


Article

Crashing Fault Residence Prediction Using a Hybrid Feature Selection Framework from Multi-Source Data

Xiao Liu ^{1,2} , Xianmei Fang ³, Song Sun ^{1,4}, Yangchun Gao ¹, Dan Yang ⁵ and Meng Yan ^{2,*}¹ State Key Laboratory of Intelligent Vehicle Safety Technology, Chongqing 400023, China;

lxsgdsgg@gmail.com (X.L.); 20220033@cqu.edu.cn (S.S.); gaoyangchun@caeri.com.cn (Y.G.)

² School of Big Data and Software Engineering, Chongqing University, Chongqing 400044, China³ School of Big Data and Computer Science, Hechi University, Hechi 546399, China; fangxianmei@hcnu.edu.cn⁴ School of Computer and Information Science, Chongqing Normal University, Chongqing 400044, China⁵ Southwest Jiaotong University, Chengdu 610032, China; dyang@cqu.edu.cn

* Correspondence: mengy@cqu.edu.cn

Abstract: The inherent complexity of modern software frequently leads to critical issues such as defects, performance degradation, and system failures. Among these, system crashes pose a severe threat to reliability, as they demand rapid fault localization to minimize downtime and restore functionality. A critical step of fault localization is predicting the residence of crashing faults, which involves determining whether a fault is located within the stack trace or outside it. This task plays a crucial role in software quality assurance by enhancing debugging efficiency and reducing testing costs. This study introduces SCM, a two-stage composite feature selection framework designed to address this challenge. The SCM framework integrates spectral clustering for feature grouping, which organizes highly correlated features into clusters while reducing redundancy and capturing non-linear relationships. Maximal information coefficient analysis is then applied to rank features within each cluster and select the most relevant ones, forming an optimized feature subset. A decision tree classifier is then applied to predict the residence of crashing faults. Extensive experiments on seven open-source software projects show that the SCM framework outperforms seven baseline methods, which include four classifiers and three ranking approaches, across four evaluation metrics such as F-measure, g-mean, MCC, and AUC. These results highlight its potential in improving fault localization.



Academic Editor: Andrea Prati

Received: 15 December 2024

Revised: 25 January 2025

Accepted: 28 January 2025

Published: 28 February 2025

Citation: Liu, X.; Fang, X.; Sun, S.; Gao, Y.; Yang, D.; Yan, M. Crashing Fault Residence Prediction Using a Hybrid Feature Selection Framework from Multi-Source Data. *Appl. Sci.* **2025**, *15*, 2635. <https://doi.org/10.3390/app15052635>

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

Keywords: crashing fault; feature selection; spectral clustering; maximal information coefficient

1. Introduction

In today's technology-driven world, software plays a pivotal role in daily life, yet faults in software systems remain unavoidable during development. These faults often compromise user requirements and may lead to system crashes, known as crashing faults. Identifying and resolving these faults efficiently has become a cornerstone of software quality assurance [1].

Crash reports, automatically generated by the system, provide crucial insights into the conditions leading to a program crash. These reports typically include a stack trace, which documents the sequence of function calls executed prior to the crash. Correctly interpreting the stack trace is vital for developers and testers to diagnose and locate the root cause of the fault. The stack trace is composed of multiple frames, where the first frame identifies the exception type, and subsequent frames represent details about the called functions. Each frame includes critical information, such as the name of the class being called, the name of the function, and the specific line number [2], collectively forming a diagnostic tuple.

When the fault matches one of the frames in the stack trace, debugging is significantly streamlined, allowing developers to focus their inspection on a manageable number of lines of code. This scenario is referred to as the fault being inside the stack trace. However, when the root cause resides outside the stack trace, developers must explore broader sections of the program, encompassing functions related to the entire call sequence [3]. Such cases demand considerably more resources, including time and effort, to identify and rectify the fault. This distinction between faults located inside or outside the stack trace has increasingly attracted the attention of researchers and practitioners. As a result, accurately predicting the residence of crashing faults has emerged as a critical software quality assurance activity within the domain of Software Engineering (SE).

Gu et al. [3] made the first attempt to study the issue of predicting the crashing fault residence and later released their benchmark dataset. They simulated some crash instances and collected a total of 89 features to characterize them. Actually, predicting the residence of the crashing faults is a binary classification task, which uses a machine learning (ML) model to determine whether a crashing fault instance is within the stack trace or not. Just like general ML applications, the performance of the used model is affected by the feature quality of the dataset. When the data contain many features, irrelevant and redundant features may negatively affect model performance. Thus, in order to obtain satisfactory performance for this task, the crash data need to be preprocessed for obtaining high-quality feature representation.

In this paper, we focus on the challenge of predicting fault residence, particularly addressing the need for reducing redundant and irrelevant features in crash data to enhance classification performance. To tackle this challenge, we propose a novel two-stage composite feature selection framework to reserve the representative features. More specifically, the hybrid framework integrates feature clustering and relevance ranking to select an optimal feature subset. This hybrid framework includes two stages: first, the Spectral Clustering (SC) method [4,5] is used to cluster the original features into multiple groups; then, the Maximal Information Coefficient (MIC) [6] is employed to rank the features using their MIC values in each cluster. The feature with the highest MIC value in each cluster is selected as the candidate member of the final feature subset. This selected feature subset has low redundancy among its features and a strong relationship with the crash labels. Since the hybrid feature selection framework combines the SC method for feature clustering and the MIC method for feature relevance analysis, we refer to the proposed method as SCM. After the two stages, we use a decision tree classifier to build the classification model based on the selected feature subset for residence prediction of the crashing faults.

To measure the prediction performance of our proposed SCM framework for the residence of crashing faults, we used a dataset that contains seven open-source software projects denoted by Gu et al. [3] and employed four indicators for performance evaluation. The experimental results elaborate that our proposed SCM framework achieves an average F-measure of 0.643, average g-mean of 0.759, MCC of 0.569, and average AUC of 0.780. Our proposed SCM framework consistently outperforms eight comparative methods across all projects and all four evaluation metrics.

