Masters in Software Engineering
November 2024

Software Fault Prediction
using Machine Learning Algorithms

Nagajyothi Devarapalli

Faculty of Engineering, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

Contact Information:
Author(s):
Nagajyothi Devarapalli
E-mail: nade22@student.bth.se

University advisor:
Dr. Ahmad Nauman Ghazi
Department of Software Engineering

Co-advisor:
Dr. Ashir Javeed
Department of Computer Science

Faculty of Engineering	Internet	:	www.bth.se
Blekinge Institute of Technology	Phone	:	+46 455 38 50 00
SE–371 79 Karlskrona, Sweden	Fax	:	+46 455 38 50 57

Abstract

Background: Software fault prediction (SFP) is a critical task in software engineering, enabling early identification of fault modules to improve software quality and reduce maintenance costs. SFP datasets, such as PROMISE, are often characterized by high-dimensional metrics and multicollinearity, posing unique challenges. This research investigates the combined effects of feature selection and parameter tuning on the performance of machine learning models for SFP.

Methods: This study evaluates the interaction between feature selection methods, including Correlation-Based Feature Selection (CFS), Recursive Feature Elimination (RFE), Mutual Information (MI), and L1 Regularization, and hyperparameter tuning techniques such as Grid Search, Randomized Search, and Genetic Algorithm. Widely used machine learning algorithms, including Random Forest, Logistic Regression, and Support Vector Machines (SVM), are employed to optimize fault prediction performance.

Results: The combined application of CFS and Genetic Algorithm yielded the highest accuracy, achieving 88.40% with Random Forest, representing an 18% improvement over baseline models without feature selection or tuning. Feature selection reduced dimensionality and identified critical attributes such as Weighted Methods per Class (wmc) and Coupling Between Objects (cbo), while iterative parameter tuning optimized model alignment to these feature sets. Notably, the proposed methods demonstrated robustness, with minimal cross-validation variability (±1.0%), and efficiency, reducing training times in univariate methods like L1 Regularization.

Conclusions: This study concludes that integrating multivariate feature selection with iterative hyperparameter tuning significantly improves the accuracy, robustness, and computational efficiency of software fault prediction models. The findings establish a framework in this research optimizing fault prediction models, such as combining CFS and Genetic Algorithms for high-stakes scenarios or Randomized Search with sparse feature selection for resource-constrained environments. These findings bridge critical gaps in SFP optimization, offering a structured approach to achieve scalable and high-performing prediction models.

Keywords: Software Fault Prediction, Machine Learning, Parameter Tuning, Feature Selection, Search Optimization

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, Dr. Ahmad Nauman Ghazi, for his continuous support, invaluable guidance, and encouragement throughout my research and thesis writing. His insightful feedback, expertise, and patience have been instrumental in shaping this work. I feel incredibly fortunate to have had the opportunity to learn from such a knowledgeable and inspiring mentor.

I also wish to extend my heartfelt gratitude to my co-advisor, Dr. Ashir Javeed, for his invaluable assistance with the machine-learning component of this thesis. His expertise, insightful suggestions, and hands-on guidance were pivotal in navigating the complexities of this research area.

I am deeply thankful to my family and friends for their unwavering support, encouragement, and understanding throughout my thesis journey. I am especially grateful to my parents, Gopala Krishna Devarapalli and Rama Devi Devarapalli, my uncle, Adi Seshu Devarapalli, and my sisters, whose constant support has been invaluable to me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

It's important to realize that mistakes in software development can happen at many different levels, from complicated design issues to syntactic errors. Software faults that are not found might have a detrimental effect on intended behavior and performance. One of the biggest obstacles in the software development process is handling these errors [70, 74, 83].

Most people agree that the software testing phase of the software development life cycle (SDLC) is the most costly [87]. Throughout the SDLC, software developers can ensure trustworthy decision-making by accurately estimating the number of software faults in advance. Because of the rework required, identifying and fixing faults following software development activities is costly and time-consuming [83]. The costs of production-ready software include the costs of reproducing, discovering, fixing, and verifying faults and their locations. Early detection of errors in the software development process, particularly during initial stages such as design and coding, significantly reduces the time and costs associated with fixing bugs at later stages or after deployment. [70, 83, 93]

Machine Learning is a technique that can predict faults for a given software, especially in the case of large and complex software, with greater accuracy. These techniques learn from past input data, leaving less room for errors and assumptions as compared to statistical methods [42, 70]. A method with three steps called Software Fault-Prone Module Classification is used for classifying modules as fault and not-fault. The first step involves collecting historical data or data sets. The second step is to train the model using the data sets, and the third step is to classify the modules. Many researchers have proposed software fault prediction models using machine learning [37].

The need for effective Software Fault Prediction (SFP) models has become more apparent with the growing complexity of software systems. Machine Learning (ML) models have emerged as powerful tools for predicting software faults by analyzing historical data to identify patterns that may indicate future faults [58]. However, ML models for SFP face significant challenges such as computational complexity, resource consumption, overfitting, and the need for model optimization [34, 58].

Feature Selection is a crucial technique that aims to enhance the performance of machine learning models. It involves selecting the most relevant features from raw data, which can reduce model complexity and improve interpretability [32]. Feature Selection plays a significant role in preventing overfitting by eliminating redundant and irrelevant features. This, in turn, can lead to models that generalize better on unseen data. However, the implementation of feature Selection in SFP models that balance computational efficiency with prediction accuracy is still an area that requires more exploration [10, 102]

Optimizing machine learning (ML) models for SFP through search algorithms provides a promising approach for significant performance improvements. Search algorithms, such as Genetic Algorithms, Grid Search, Random Search, Bayesian Optimization, and Genetic Algorithms, can help tune hyperparameters to enhance the performance of models [14, 85]. However, the systematic application of these algorithms in the context of SFP, particularly in combination with feature selection strategies, is an area that requires further research [13, 68].

