limited number of datasets and metrics. More extensive experimentation is required for stronger statistical conclusions

8. CONCLUSION AND FUTURE WORK

This study portrays a strong and generalizable multi-method feature selection framework for early software defect prediction. The integration of heterogeneous selection techniques with consensus-based ranking significantly improves feature stability across datasets.

However, experimental findings reveal that feature robustness alone is insufficient to deliver the tasks posed by class imbalance. Future research should spotlight hybrid frameworks that integrate feature selection with advanced learning paradigms, including ensemble methods, cost-sensitive optimisation, and deep learning. Despite its effectiveness, the proposed framework has certain weaknesses. The study relies on static code metrics and does not incorporate dynamic or process-based features. Additionally, only a single classifier is extensively evaluated, which may limit generalizability across learning paradigms.

Future work will focus on enhancing prediction performance through:

•Integration of imbalance-aware learning techniques (e.g., SMOTE, cost-sensitive learning)
•Exploration of ensemble and deep learning models
•Cross-project validation for improved generalisation
•Incorporation of explainable AI techniques for better interpretability

The proposed framework provides a strong foundation for developing more reliable and generalizable early defect prediction systems. Future work will explore hybrid frameworks that integrate feature selection with deep learning and cost-sensitive ensemble models. Additionally, explainable AI techniques will be incorporated to enhance interpretability and trust in defect prediction systems.

REFERENCES

1. Menzies T, Milton Z, Turhan B et al (2010) Defect prediction from static code features: current results, limitations, new approaches. Empir Softw Eng 15(4):375–407
2. Hall T, Beecham S, Bowes D et al (2012) A systematic literature review on fault prediction performance in software engineering. IEEE Trans Softw Eng 38(6):1276–1304
3. Wang S, Liu T, Tan L (2016). Automatically learning semantic features for defect prediction. In: Proc IEEE/ACM ICSE, pp 297–308
4. Dam HK, Pham T, Ng SW et al (2018) A deep tree-based model for software defect prediction. IEEE Trans Softw Eng
5. Li J, Cheng K, Wang S et al (2018) Feature selection: a data perspective. ACM Comput Surv 50(6):94
6. Zhou Y, Liu Y (2019) Deep learning for software defect prediction: a survey. IEEE Softw 36(5):34–45
7. Bennin KE, Keung J, Monden A (2020) Defect prediction using machine learning: a systematic literature review. Empir Softw Eng 25:1–35
8. Herbold S (2020) Cross-project defect prediction: a systematic mapping study. IEEE Trans Softw Eng 46(2):234–257
9. Zhang F, Keivanloo I, Zou Y (2021) Data transformation in cross-project defect prediction. Inf Softw Technol 137:106588
10. Liu W, Zhang H, Wang X (2023) Handling class imbalance in software defect prediction using hybrid sampling. Expert Syst Appl 213:119042
11. Mallik S, Pradhan D (2024) Hybrid feature selection using metaheuristic optimization for software defect prediction. Softw Qual J
12. Nasser A, Ghanem W (2024) Adaptive feature selection framework for improving defect pre-diction. Expert Syst Appl
13. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
14. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
15. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: Synthetic minority over-sampling technique. J Artif Intell Res 16:321–357
16. Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: Machine learning in Python. J Mach Learn Res 12:2825–2830
17. Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30