This work makes two main contributions. First, we develop a novel two-stage composite framework, SCM, which selects the most representative features from the crash data to predict the residence of crashing faults. Second, we conduct experiments on seven software projects from an open-source benchmark dataset, using four indicators to measure the effectiveness of our proposed SCM framework.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 describes the details of our proposed SCM-based hybrid feature selection framework. Section 4 describes the dataset, the performance indicators, the data partition,

and the parameter settings. Section 5 reports the detailed experimental results. Finally, Section 6 concludes our work and presents future work.

2. Related Work

For the task of predicting the residence of crashing faults, Gu et al. [3] were the first to investigate this topic. Based on their denoted benchmark dataset, Xu et al. [7] proposed employing an advanced transfer learning method called balanced distribution adaptation to conduct the crashing fault residence prediction task under the cross-project scenario. Moreover, they extended this work by proposing a novel feature selection and embedding framework, which added an information-gain-based selection operation and used a weighting strategy to address the class imbalanced characteristic of the initially collected crash data. Xu et al. [8] applied imbalanced metric learning to the crash data, aiming to learn a new feature representation with higher quality. This method aims to simultaneously reduce the distance between crash instances which have consistent labels and increase the distance between crash instances which have distinct labels. In 2021, Zhao et al. [9] conducted a comprehensive comparative analysis to investigate the impact of more than 20 feature selection strategies from four families with more than 20 classifiers on the model performance of predicting the residence of crashing faults. The studies' feature selection strategies are typical filter-based or wrapper-based ones. Large-scale experiments showed that there exist significant differences in the performance of these feature selection strategies across different classifiers and software projects.

For the applications of feature selection methods, researchers have employed such methods on many tasks in the field of SE. For the defect prediction task, which predicts defective software modules, Song et al. [10] pointed out that feature selection is an important process in the general framework of software defect prediction. Liu et al. [11] proposed a self-designed feature selection framework to identify irrelevant and redundant features. Rehman et al. [12] proposed a BCF-WASPAS approach, integrating Einstein operators with bipolar complex fuzzy sets, enhancing feature selection for software defect prediction by jointly addressing positive and negative selection criteria. Malhotra et al. [13] developed a multi-filter wrapper feature selection approach to further improve defect prediction performance. Xu et al. [14] assessed the impact of over 30 feature selection strategies with random forest, and Ghotra et al. [15] further expanded this analysis using 21 classifiers, confirming the value of feature selection for defect prediction. For the software effort estimation task, which estimates the amount of consumed resources in the software development process, Ali et al. [16] conducted an empirical study on six bio-inspired and four non-bio-inspired selection methods, with genetic algorithms and harmony search showing the best performance. Liu et al. [17] presented a neighborhood mutual information-based method and achieved a 24% average improvement on five out of six datasets when compared with three representative feature selection methods. Goyal et al. [18] introduced a neighborhood component analysis-based approach with multi-layer perceptrons to enhance accuracy on the Desharnais dataset. Hosni et al. [19] observed that three filter-based methods outperformed random selection in homogeneous ensemble models. In addition, feature selection methods are also applied to other SE tasks, such as the software product line [20,21], code smell detection [22,23], and fault localization [24–26].

Our work draws inspiration from prior studies. However, our approach distinguishes itself by introducing a two-stage composite framework (SCM) that integrates feature clustering and relevance ranking to optimize feature representation. By grouping highly correlated features, this unified framework reduces redundancy while simultaneously enhancing feature relevance, effectively addressing the challenges of crashing fault residence prediction [27].

3. The Proposed Feature Selection Framework

3.1. Overview

The overview of our proposed SCM framework, which contains a **Feature Clustering** step and a **Feature Ranking** step, is depicted in Figure 1. In the **Feature Clustering** step, SC groups highly correlated features into clusters by capturing both linear and non-linear relationships among features. This clustering not only reduces feature redundancy but also provides a structured foundation for further analysis. In the **Feature Ranking** step, the grouped features are first normalized to ensure consistency in scale. This preprocessing step prevents features with larger ranges from disproportionately influencing the MIC computation, enabling fair comparisons across features within each cluster. MIC is then used to evaluate the relevance of each feature to the target labels, and the most representative feature from each cluster is selected to form the final optimized feature subset.

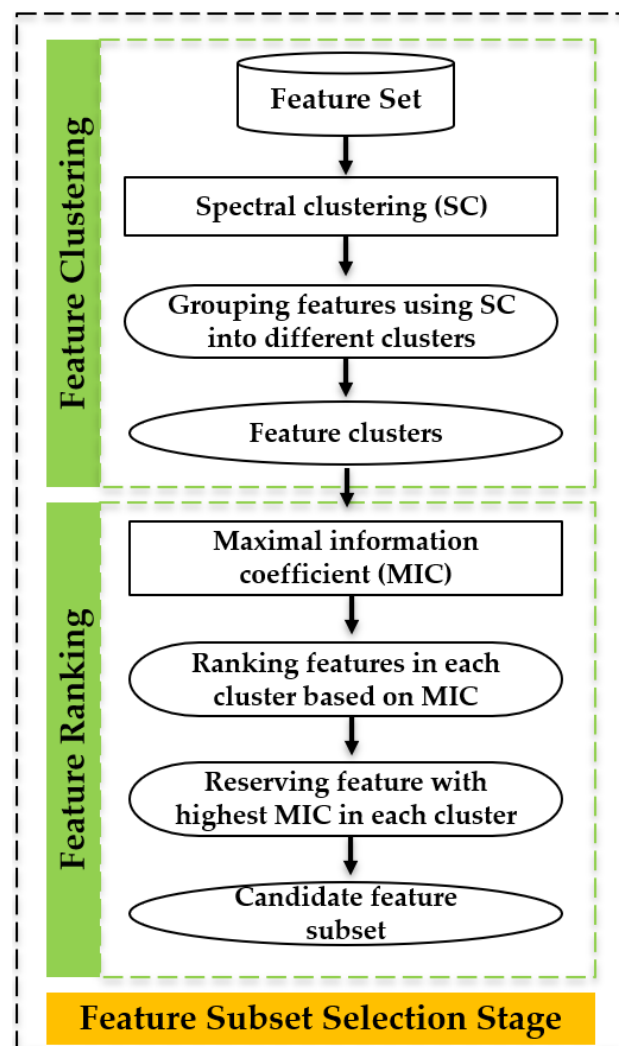


Figure 1. An overview of our SCM framework.