=The goal of this research is to propose an approach that utilizes feature selection and search algorithms to optimize ML models for software fault prediction. The study aims to investigate the significance of feature selection and parameter tuning through search optimization algorithms in software fault prediction. We further aim to enhance both accuracy and computational efficiency, ultimately leading to the development of more reliable, scalable, and efficient software fault prediction tools that can adapt to the evolving needs of software development projects [53].

In addition to improving the predictive models in terms of accuracy and efficiency, the study collects metrics such as F1 Score, Precision, and Recall. These metrics help assess how well the models identify faults and evaluate the accuracy of their predictions. Additionally, the analysis of stratified cross-validation results will provide insights into potential overfitting issues in the models.

1.2 Problem Statement

Software fault prediction uses traditional methods that have limitations and rely on manual feature selection [17, 39, 54, 77], making it difficult to achieve precision for complex software environments. Machine learning models have advanced SFP by identifying patterns indicative of potential faults [58], but they face challenges such as computational complexity, resource demands, and susceptibility to overfitting [53,80]. There is still a gap in applying feature selection and model optimization techniques, which are essential for improving model performance and efficiency [91].

As software systems become more complex and critical to daily operations, it is essential to efficiently predict and mitigate faults to maintain system reliability and functionality. Additionally, improving the computational efficiency and generalizability of Machine Learning (ML) models for SFP can significantly reduce the time and cost associated with software development and maintenance, thereby promoting more sustainable software engineering practices.

As, ML models for SFP lack efficient integration of feature selection with model optimization techniques, resulting in limited computational efficiency and overfitting. Existing approaches either focus solely on accuracy or do not fully explore the potential of search algorithms. A systematic approach that employs feature selections as an integral part of developing resource-efficient ML models is needed. The effectiveness of search algorithms in fine-tuning models for optimal performance has not been thoroughly investigated.

While feature selection and hyperparameter tuning have been extensively studied for general prediction problems, software fault prediction presents unique challenges that warrant dedicated investigation [72,89]. Fault prediction datasets are characterized by high-dimensional metrics such as complexity (e.g., cyclomatic complexity), coupling (e.g., coupling between objects), and cohesion (e.g., lack of cohesion in methods), which often exhibit multicollinearity and redundancy [27, 69]. Moreover, these metrics have domain-specific interpretations that influence fault prediction outcomes differently compared to generic prediction tasks.

Additionally, software fault data often involves imbalanced class distributions, where faulty modules represent a minority, making the optimization of feature selection and hyperparameter tuning particularly critical [24, 69].

1.2.1 Research Gap

The use of machine learning models for SFP has significantly advanced. Many techniques have been explored to improve the model's performance, including feature selection and hyperparameter adjustment. To prevent overfitting and enhance computing efficiency, feature selection methods such as genetic algorithms (GA) and particle swarm optimization (PSO) have been employed to identify important features and reduce model complexity. Similarly, hyperparameter tuning approaches like grid search and randomized search have been used to optimize machine learning models for improved accuracy. Despite these advancements, there is still a gap in the existing literature that needs to be addressed:

Limited Combination of Techniques: Most research focuses on either hyperparameter tuning or feature selection separately. While these approaches improve model efficiency and accuracy individually, there has been little study on how to systematically combine them to maximize computing efficiency and performance [3,38,68]. The combined examination of feature selection and hyperparameter tuning could lead to significant advancements in integrated software fault prediction techniques.

Hyperparameter Tuning Efficiency: The effectiveness of hyperparameter tuning strategies has not been thoroughly explored when paired with feature selection, even though they can be computationally intensive. There is a lack of research on the benefits of combining various tuning algorithms (such as genetic algorithms) with feature selection to strike a balance between computational efficiency and prediction accuracy [59, 65, 68].

To address the existing research gaps, this thesis introduces an integrated framework that combines hyperparameter tuning (Grid Search, Randomized Search, and Genetic Algorithms) with feature selection (Recursive Feature Elimination (RFE), L1 Regularization, Mutual Information (MI)) to improve software fault prediction. The effectiveness of the framework is evaluated across various scenarios by implementing it on different machine learning models, such as Random Forest, Logistic Regression, and Support Vector Machines. Our research aims to enhance both prediction accuracy and computing efficiency by combining these strategies. Additionally, the use of 10-fold stratified cross-validation ensures the reliability and applicability of the results across the dataset.

This approach fills a significant void in software fault prediction research by rectifying the limitations of previous studies and emphasizing the benefits of integrating feature selection and hyperparameter tuning.

1.3 Aim & Objective

Aim: The primary aim of this research is to develop and validate an enhanced machine learning model for software fault prediction that leverages advanced feature selection techniques and optimization algorithms to improve accuracy, reduce computational complexity, and mitigate resource consumption.

To accomplish this aim, Developing the ML model integrates effective feature engineering and optimization strategies and evaluates it against standard benchmarks. The following objectives are formulated based on the above-stated aim.

Obj 1: To reduce computational complexity and prevent overfitting, advanced feature selection techniques, and search algorithms will be explored and implemented to reduce the dimensionality of data and increase the accuracy, and efficiency of the ML model for predicting software faults.

Obj 2: To improve software fault prediction, this research aims to develop a tailored ML model that integrates feature selections and optimization techniques, ensuring high prediction accuracy, computational efficiency, and resource utilization.

Chapter 2

Background

2.1 Software Fault Prediction

When software contains faults, it can affect its quality, maintenance cost, and reliability. This is where software fault prediction plays a crucial role (refer to Fig. 2.1) [70, 74, 83]. It involves identifying and predicting these faults during the operational phase to enhance software quality, maintainability, and reliability by identifying and resolving issues early on. Over the years, several techniques have been developed for predicting software faults, including statistical methods, machine learning algorithms, and more recently, deep learning models.

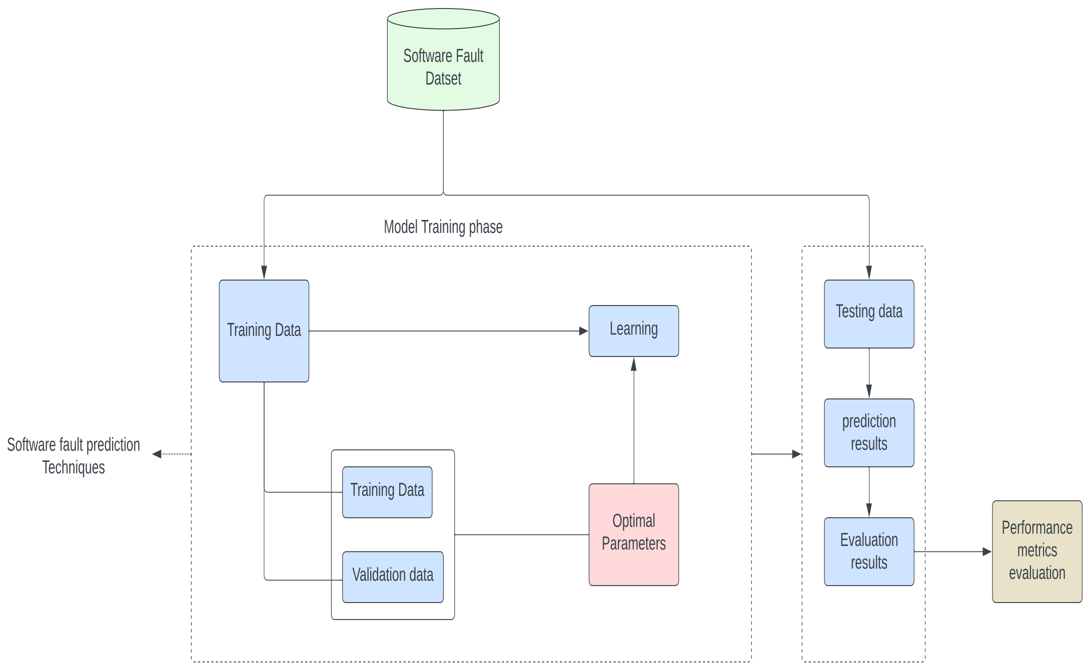


Figure 2.1: Software Fault Prediction Process

Conventional approaches to software fault prediction are time-consuming and based on manual feature engineering [17, 39, 54, 77]. They frequently have trouble handling high-dimensional, complex data. Machine learning methods including Decision Trees, Support Vector Machines, and Neural Networks have been developed to get around these constraints. These techniques look for trends in software fault data from the past and predict potential future problems. Deep learning algorithms, which can automatically extract characteristics from raw data and identify complex relationships within the data, have recently shown tremendous potential in software fault prediction.

Nevertheless, there are still difficulties to be resolved, such as the requirement for large datasets with labels, managing data that is unbalanced, and guaranteeing that the predictions have broad applicability. To solve these issues and improve the applicability and robustness of software fault prediction models, more research is required.

2.2 Feature Selection

Effective software fault prediction models require thorough feature selection. Finding the most important characteristics that affect prediction accuracy is crucial. Numerous studies have examined various approaches, including wrapper, filter, and embedded strategies, to improve software fault prediction model performance. [5, 19, 50, 73, 94]

Feature selection techniques are crucial for reducing the amount of data that needs to be processed and for obtaining the most relevant features. As a result, they help generate more accurate and meaningful predictions [15, 50, 56, 63]. By using these techniques, crucial characteristics for software fault prediction can be identified while eliminating redundant or irrelevant characteristics that could impact the final results.

There are several popular techniques for selecting important features for software fault prediction models, such as Principal Component Analysis, Recursive Feature Elimination, Information Gain, and Chi-square. These techniques help to improve the overall performance of the prediction model by highlighting the most important characteristics that differentiate malfunctioning software components from functional ones.

Feature selection techniques are crucial for creating precise and reliable software fault prediction models. By determining the most essential features, these techniques help maximize the performance of the model, decrease overfitting, increase prediction accuracy, and make the model easier to understand. Including feature selection techniques is essential for software fault prediction models to generate accurate forecasts and enhance software quality

2.2.1 Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a wrapper-style feature selection technique that generates many classification models and iteratively eliminates features that do not increase prediction accuracy to find the best feature combinations [26]. This method is paired with algorithms like Support Vector Machines, Logistic Regression, and Random Forests, which provide insights into feature importance.

The most influential features can be dynamically chosen because of RFE's iterative feature importance evaluation. Until a desired subset of features is obtained, the RFE process is training a model, prioritizing features, eliminating the least significant feature or features, and then retraining [22]. For example, in a linear model, RFE ranks features by the absolute value of the coefficients assigned to them. In tree-based models like Random Forests, feature importance scores determine the ranking [22, 105]. The algorithm proceeds as follows:

1. Begin with all features in the dataset and select an estimator model, such as Logistic Regression or Random Forest or SVM, which can provide feature importance scores or coefficients.
2. The importance of each feature is assessed. The ranking can be done using feature coefficients (in linear models) [23] or feature importance scores (in tree-based models) [105].
3. Train the model and compute the importance of each feature, either by coefficients (for linear models) or importance scores (for tree-based models).
4. Identify the least important feature(s) based on the computed rankings and remove them from the feature set.
5. Retrain the model on the reduced feature set and repeat the ranking and elimination steps until the desired number of features remains.
6. The final set of features represents those with the highest predictive power, improving model interpretability and potentially reducing overfitting.