The interaction between these two components is crucial: **Feature Clustering** reduces the complexity of the data by grouping related features, while **Feature Ranking** refines these groups by isolating the most impactful features. This collaborative process produces an optimized feature subset, which is then input into the decision tree classifier to predict whether the residence of a crashing fault is inside or outside the stack trace.

Detailed descriptions of the SC and MIC methods are provided in Sections 3.2 and 3.3, respectively, as both are integral to the proposed SCM framework.

3.2. Feature Clustering with SC

The aim of the feature clustering step is to group highly correlated features while minimizing redundancy across groups. Traditional clustering methods like k-means, which assume spherical data distributions, often struggle to capture complex, non-linear relationships among crash data features. To achieve this, we adopted the spectral clustering (SC) algorithm [4,5], a graph-based algorithm capable of capturing non-linear correlations through pairwise similarity analysis. Specifically, SC divides the original feature set into multiple clusters, where features within the same cluster exhibit high correlation, whereas features in different clusters are minimally overlapping. The technical details of SC are elaborated below.

Assuming the feature set of the crash data as $F(f_1, f_2, \dots, f_n)$ (n is the number of features), we construct an undirected graph $G(V, E)$, in which V is a set of points (v_1, v_2, \dots, v_n) corresponding to each feature and E is a set of edges (e_1, e_2, \dots, e_n) . Considering one point pair (v_i, v_j) , we define w_{ij} as the weight of the edge connecting v_i and v_j . If there is a connection between v_i and v_j , we set $w_{ij} > 0$, otherwise, $w_{ij} = 0$.

To quantify the similarity between features, we first establish the adjacency matrix W by applying the fully connected technique in which the weights between all point pairs are non-negative. The Gaussian radial basis function was chosen to define the weight of the pair, offering a continuous and differentiable metric for quantifying similarity, as shown in Equation (1).

$$w_{ij} = \exp\left(-\frac{\|v_i - v_j\|_2^2}{2\delta^2}\right) \quad (1)$$

where δ is a free parameter denoting the width of the Gaussian kernel and $\|\cdot\|_2$ denotes the L_2 -norm. δ controls the scale of similarity measurement, where smaller δ values emphasize local relationships by assigning higher weights to closer features, while larger δ values capture broader relationships with reduced local sensitivity. In this study, δ was selected through cross-validation to balance local and global feature relationships. This function ensures that the adjacency matrix W effectively represents the relationships between features in the feature space.

Based on the adjacency matrix W , we compute a degree matrix D by defining degree d_i corresponding to v_i as $d_i = \sum_{j=1}^n w_{ij}$. This degree matrix reflects the overall connectivity of each feature within the graph, summarizing the influence of each node.

Next, using D and W , we derive the unnormalized Laplacian matrix, $L = D - W$, and the normalized Laplacian matrix, $L_{norm} = D^{-1/2}LD^{-1/2}$. The normalized Laplacian matrix ensures that features with varying connectivity are appropriately scaled, making the clustering results less sensitive to the magnitude of node degrees.

Subsequently, to obtain the clustering results, SC adopts the idea of graph cutting and makes the weight of the edges between different subgraphs as low as possible; meanwhile, the weight of inner edges of the same subgraph is as high as possible. Different subgraphs represent different groups, and points in the same subgraph represent features in the same group with a higher similarity. For each feature, $f_i (i = 1, 2, \dots, n)$, we have the formula shown in Equation (2).

$$f^T L_{norm} f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} \left(f_i / \sqrt{d_i} - f_j / \sqrt{d_j} \right)^2 \quad (2)$$

The eigenvectors corresponding to the smallest s eigenvalues capture the most significant structural information of the graph, allowing the features to be effectively projected into a reduced s -dimension. These eigenvectors are then used to construct the final eigenmatrix, $M \in \mathcal{R}^{n \times s}$. Finally, the k -means algorithm is applied to the rows of M , with each

row representing a feature in the reduced s -dimension space. This approach ensures that the clustering is performed on features with preserved pairwise similarities, as encoded by the graph.

3.3. Feature Ranking with MIC

The goal of the feature ranking stage is to select the feature in each group that is most relevant to the class label and discard the redundant ones. The key to this step is to precisely measure the correlation between the crash data features and the corresponding instance labels. To achieve this, we employ MIC [6], a robust statistical measure capable of capturing both linear and non-linear dependencies. By ranking features within each cluster based on MIC values and retaining the top-ranked feature, we ensure the final subset preserves high relevance to fault residence while eliminating redundant features.

MIC was chosen for its ability to detect both linear and non-linear dependencies, making it particularly suitable for crash data with potentially complex relationships. While Pearson's correlation coefficient effectively captures linear relationships, it may fail to identify non-linear patterns. In comparison, MIC normalizes its values within the range $[0, 1]$, enhancing interpretability and facilitating feature ranking. These qualities make MIC a strong choice for identifying relevant features in diverse and high-dimensional datasets.

To better understand its suitability, MIC was originally designed to measure the dependence between two variables. For a given dataset S with m samples, let the feature set and the corresponding label set be represented as two variables, P and Q , individually. If a relevance between P and Q exists, then we can draw a grid on their scatter plot. Given that the value of two variables is divided into p bins and q bins individually, we construct a p -by- q grid, G . We denote S_G as the probability distribution of the number of points falling into the grid G in S . The probability of each cell of G is determined by the proportion of points falling inside it in all points. The maximum mutual information of S_G is denoted as shown in Equation (3).

$$I_S^*(p, q) = \max I_{S_G}(p, q) \quad (3)$$

where $I_{S_G}(p, q)$ denotes the mutual information of S_G , and $I_S^*(p, q)$ is the maximum value of $I_{S_G}(p, q)$ among all cells in a specific grid. Equation (4) provides the definition of mutual information.

$$I(P; Q) = H(P) + H(Q) - H(P, Q) \quad (4)$$

where $H(P)$ and $H(Q)$ represent the entropy of random variables P and Q individually, and $H(P, Q)$ denotes the joint entropy between P and Q .