This method is particularly advantageous for machine learning models where the importance of feature interactions can affect performance. RFE reduces the model's complexity by focusing on the most relevant features and ignoring those with little to no predictive power, thereby enhancing computational efficiency [26], 2.1.

Category	**Features**
Retained (Selected)	wmc, rfc, ce, cbo, cam, loc
Eliminated (Not Selected)	noc, mfa, moa

Table 2.1: Feature Selection Results by RFE

2.2.2 L1 Regularization

Lasso (Least Absolute Shrinkage and Selection Operator), another name for L1 regularization, is a helpful method for choosing crucial features in machine learning models [67]. By adding a penalty to the loss function dependent on the size of the coefficients, Lasso makes the model more sparse. This is especially helpful for datasets where it's crucial to choose just the most appropriate features for the interpretability and performance of the model.

In a regression model, the L1 regularization term is added to the cost function. For a linear regression model, the objective function with L1 regularization looks like this: The cost function for L1 regularization is given by [101]:

$$\text{Cost Function} = \text{Loss} + \lambda \sum_{i=1}^{n} |w_i|$$

Where:

- Loss is the standard loss function (e.g., Mean Squared Error for regression tasks).
- w_i represents the coefficients of the features.
- λ is the regularization parameter that controls the strength of the penalty (larger λ values lead to stronger regularization).

The key feature of L1 regularization is its ability to shrink some coefficients exactly to zero, thereby eliminating the corresponding features from the model 2.2.

Category	**Features**
Positively Weighted (Selected)	wmc, ce, cam, ic, cbm, amc, avg_cc
Negatively Weighted (Selected)	cbo, rfc, loc (high predictive importance despite inverse relationships)
Not Selected (Assigned Zero Weight)	ca, noc, dit, mfa, moa

Table 2.2: Feature Selection Results by L1 regularization

2.2.3 Mutual Information (MI)

Mutual Information (MI) is a measure from information theory that quantifies the amount of information obtained about one feature through another feature. It is particularly useful in feature selection as it can assess the relationship between two variables without assuming a linear relationship, making it more flexible than correlation-based measures [95].

Mutual Information measures the dependence between two random variables. For features in a dataset, Mutual Information measures how much knowing one feature reduces uncertainty about the other feature [84, 95]. It is defined as:

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

Where:

- $I(X;Y)$ is the mutual information between variables X and Y.
- $H(X)$ and $H(Y)$ are the entropies (measures of uncertainty) of X and Y, respectively.

- $H(X, Y)$ is the joint entropy of X and Y, which measures the uncertainty in both variables together.

If X and Y are independent, their mutual information is zero. A higher mutual information indicates a stronger dependency between the features.

Unlike correlation, which only captures linear relationships, Mutual Information can capture both linear and non-linear relationships between features. Mutual Information is a model-free feature selection method, meaning it does not make assumptions about the distribution of the data. It helps identify which features provide the most information about the target variable or other features, allowing you to retain the most informative ones for the model [36].The feature set by MI is shown in table 2.3

Category	**Features**
Top-Ranked (Selected)	rfc, loc, wmc, cbo, ce, lcom3, amc, dam, max_cc, cam
Low-Ranked (Excluded)	ca, dit, noc, ic

Table 2.3: Feature Selection Results by MI

2.2.4 Correlation-based Feature Selection (CFS)

Correlation-based Feature Selection (CFS) is a feature selection method that selects subsets of features based on their correlation with the target variable and with each other [33]. The core idea is to select features that are highly correlated with the target variable but have low correlations with each other (i.e., reducing redundancy).

CFS evaluates the usefulness of a feature subset by considering both the individual predictive ability of each feature (how well it correlates with the target) and the degree of redundancy among features (how correlated they are with each other) [28]. The basic idea is that good feature subsets should contain features that are highly correlated with the target variable but not too highly correlated with each other.

The evaluation criterion for a feature subset S is given by the formula:

$$\text{Merit}(S) = \frac{k_{\text{avg}}}{\sqrt{k(k-1)} + k_{\text{corr}}}$$

Where:

- k_{avg} is the average correlation of the features in S with the target variable.
- k_{corr} is the average correlation between the features in S.
- k is the number of features in the subset.

The idea is to select a subset of features where the features are highly correlated with the target variable but less correlated with each other, thus minimizing redundancy.

The features selected through CFS for software fault prediction will likely represent important software metrics that are strongly linked to fault likelihood but are less correlated with each other [18]. This results in better model performance and the identification of the most crucial software attributes for predicting faults 2.4 .

Category	**Features**
Retained (Selected)	wmc, cbo, loc, rfc, ce
Excluded (Not Selected)	noc, ca, lcom3

Table 2.4: Feature Selection Results by CFS

2.3 Machine learning

2.3.1 Random Forest

An effective method for determining whether or not a particular software module is defective is Random Forest (RF). For tasks involving binary classification, it is especially helpful [47]. Because it uses an ensemble of several decision trees, it can handle high-dimensional data and resists overfitting, which is one of its main advantages.

As shown in Fig 2.2 [30], the Random Forest algorithm builds multiple decision trees using the following steps:

- Bootstrap Sampling: Random subsets of the data are created by sampling with replacement (bootstrap). Each subset is used to train a single decision tree.
- Random Feature Selection: At each split in a tree, a random subset of features is considered, which introduces diversity among the trees.
- Model Aggregation: Once all the trees are trained, predictions are aggregated (majority voting for classification or averaging for regression).