For different options of p and q , we obtain multiple values for $I_S^*(p, q)$. In this case, a characteristic matrix $M(S)$ is constructed, as defined in Equation (5).

$$M(S) = \frac{I_S^*(p, q)}{\log \min\{p, q\}} \quad (5)$$

The goal of Equation (5) is to restrict the matrix values between 0 and 1 to reduce the differences of the dimensions. Then, the MIC value can be calculated as shown in Equation (6).

$$\text{MIC}(S) = \max_{pq < B(m)} M(S) \quad (6)$$

where $B(m)$ represents the upper bound of the grid size. We set $B(m)$ as $m^{0.6}$, following the same setting in previous work [6]. The choice of $m^{0.6}$ balances computational efficiency and granularity, ensuring that the dependencies are adequately represented without excessive computational overhead. In each feature group clustered in the first step, we computed the MIC value for the corresponding features and reserved the one with the highest MIC value

by adding it into the final feature subset. Thus, we obtained an optimal feature set with k features, in which these remaining features have low redundancy with each other but are highly correlated with the class labels. While MIC is robust for capturing feature–label dependencies, its theoretical complexity is $O(B(m) \cdot m^2)$ per feature. For large datasets, this complexity can become prohibitive. To address this, we reduce the feature space through clustering, significantly limiting the number of feature pairs requiring MIC computation. Additionally, we leverage parallel computing to accelerate the MIC calculation, improving the efficiency and scalability of the process within the dataset sizes used in this study.

At the end of the two-stage feature selection process, the most representative feature from each cluster is selected based on its MIC value, forming the final optimized feature subset. This subset, with reduced redundancy and high relevance to the class labels, is then used as input for the decision tree classifier. The classifier leverages these selected features to construct a model for predicting the residence of crashing faults efficiently.

4. Experimental Setup

4.1. Dataset

To evaluate the effectiveness of our proposed SCM feature selection framework in improving the prediction performance for crashing fault residence, we used a widely recognized benchmark dataset released by Gu et al. [3]. This dataset contains crashing fault instances from seven Java-based software projects: Apache Commons **Codec**, Apache Commons **Collections**, Apache Commons **IO**, **Jsoup**, **JsoupParser**, **Mango**, and **Ormlite-Core**.

The Apache Commons **Codec** provides various encoding and decoding techniques, including Base64, Hex, Phonetic, and URL-related methods. Apache Commons **Collections** offers numerous enhancements and additional functionalities for working with Java collections in the JDK. Apache Commons **IO** supports Java IO operations by providing a variety of utility classes and methods. **Jsoup** is an HTML parser library designed for manipulating and extracting data from real-world HTML files. **JsoupParser** parses SQL statements and translates them into hierarchical Java class structures. **Mango** is a high-performance, distributed framework aimed at facilitating object-relational mapping. Finally, **Ormlite-Core** serves as the core component of Ormlite, offering a lightweight parser for mapping Java objects to SQL databases.

The basic information of these projects is reported in Table 1, in which ‘LoC’ denotes the number of lines of code in the project, and ‘No. Classes’ represents the total number of classes in the project. The crash fault instances in these projects are also summarized as follows: ‘No. Crashes’ indicates the total number of crashing fault samples, while ‘No. InT’ and ‘No. OutT’ represent the number of samples located inside and outside the stack trace, respectively. Additionally, ‘% R’ denotes the ratio of samples located inside the stack trace to those located outside, providing insight into the distribution of faults.

Table 1. The basic statistics of seven projects.

Project	Domain	LoC	No. Classes	No. Crashes	No. InT	No. OutT	% R
Codec	Utility	14,480	84	610	177	433	29.0%
Collections	Utility	61,283	435	1350	273	1077	20.2%
IO	I/O	26,018	122	686	149	537	21.7%
Jsoup	Utility	15,460	137	601	120	481	20.0%
JsoupParser	Utility	32,868	203	647	61	586	9.4%
Mango	Database	30,208	475	733	53	680	7.2%
Ormlite-Core	Database	20,240	175	1303	326	977	25.0%

There are three steps to collect this dataset, including crash generation by simulation with a mutation tool, feature extraction, and crash instance label assignment. The extracted

features come from five families, which are briefly described as follows: 11 features characterizing the stack trace and 23 features from the class or function in the upper frame, including the number of local variables, fields (except constructor functions), and imported packages of the class in the upper frame; whether the class in the upper frame is inherited from others; the line of code of the class in the upper frame; the line of code of the function in the upper frame; and the quantity of parameters, local variables, if-statements, loops, for statements, for-each statements, while statements, do-while statements, try blocks, catch blocks, finally blocks, assignment statements, function calls, return statements, unary operators, and binary operators of the function in the upper frame. Similarly, there are 23 features from the class or function in the nether frame. Additionally, there are 16 features which are the features of the functions in the upper frame normalized by LoC and 16 features which are the features of the functions in the nether frame normalized by LoC.

4.2. Performance Measurement

Since our work is a binary classification problem, aiming to predict if a crashing instance is inside or outside the stack trace, we employed four metrics, also referred to as indicators, including F-measure, g-mean, Matthews Correlation Coefficient (MCC), and Area Under the Receiver Operating Characteristic Curve (AUC) to evaluate the effectiveness of our proposed SCM feature selection framework. The four concepts are defined as follows: the quantity of crashing fault samples whose true labels and predicted labels are both 'InTrace' is referred to as True Positive (TP), the quantity of samples whose true labels are 'OutTrace' but predicted labels are 'InTrace' is referred to as False Positive (FP), the quantity of samples whose true labels are 'InTrace' but predicted label are 'OutTrace' is referred to as False Negative (FN), and the quantity of samples whose true labels and predicted labels are both 'OutTrace' is referred to as True Negative (TN). The F-measure for crashing fault samples is a trade-off between Precision ($Precision = \frac{TP}{TP+FP}$) and Recall ($Recall = \frac{TP}{TP+FN}$); its formula is provided in Equation (7).

$$F\text{-measure} = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (7)$$

F-measure ranges from 0 to 1, with higher values indicating better performance.

Additionally, the g-mean is another important metric used to evaluate model performance on imbalanced data, and it is defined in Equation (8).

$$g\text{-mean} = \sqrt{\frac{TP}{TP+FN} \times \frac{TN}{TN+FP}} \quad (8)$$

g-mean also ranges from 0 to 1, where higher values indicate better performance.

The indicator MCC is defined in Equation (9).

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (9)$$

MCC ranges from −1 to 1, with higher values indicating better performance.

AUC is defined as the area formed under the receiver operating characteristic (ROC) curve with the coordinate axes. The ROC curve can easily detect the effect of an arbitrary threshold on the generalization performance of the model. The probabilistic significance of AUC is that, when a positive instance and a negative instance are randomly taken out and put into the classifier to discriminate the corresponding probability of positive output, then AUC is the probability that the positive instance is positive is greater than the probability that the negative instance is positive. In general, an AUC greater than 0.5 indicates that the

classifier achieves performance exceeding random guessing and effectively distinguishes between classes.

4.3. Generation of Training and Test Sets

We employed the stratified sampling strategy to generate the training and test set. To be more specific, for each software project in the benchmark dataset, half of the crashing fault samples whose labels are ‘InTrace’ and ‘OutTrace’ are randomly selected to form the training set, whereas the test set is made up with the remaining samples. This sampling strategy keeps the proportions of the two types of crash instances in the two sets consistent with that of the original data. To minimize the effect of random bias and increase the robustness of the results, we repeated this process 30 times and recorded the average values for each corresponding indicator.

4.4. Parameter Assignment

In our experiment, a parameter is needed to be specified, i.e., the quantity of groups k in the step of feature clustering analysis. As suggested by prior research [28,29], retaining 15% of the original features made the model achieve the same or even better prediction performance than the model using original features for the SE task; therefore, in this work, we set k as $89 \times 15\% = 14$.

4.5. Research Questions

We formalize our study with the following two research questions:

RQ1: Does our proposed SMC feature selection framework using a decision tree classifier achieve better prediction performance than it does with other classifiers?

After obtaining the reduced feature set via the feature selection stage, we input crash data with the reserved features into a decision tree classifier to build the model for predicting the crashing fault. This research question evaluates whether the SCM framework, when paired with a decision tree classifier, outperforms its integration with other widely employed classifiers for predicting crashing fault residence.

RQ2: Does the MIC-based feature ranking in our proposed feature selection framework perform better than other feature ranking strategies?

Our proposed feature selection framework contains two stages, i.e., the SC-based feature clustering and MIC-based feature ranking. According to the description of the framework, the reserved features are largely determined by the ranking results of the features in each cluster. Thus, the feature ranking strategy is important for the reduced feature set, which is related to the prediction performance for crashing fault residence. We designed this second RQ to examine whether the MIC-based feature ranking strategy used in our SCM hybrid feature selection framework performs better than other feature ranking strategies.

5. Evaluation

In this section, we aim to address the aforementioned two research questions. In RQ1, we examine whether our proposed SCM feature selection framework, combined with a decision tree classifier, outperforms four other widely employed classifiers. In RQ2, we investigate whether incorporating the MIC-based feature ranking strategy within the SCM framework yields better performance than three other commonly used ranking strategies.

5.1. RQ1: Does Our Proposed SMC Feature Selection Framework Using a Decision Tree Classifier Achieve Better Prediction Performance than It Does with Other Classifiers?

Methods: We carefully selected four classifiers that are widely employed in the tasks of SE for comparison, including the support vector machine (SVM) classifier [30], logistic regression (LR) classifier [31], random forest (RF) classifier [32], and k-nearest neighbor

(*k*NN) classifier [33]. More specifically, SVM is a generalized linear classification model, LR is a statistic-based classification model, RF is an ensemble-based classification model, *k*NN is a sample-based classification model. In our study, DT is classified as an information theory-based classification model. In other words, these classifiers come from different families. In addition, these classifiers have been widely used in previous studies [9,34]. Before applying our SCM framework to different classifiers, the data was standardized using the widely adopted z-score technique [5].

Results: Tables 2–5 report the results of F-measure, g-mean, MCC, and AUC values, respectively, for our proposed SCM feature selection framework using five different classifiers. These five classifiers are abbreviated as S, L, R, *k*N, and D in the tables. In the four tables, we discuss the following findings:

Table 2. The F-measure values of our proposed SCM feature selection framework, using five different classifiers, for each software project.

Projects	SCM-S	SCM-L	SCM-R	SCM- <i>k</i> N	SCM-D
Codec	0.380	0.287	0.018	0.511	0.663
Collections	0.485	0.362	0.013	0.526	0.707
IO	0.547	0.535	0.294	0.602	0.721
Jsoup	0.027	0.087	0.001	0.142	0.382
JsoupParser	0.728	0.722	0.720	0.696	0.687
Mango	0.560	0.393	0.098	0.478	0.503
Ormlite-Core	0.575	0.536	0.230	0.624	0.840
Average	0.472	0.417	0.196	0.511	0.643

Table 3. The g-mean values of our proposed SCM feature selection framework, using five different classifiers, for each software project.