These mechanisms help reduce overfitting and improve the model's ability to generalize to unseen data, making it highly effective for predictive tasks such as software fault prediction. By averaging predictions across multiple trees, Random Forest lowers the risk of overfitting, particularly in noisy software datasets [47]. Additionally, Random Forest has a built-in feature to estimate the importance of different metrics, which aids in identifying the most critical indicators for predicting software faults [62].

2.3.2 Logistic Regression

A statistical technique called logistic regression is used to predict binary results based on a group of independent factors. Considering a variety of software variables, logistic regression estimates the probability that a software module includes a fault in

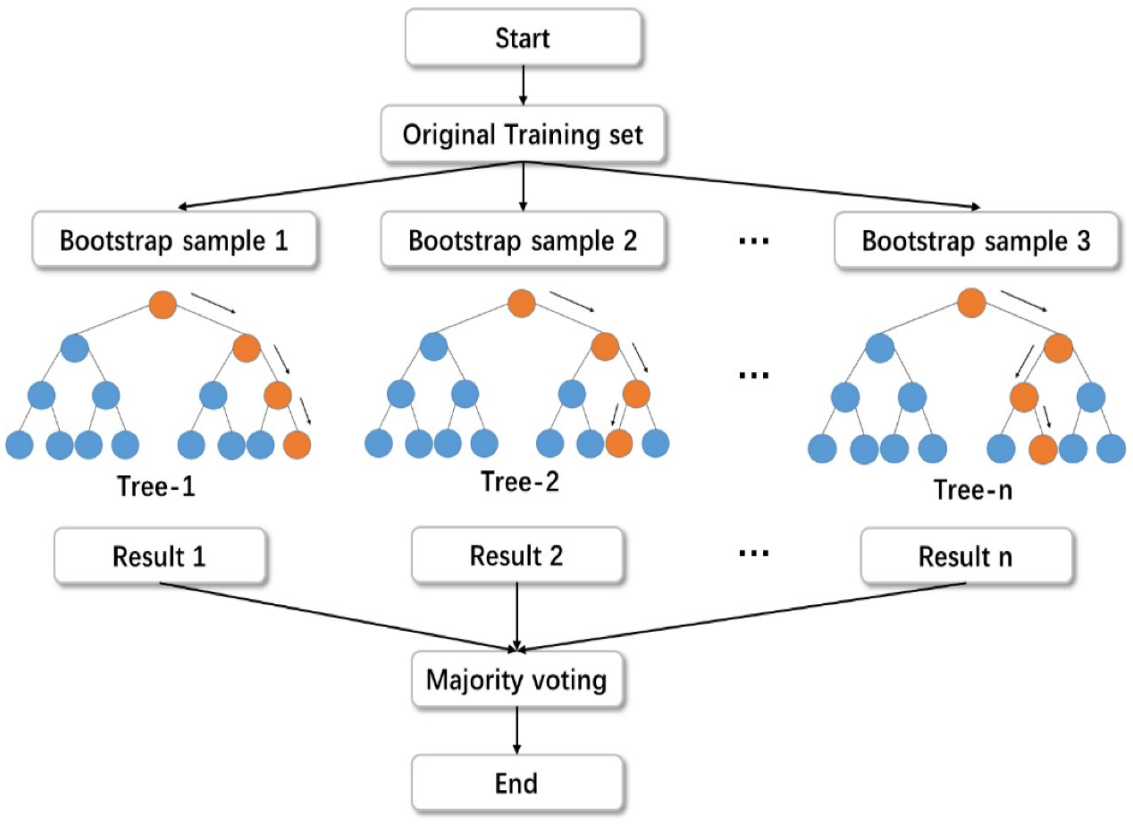


Figure 2.2: Random Forest

the context of software fault prediction [76, 81].

The logistic regression model takes a collection of predictor variables $X = (X_1, X_2, \ldots, X_p)$ and predicts the probability $P(Y = 1|X)$ that a software module Y is faulty. The model is defined as below [81]:

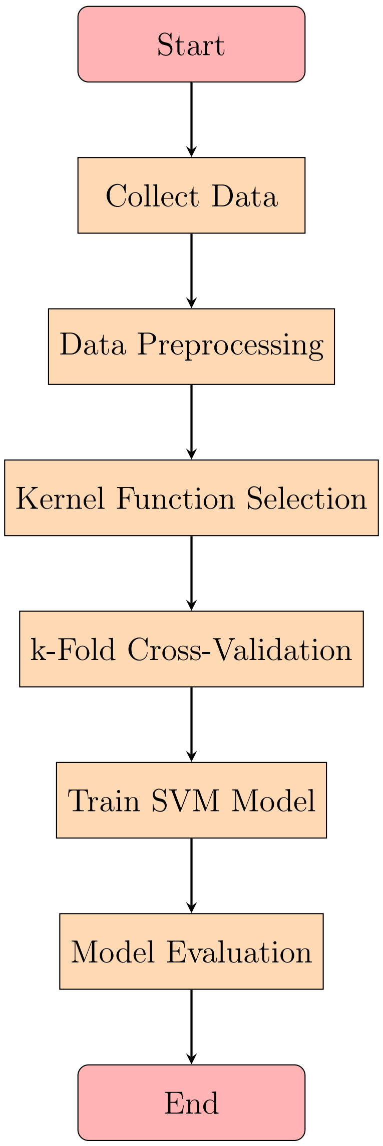
$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_p X_p)}}$$

Where:

- Y is the dependent binary variable (1 if the module is faulty, 0 otherwise).
- $X_1, X_2, \ldots, X_p$ are the independent variables (software metrics).
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ are the coefficients of the model.

The logistic function takes a linear combination of input features and transforms it into a value between 0 and 1, which can be interpreted as the probability that a component is faulty. In logistic regression, the coefficients represent both the direction and magnitude of the relationship between each feature and the likelihood of a fault occurring. For instance, a positive coefficient for a metric such as "lines of code" indicates that higher values of this metric are associated with an increased likelihood of faults.