Projects	SCM-S	SCM-L	SCM-R	SCM- <i>k</i> N	SCM-D
Codec	0.503	0.426	0.050	0.631	0.755
Collections	0.585	0.493	0.045	0.636	0.803
IO	0.644	0.638	0.413	0.698	0.803
Jsoup	0.078	0.205	0.004	0.274	0.563
JsoupParser	0.771	0.793	0.753	0.757	0.786
Mango	0.634	0.518	0.147	0.582	0.716
Ormlite-Core	0.659	0.632	0.348	0.722	0.886
Average	0.553	0.529	0.251	0.614	0.759

Table 4. The MCC values of our proposed SCM feature selection framework, using five different classifiers, for each software project.

Projects	SCM-S	SCM-L	SCM-R	SCM- <i>k</i> N	SCM-D
Codec	0.292	0.179	0.029	0.342	0.529
Collections	0.461	0.318	0.040	0.459	0.638
IO	0.490	0.468	0.353	0.527	0.651
Jsoup	0.047	0.048	0.002	0.112	0.231
JsoupParser	0.728	0.706	0.733	0.691	0.663
Mango	0.595	0.420	0.142	0.506	0.479
Ormlite-Core	0.512	0.462	0.290	0.519	0.789
Average	0.446	0.372	0.227	0.451	0.569

Table 5. The AUC values of our proposed SCM feature selection framework, using five different classifiers, on each software project.

Projects	SCM-S	SCM-L	SCM-R	SCM-kN	SCM-D
Codec	0.607	0.564	0.504	0.661	0.764
Collections	0.664	0.609	0.503	0.689	0.815
IO	0.698	0.691	0.587	0.731	0.814
Jsoup	0.506	0.513	0.500	0.529	0.612
JsoupParser	0.797	0.813	0.785	0.786	0.817
Mango	0.704	0.636	0.528	0.671	0.751
Ormlite-Core	0.707	0.686	0.567	0.743	0.888
Average	0.669	0.645	0.568	0.687	0.780

F-measure results: As shown in Table 2, our proposed SCM feature selection framework, using a DT classifier, achieves the best F-measure on five of the seven projects and the best average F-measure across all projects, compared with SCM using the other four classifiers. More specifically, SCM with DT improves the average F-measure by 0.171, 0.226, 0.447, and 0.132 compared to SCM with SVM, LR, RF, and k NN, respectively.

g-mean results: As shown in Table 3, our proposed SCM feature selection framework, using a DT classifier, achieves the best g-mean on six of the seven projects and the best average g-mean across all projects, compared with SCM using the other four classifiers. More specifically, SCM with DT improves the average F-measure by 0.206, 0.230, 0.508, and 0.145 compared to SCM with SVM, LR, RF, and k NN, respectively.

MCC results: As shown in Table 4, our proposed SCM feature selection framework, using a DT classifier, achieves the best MCC on five of the seven projects and the best average MCC across all projects, compared with SCM using other four classifiers. More specifically, SCM with DT improves the average F-measure by 0.123, 0.197, 0.342, and 0.118 compared to SCM with SVM, LR, RF, and k NN, respectively.

AUC results: As shown in Table 5, our proposed SCM feature selection framework, using a DT classifier, achieves the best AUC on all seven projects and the best average AUC across all projects, compared to SCM with the other four classifiers. More specifically, SCM with DT improves the average F-measure by 0.111, 0.135, 0.212, 0.093 compared to SCM with SVM, LR, RF, and k NN, respectively.

Answer: Our proposed SCM feature selection framework using a DT classifier performs markedly better than the comparative classifiers for predicting the residences of crashing faults in terms of all four performance indicators. Thus, it is reasonable to choose DT as our classification model to predict crashing fault residence for better performance.

5.2. RQ2: Does the MIC-Based Feature Ranking in Our Proposed Feature Selection Framework Perform Better than Other Feature Ranking Strategies?

Methods: The essence of the feature ranking strategy is to calculate the correlation between each feature and the class label using different criteria. For this question, we carefully selected three classic feature correlation calculation methods to combine with the same SC-based feature clustering for comparison, i.e., chi-square (CS), information gain (IG), and ReliefF (ReF). More specifically, CS is a statistics-based method, IG is a probability-based method, ReF is an instance-based method, and MIC is an information-theory-based method. For a fair comparison, we just replaced MIC with the baseline method in our hybrid feature selection framework. And the DT classifier was used as the classification model for all methods, including the three baseline methods.

Results: Tables 6–9 report the result of F-measure, g-mean, MCC, and AUC values, respectively, for SC-based feature clustering with four different feature ranking strategies. In the four tables, we discuss the following findings:

Table 6. The F-measure values of SC-based feature clustering with different feature ranking strategies using a decision tree classifier across each software project.

Projects	SCCS	SCIG	SCReF	SCM
Codec	0.459	0.647	0.426	0.663
Collections	0.448	0.606	0.446	0.707
IO	0.632	0.707	0.637	0.721
Jsoup	0.354	0.345	0.366	0.382
JsoupParser	0.709	0.666	0.610	0.687
Mango	0.372	0.516	0.397	0.503
Ormlite-Core	0.627	0.832	0.639	0.840
Average	0.514	0.617	0.503	0.643

Table 7. The g-mean values of SC-based feature clustering with different feature ranking strategies using a decision tree classifier across each software project.

Projects	SCCS	SCIG	SCReF	SCM
Codec	0.593	0.740	0.562	0.755
Collections	0.608	0.702	0.600	0.803
IO	0.739	0.794	0.737	0.803
Jsoup	0.544	0.533	0.548	0.563
JsoupParser	0.817	0.784	0.728	0.786
Mango	0.559	0.704	0.559	0.716
Ormlite-Core	0.733	0.882	0.741	0.886
Average	0.656	0.734	0.639	0.759

Table 8. The MCC values of SC-based feature clustering with different feature ranking strategies using a decision tree classifier across each software project.

Projects	SCCS	SCIG	SCReF	SCM
Codec	0.256	0.509	0.235	0.529
Collections	0.320	0.636	0.332	0.638
IO	0.541	0.634	0.554	0.651
Jsoup	0.192	0.187	0.218	0.231
JsoupParser	0.688	0.640	0.585	0.663
Mango	0.346	0.490	0.382	0.479
Ormlite-Core	0.514	0.677	0.529	0.789
Average	0.408	0.539	0.405	0.569

Table 9. The AUC values of SC-based feature clustering with different feature ranking strategies using a decision tree classifier across each software project.