2.3.3 Support Vector Machine(SVM)



Support Vector Machine (SVM) is a powerful machine learning algorithm commonly used for classification and regression tasks, such as software fault prediction. Support Vector Machines (SVM) aim to identify the optimal hyperplane that effectively separates data into distinct classes with the maximum margin. In software fault prediction, SVM has become popular because it can handle high-dimensional data, model non-linear relationships, and deliver accurate predictions even when data is limited. [8, 29].

For binary classification problems, the goal is to maximize the margin, i.e., the distance between the hyperplane and the nearest data points from each class, known as support vectors. In software fault prediction, the SVM algorithm classifies software modules or components as faulty or non-faulty based on various software metrics, such as complexity, code churn, and coupling [31]. Its ability to generalize well on unseen data makes it a preferred choice for building fault prediction models.

2.3.3.1 How SVM Predicts Software Faults

Hyperplane and Margin: The Support Vector Machine (SVM) algorithm aims to find a hyperplane that maximizes the margin between two classes. In a binary classification problem, the hyperplane can be defined as [75]:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input vector, and b is the bias term.

Maximizing the Margin: The margin is the distance between the hyperplane and the nearest data points from both classes, known as support vectors. The objective of SVM is to maximize this margin, which can be formulated as a convex optimization problem [75]:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i$$

where y_i is the class label of the i-th data point.

Kernel Trick: For non-linearly separable data, SVM can transform the input space into a higher-dimensional space using kernel functions. This allows the algorithm to find a linear hyperplane in this new space. Common kernel functions include [75]:

- Linear Kernel: For linearly separable data.
- Polynomial Kernel: For polynomial decision boundaries.
- Radial Basis Function (RBF) Kernel: For non-linear data.

2.4 Parameter Tuning

By modifying hyperparameters to improve performance, parameter tuning is used to refine Machine Learning models and provide accurate software fault prediction while preserving computational efficiency.

Hyperparameters are significant external settings that are vital to a machine-learning algorithm's learning process. Hyperparameters are preset before the training process starts, in contrast to model parameters, which are obtained from the training data. The learning rate, the number of epochs, the batch size, and certain settings that are pertinent to various algorithms, like the kernel type in Support Vector Machines or the depth of trees in a Random Forest, are some often encountered hyperparameters (SVM).

In a study published in 2016, [90], Tantithamthavorn et al. showed how hyperparameter adjustment significantly affects fault prediction model performance. Their results showed that researchers can get the best outcomes across a range of datasets and classifiers by methodically adjusting hyperparameters. Similarly, to achieve desired performance levels, Fu et al. (2016) [25] highlighted the significance of fine-tuning hyperparameter settings. This research highlight how important hyperparameter adjustment is to improving fault prediction models efficacy.

2.4.1 Grid Search

Grid search is a systematic approach used to fine-tune hyperparameters by exhaustively exploring a specific subset of the hyperparameter space in a learning algorithm [44].

The primary goal of grid search is to identify the optimal combination of hyperparameters that maximizes the performance of a model on a validation set. For each hyperparameter combination, the model is trained on the training data and evaluated on the validation data. This process often involves cross-validation, where the data is divided into folds to ensure reliable performance assessment [75]. The hyperparameter combination that yields the highest performance, such as accuracy, precision, recall, or F1 score, is selected as the optimal set for the model.

The key steps Grid Search involves are :

- Cross-validation: K-fold cross-validation is a commonly used technique in machine learning. It involves splitting the data into K subsets, or folds, and training the model K times. During each iteration, one fold is used as the validation set while the remaining K-1 folds are used for training.
- Performance Metrics: This process helps us evaluate the model's performance by calculating metrics such as accuracy, precision, recall, F1 score, and ROC-AUC. These metrics provide valuable insights into the model's effectiveness when different hyperparameter configurations are considered."

2.4.1.1 Parameters for baseline models

Random Forest Parameters

The parameters of the Random Forest that are tuned are shown in Table 2.5 [82].

- n_estimators: The number of trees in the forest is specified by this parameter. Increasing the number of trees usually enhances the model's performance, but it also extends the computation time. More trees can lead to improved performance but also increase training time.
- max_depth: This parameter controls the maximum depth of each tree, influencing the model's complexity. If set to None, the trees can grow until all leaves are pure or contain fewer than the specified number of samples for splitting. It governs the complexity of the trees. Shallow trees may underfit, while deep trees may overfit.
- min_samples_split: This parameter sets the minimum number of samples required to split an internal node. A higher value can help prevent the model from learning overly specific patterns, which can lead to overfitting. It prevents overfitting by requiring a minimum number of samples to split a node.
- min_samples_leaf: This parameter determines the minimum number of samples required at a leaf node. It helps in reducing overfitting by refining the

model. It ensures each leaf has enough samples, smoothing the model and reducing variance.

Random Forest Parameter Grid	**Values**
n_estimators	[100, 200, 300]
max_depth	[None, 10, 20, 30]
min_samples_split	[2, 5, 10]
min_samples_leaf	[1, 2, 4]

Table 2.5: Parameter grid for Random Forest

Logistic Regression Parameters

The parameters of the logistic regression that are tuned are shown in Table 2.6 [52].

- C: Inverse of regularization strength. Smaller values indicate stronger regularization, which helps prevent overfitting by penalizing large coefficients.
- penalty: Specifies the norm used in penalization (either l1 or l2). l1 can lead to sparse models, useful for feature selection, while l2 is a common choice for logistic regression and prevents large coefficients.
- solver: Algorithm used in the optimization problem. For small datasets, liblinear is efficient and supports both l1 and l2 penalties.

Best Parameters: 'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'

Logistic Regression Parameter Grid	**Values**
C	[0.001, 0.01, 0.1, 1, 10, 100]
penalty	['l1', 'l2']
solver	['liblinear']

Table 2.6: Parameter grid for Logistic Regression

SVM Algorithm Parameters

The parameters of the SVM that are tuned are shown in Table 2.7 [88].

- C: The regularization parameter is an important factor in determining the margin size and misclassification in the algorithm. A larger C value will result in a narrower margin hyperplane, while a smaller C value will lead to a wider margin. This parameter helps balance the trade-off between margin size and misclassification. Higher C values are suitable for accommodating more intricate boundaries.
- gamma: The gamma parameter is used in the RBF, poly, and sigmoid kernels. It can be set to either scale (default) or auto. The gamma value defines the reach of the influence of a single training example. When set to scale, the gamma value adjusts automatically based on the number of features.

- kernel: The kernel parameter specifies the type of kernel used in the algorithm. The linear, RBF (radial basis function), and poly (polynomial) kernels are commonly used choices. The kernel type determines the data transformation applied to the input data. The linear kernel is effective for capturing simple, linear relationships between the data points.

SVM Parameter Grid	**Values**
C	[0.1, 1, 10, 100]
gamma	['scale', 'auto']
kernel	['linear', 'rbf', 'poly']

Table 2.7: Parameter grid for SVM

2.4.2 Randomised Search

Randomized search is a highly effective method for tuning hyperparameters. Instead of exhaustively searching through all possible combinations, it samples a specified number of parameter settings from a given distribution. This approach is particularly useful for models with a large hyperparameter space, as it reduces the computational burden while still providing a good approximation of the optimal hyperparameters [11].

In random sampling, parameters are sampled from specified statistical distributions, such as uniform, log-uniform, or specific discrete sets. The distribution for each parameter determines how the sampling is done. The parameter p is sampled from the distribution $\mathcal{D}(p)$ [11]:

$$\text{Sample}(p) \sim \mathcal{D}(p)$$

where $\mathcal{D}(p)$ is the distribution for parameter p.

To evaluate each sampled set of parameters, k-fold cross-validation is often used. This involves splitting the data into k subsets, training the model on k-1 subsets, and evaluating its performance on the remaining subset. The cross-validation score is then calculated as the average of the scores obtained on each fold. The cross-validation (CV) score is given by [14]:

$$\text{CV Score} = \frac{1}{k} \sum_{i=1}^{k} \text{Score}(D_{\text{train}_i}, D_{\text{valid}_i})$$

where D_{train_i} and D_{valid_i} are the training and validation splits for the i-th fold.

Performance metrics, such as accuracy, precision, recall, and F1 score, are commonly used to determine the best parameter set. These metrics provide insights into the model's performance and help in selecting the most suitable hyperparameters.

Overall, randomized search, combined with random sampling, cross-validation, and performance metrics, offers a powerful approach to hyperparameter tuning. It allows for efficient exploration of the hyperparameter space and helps in finding the optimal parameter settings for machine learning models.

Random Forest Parameters

Parameters that are considered for tuning for random forest using randomized search tuning algorithm are shown in Table 2.8.

Random Forest Parameter Grid	**Values**
n_estimators	randint(100, 500)
max_depth	[None, 10, 20, 30, 40, 50]
min_samples_split	randint(2, 11)
min_samples_leaf	randint(1, 11)
bootstrap	[True, False]
criterion	[gini, entropy]

Table 2.8: Parameter grid for Random Forest for Randomised Search

Logistic Regression Parameters

Parameters that are considered for tuning for Logistic Regression using a randomised search tuning algorithm are shown in Table 2.9.

Logistic Regression Parameter Grid	**Values**
C	loguniform(1e-4, 1e+4)
penalty	[l1, l2]
solver	[liblinear]

Table 2.9: Parameter grid for Logistic Regression for Randomised Search

SVM Parameters

Parameters that are considered for tuning for SVM using randomized search tuning algorithm are shown in Table 2.10.

SVM Parameter Grid	**Values**
C	reciprocal(0.1, 20)
kernel	[rbf, linear, poly, sigmoid]
gamma	expon(scale=1.0)

Table 2.10: Parameter grid for SVM for Randomised Search

2.4.3 Genetic Algorithm

Genetic Algorithms (GAs) are optimization techniques inspired by natural selection and genetics. They are valuable for fine-tuning parameters in machine learning models, including those for software fault prediction. GAs guide a population of candidate solutions towards optimal hyperparameter settings using operations similar to biological evolution: selection, crossover, and mutation [46,83,90,92].

How Genetic Algorithms Work [51]:

- Initialization: A population of individuals (candidate solutions) is randomly generated, with each individual representing a specific combination of hyperparameters.
- Evaluation: The fitness of each individual is evaluated using a fitness function, typically based on the model's performance on a validation set.
- Selection: Individuals are selected based on their fitness scores to create a mating pool. Selection methods include tournament selection, roulette wheel selection, and rank selection.
- Crossover (Recombination): Pairs of individuals from the mating pool are combined to produce offspring. Techniques such as single-point crossover, multi-point crossover, and uniform crossover are used to exchange genetic information.
- Mutation: Random changes are introduced to the offspring's genes to maintain genetic diversity, preventing the algorithm from converging too quickly to a local optimum.
- Replacement: The new generation of individuals replaces the old population, and the process repeats until a stopping criterion is met, such as a maximum number of generations or convergence to a satisfactory fitness level.

The fitness function evaluates how well a set of hyperparameters performs. For software fault prediction, this could be accuracy, precision, recall, F1 score, or a combination of these metrics. The fitness function evaluates the performance of the hyperparameter set across k cross-validation folds [45]:

$$\text{Fitness} = \sum_{i=1}^{k} \text{Performance}(h_i)$$

where k is the number of cross-validation folds and h_i is the performance of the hyperparameter set in the i-th fold.