Projects	SCCS	SCIG	SCReF	SCM
Codec	0.624	0.652	0.608	0.764
Collections	0.653	0.815	0.651	0.815
IO	0.757	0.806	0.759	0.814
Jsoup	0.596	0.592	0.605	0.612
JsoupParser	0.831	0.812	0.775	0.817
Mango	0.649	0.744	0.652	0.751
Ormlite-Core	0.749	0.884	0.756	0.888
Average	0.694	0.758	0.687	0.780

F-measure results: As shown in Table 6, our proposed SCM feature selection framework, combined with the MIC-based feature ranking strategy, achieves the best F-measure on five of seven projects and the best average F-measure across all projects, compared with SCM combined with the other three feature ranking strategies. More specifically, SC combined with MIC improves the average F-measure by 0.129, 0.026, and 0.140 compared to SC combined with CS, IG, and ReF, respectively.

g-mean results: As shown in Table 7, our proposed SCM feature selection framework, combined with the MIC-based feature ranking strategy, achieves the best g-mean on six of the seven projects and the best average g-mean across all projects, compared with SCM combined with the other three feature ranking strategies. More specifically, SC combined with MIC improves the average F-measure by 0.103, 0.025, and 0.120 compared to SC combined with SC, IG, and ReF, respectively.

MCC results: As shown in Table 8, our proposed SCM feature selection framework, combined with the MIC-based feature ranking strategy, achieves the best MCC on five of the seven and the best average MCC across all projects, compared with SCM combined with the other three feature ranking strategies. More specifically, SC combined with MIC improves the average F-measure by 0.161, 0.030, and 0.164 compared to SC combined with SC, IG, and ReF, respectively.

AUC results: As shown in Table 9, our proposed SCM feature selection framework, combined with the MIC-based feature ranking strategy, achieves the best AUC on six of the seven projects and the best average AUC across all projects, compared with SCM combined with the other three feature ranking strategies. More specifically, SC combined with MIC improves the average F-measure by 0.086, 0.022, and 0.093 compared to SC combined with SC, IG, and ReF, respectively.

Answer: The proposed hybrid feature selection framework combining SC-based feature clustering with MIC-based feature ranking is more effective than the ones combining SC with other three feature ranking strategies. This implies that the reserved features, based on their MIC values, are more representative for the task of predicting residence of crashing faults.

6. Conclusions

In this study, we develop a novel two-stage hybrid feature selection framework, named SCM, to select an optimal feature subset to give developers or testers recommendations about whether the residence of a crashing faults is located inside or outside the stack trace. This hybrid framework eliminates redundant and irrelevant features using spectral clustering for grouping and maximal information coefficient analysis for ranking. We have conducted a performance evaluation for our proposed SCM framework on a publicly available benchmark dataset consisting of seven Java projects by employing four performance indicators. The experimental results show that the SCM framework outperforms seven baseline methods in predicting the residence of crashing faults.

For future work, we plan to apply class imbalance handling techniques to further improve performance.

Author Contributions: Conceptualization, X.L. and M.Y.; methodology, X.L.; software, X.L.; validation, X.F. and X.L.; writing—original draft preparation, X.L.; writing—review and editing, X.L., S.S. and M.Y.; visualization, Y.G. and X.F.; supervision, M.Y. and D.Y.; funding acquisition, Y.G. and S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by State Key Laboratory of Intelligent Vehicle Safety Technology, grant number IVSTSKL-202412. This research was also funded by Scientific and Technological Research Program of Chongqing Municipal Education Commission, grant number KJQN202300547.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are available at <http://cstar.whu.edu.cn/p/crater/> (accessed on 25 January 2025).

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

Abbreviations

The following abbreviations are used in this manuscript:

ML	Machine Learning
SE	Software Engineering
SC	Spectral Clustering
MIC	Maximal Information Coefficient
MCC	Matthews Correlation Coefficient
AUC	Area Under the Curve
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
LR	Logistic Regression
RF	Random Forest
kNN	k-Nearest Neighbor
CS	chi-square
IG	Information Gain
ReF	RelieFF

References

1. Wong, W.E.; Gao, R.; Li, Y.; Abreu, R.; Wotawa, F. A survey on software fault localization. *IEEE Trans. Softw. Eng.* **2016**, *42*, 707–740. [\[CrossRef\]](#)
2. Ghafoor, M.A.; Siddiqui, J.H. Cross platform bug correlation using stack traces. In Proceedings of the 2016 International Conference on Frontiers of Information Technology (FIT), Islamabad, Pakistan, 19–21 December 2016; pp. 199–204.
3. Gu, Y.; Xuan, J.; Zhang, H.; Zhang, L.; Fan, Q.; Xie, X.; Qian, T. Does the fault reside in a stack trace? assisting crash localization by predicting crashing fault residence. *J. Syst. Softw.* **2019**, *148*, 88–104. [\[CrossRef\]](#)
4. Von Luxburg, U. A tutorial on spectral clustering. *Stat. Comput.* **2007**, *17*, 395–416. [\[CrossRef\]](#)
5. Zhang, F.; Zheng, Q.; Zou, Y.; Hassan, A.E. Cross-project defect prediction using a connectivity-based unsupervised classifier. In Proceedings of the 38th IEEE/ACM International Conference on Software Engineering (ICSE), Austin, TX, USA, 14–22 May 2016; Volume 148, pp. 309–320.
6. Reshef, D.N.; Reshef, Y.A.; Finucane, H.K.; Grossman, S.R.; McVean, G.; Turnbaugh, P.J.; Sabeti, P.C. Detecting novel associations in large data sets. *Science* **2011**, *334*, 1518–1524. [\[CrossRef\]](#) [\[PubMed\]](#)
7. Xu, Z.; Zhang, T.; Zhang, Y.; Tang, Y.; Liu, J.; Luo, X.; Cui, X. Identifying crashing fault residence based on cross project model. In Proceedings of the 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), Berlin, Germany, 28–31 October 2019; pp. 183–194.
8. Xu, Z.; Zhao, K.; Yan, M.; Yuan, P.; Xu, L.; Lei, Y.; Zhang, X. Imbalanced metric learning for crashing fault residence prediction. *J. Syst. Softw.* **2020**, *170*, 110763. [\[CrossRef\]](#)
9. Zhao, K.; Xu, Z.; Yan, M.; Zhang, T.; Yang, D.; Li, W. A comprehensive investigation of the impact of feature selection techniques on crashing fault residence prediction models. *Inf. Softw. Technol.* **2021**, *139*, 106652. [\[CrossRef\]](#)
10. Song, Q.; Jia, Z.; Shepperd, M.; Ying, S.; Liu, J. A general software defect-proneness prediction framework. *IEEE Trans. Softw. Eng.* **2010**, *37*, 356–370. [\[CrossRef\]](#)
11. Liu, S.; Chen, X.; Liu, W.; Chen, J.; Gu, Q.; Chen, D. Fecar: A feature selection framework for software defect prediction. In Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, 21–25 July 2014; pp. 426–435.

12. Rehman, U.; Mahmood, T. Identification of feature selection techniques for software defect prediction by using BCF-WASPAS methodology based on Einstein operators. *Int. J. Intell. Comput. Cybern.* **2024**, *ahead-of-print*.
13. Malhotra, R.; Chawla, S.; Sharma, A. Software defect prediction based on multi-filter wrapper feature selection and deep neural network with attention mechanism. *Neural Comput. Appl.* **2025**, 1–28. [\[CrossRef\]](#)
14. Xu, Z.; Liu, J.; Yang, Z.; An, G.; Jia, X. The impact of feature selection on defect prediction performance: An empirical comparison. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016; pp. 309–320.
15. Ghotra, B.; McIntosh, S.; Hassan, A.E. A large-scale study of the impact of feature selection techniques on defect classification models. In Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina, 20–21 May 2017; pp. 146–157.
16. Ali, A.; Gravino, C. Improving software effort estimation using bio-inspired algorithms to select relevant features: An empirical study. *Sci. Comput. Program.* **2021**, *205*, 102621. [\[CrossRef\]](#)
17. Liu, Q.; Xiao, J.; Zhu, H. Feature selection for software effort estimation with localized neighborhood mutual information. *Clust. Comput.* **2019**, *22*, 6953–6961. [\[CrossRef\]](#)
18. Goyal, S.; Bhatia, P.K. Feature selection technique for effective software effort estimation using multi-layer perceptrons. In *Proceedings of ICETIT 2019, the 1st International Conference on Emerging Trends in Information Technology, Delhi, India, June 2019*; Springer: Cham, Switzerland, 2019; pp. 183–194.
19. Hosni, M.; Idri, A.; Abran, A. On the value of filter feature selection techniques in homogeneous ensembles effort estimation. *J. Softw. Evol. Process* **2021**, *33*, e2343. [\[CrossRef\]](#)
20. Guo, J.; White, J.; Wang, G.; Li, J.; Wang, Y. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *J. Syst. Softw.* **2011**, *84*, 2208–2221. [\[CrossRef\]](#)
21. Lian, X.; Zhang, L.; Jiang, J.; Goss, W. An approach for optimized feature selection in large-scale software product lines. *J. Syst. Softw.* **2018**, *137*, 636–651. [\[CrossRef\]](#)
22. Alazba, A.; Aljamaan, H. Code smell detection using feature selection and stacking ensemble: An empirical investigation. *J. Syst. Softw.* **2021**, *138*, 106648. [\[CrossRef\]](#)
23. Jain, S.; Saha, A. Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection. *Sci. Comput. Program.* **2021**, *212*, 102713. [\[CrossRef\]](#)
24. Le, T.D.B.; Lo, D.; Li, M. Constrained feature selection for localizing faults. In Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bremen, Germany, 29 September–1 October 2015; pp. 501–505.
25. Li, A.; Lei, Y.; Mao, X. Towards more accurate fault localization: An approach based on feature selection using branching execution probability. In Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), Vienna, Austria, 1–3 August 2016; pp. 431–438.
26. Li, G.; Yu, Z.; Yang, K.; Lin, M.; Chen, C.L.P. Exploring Feature Selection With Limited Labels: A Comprehensive Survey of Semi-Supervised and Unsupervised Approaches. *IEEE Trans. Knowl. Data Eng.* **2024**, *36*, 6124–6144. [\[CrossRef\]](#)
27. Yu, Z.; Dong, Z.; Yu, C.; Yang, K.; Fan, Z.; Chen, C.P. A review on multi-view learning. *Front. Comput. Sci.* **2025**, *19*, 197334. [\[CrossRef\]](#)
28. Shivaji, S.; Whitehead, E.J.; Akella, R.; Kim, S. Reducing features to improve bug prediction. In Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE), Auckland, New Zealand, 16–20 November 2009; pp. 600–604.
29. Shivaji, S.; Whitehead, E.J.; Akella, R.; Kim, S. Reducing features to improve code change-based bug prediction. *IEEE Trans. Softw. Eng. (TSE)* **2012**, *39*, 552–569. [\[CrossRef\]](#)
30. Platt, J. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. Large Margin Classif.* **1999**, *3*, 61–74.
31. Yu, H.F.; Huang, F.L.; Lin, C.J. Dual coordinate descent methods for logistic regression and maximum entropy models. *Mach. Learn.* **2011**, *85*, 41–75.
32. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [\[CrossRef\]](#)
33. Cover, T.; Hart, P. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* **1967**, *13*, 21–27. [\[CrossRef\]](#)
34. Ghotra, B.; McIntosh, S.; Hassan, A.E. Revisiting the impact of classification techniques on the performance of defect prediction models. In Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 16–24 May 2015; Volume 1, pp. 789–800.

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