For example, single-point crossover involves randomly selecting a crossover point and exchanging genetic material between two parents at this point. The crossover operation combines genetic information from two parent individuals to create offspring:

$$\text{Offspring1} = \text{Parent1}[:\text{crossover\_point}] + \text{Parent2}[\text{crossover\_point}:]$$

$$\text{Offspring2} = \text{Parent2}[:\text{crossover\_point}] + \text{Parent1}[\text{crossover\_point}:]$$

In the mutation process, a small Gaussian noise is added to a gene to introduce variation.

$$\text{Gene}_{\text{new}} = \text{Gene}_{\text{old}} + \mathcal{N}(0, \sigma)$$

where $\mathcal{N}(0, \sigma)$ represents a Gaussian noise with mean 0 and standard deviation σ.

Parameters for Tuning:

Genetic Algorithm Parameters:

- Population: Number of individuals in the population. A larger population increases diversity but also computational cost.
- NGEN: Number of generations the algorithm will run. More generations allow the algorithm to explore a larger solution space.
- CXPB: Crossover probability, determining the fraction of the population that undergoes crossover.
- MUTPB: Mutation probability, determining the fraction of the population that undergoes mutation.

Chapter 3

Related Work

SFP is crucial for ensuring software quality and reducing development costs by predicting potential faults early in the software lifecycle [8,64,76]. Faults in software can occur during any phase of the software engineering process, from design and coding to integration with external environments. These faults can range from minor issues to critical system faults, potentially leading to significant project delays and cost overruns. Studies have shown that up to 50% of a project's budget can be consumed by detecting and removing software faults [40,41,48]. As a result, identifying faults at an early stage is essential to lowering software development costs.

A wide range of techniques has been employed to predict software faults, but machine learning (ML) methods have demonstrated superior accuracy compared to traditional statistical techniques for classifying software modules as faulty or non-faulty [39,53,68,70,80,83]. Research by Wang et al. [96] and R Mahajan [57] has confirmed that ML techniques offer robust solutions for predicting software faults. Machine learning can predict software faults with greater precision by learning from historical data, minimizing human judgment errors and increasing predictive accuracy [60].

In a comparative study, F. AlShaikh [9] evaluated five supervised machine learning algorithms for predicting software faults. The results indicated that both Random Forest (RF) and Artificial Neural Networks (ANN) outperformed other methods, with accuracy levels exceeding 81%. Another study by Ali, Roy, and Raj found that Naive Bayes, Support Vector Machines (SVM), and RF were among the most popular methods for predicting software faults due to their accuracy improvements [6].

While these models have shown success, their performance largely depends on two factors: the selection of relevant features and the optimization of hyperparameters. Both aspects are crucial in improving the model's accuracy and efficiency.

3.1 Feature Selection Techniques

Feature selection is critical for improving the accuracy of SFP models by reducing data dimensionality and eliminating irrelevant features. This process allows ML models to focus only on the most informative features, which enhances predictive performance and reduces computation time [5,21,46]. In a study that reviewed feature selection methods for SFP, it was found that such methods provide valuable guidance for practitioners and researchers to improve model accuracy and reduce development costs [5].

Feature selection methods are broadly categorized into four types: Filter methods assess the importance of features by assigning relevance scores, discarding features with low scores [104]. Wrapper methods involve the model itself in the feature selection process, selecting features that optimize the model's performance [12]. Embedded methods perform feature selection as part of the model's training process, achieving better computational efficiency compared to wrapper methods [61]. Hybrid methods combine the benefits of both filter and wrapper techniques, allowing more robust feature selection strategies [61, 98, 103].

Himanshu Das and Hrishikesh Kumar [46] proposed a Genetic Algorithm (GA)-based feature selection method, which identified the most useful subset of features, improving classifier accuracy and reducing the feature set size. In another study, a proposed feature selection method increased classifier accuracy from 82.44% to 83.72% [79]. Studies by Das et al. [15, 50, 56, 63] have demonstrated that feature selection not only improves model performance but also enhances the quality of data by reducing irrelevant and redundant features.

Despite the advances in feature selection, most studies have treated feature selection as an independent task, without exploring its combination with hyperparameter tuning. This is a key gap that this thesis aims to address [3, 38, 91].

3.2 Hyperparameter Tuning in Machine Learning Models

The performance of machine learning models is heavily influenced by their hyperparameters, which control how the model learns from data. Research has shown that default hyperparameter settings often lead to suboptimal results, and models with optimized hyperparameters outperform those with default settings [2, 100]. Grid search and randomized search are commonly used methods for hyperparameter tuning, but these methods can be computationally expensive, especially for large datasets and complex models [66].

Wahono [92] applied a Genetic Algorithm to optimize hyperparameters in neural networks, resulting in improved predictive performance for software fault prediction. Other studies have explored various hyperparameter tuning methods, including genetic algorithms, grid search, random search, and differential evolution, to automate the optimization process. These techniques have been shown to improve model accuracy by up to 40% [90]. Osman et al. [67] applied grid search to optimize SVM and KNN models, achieving accuracy improvements of 10% for SVM and 20% for KNN.

Despite these improvements, most hyperparameter tuning techniques are used in isolation and are not combined with feature selection, which could further enhance model performance [3, 38, 91].

3.3 Deep Learning Approaches in Software Fault Prediction

Deep learning techniques have become increasingly popular in SFP due to their ability to automatically extract features and improve predictive accuracy. Convolutional Neural Networks (CNNs) have demonstrated significant improvements in software fault identification by learning from Abstract Syntax Trees (ASTs) [54]. CNNs have also outperformed traditional machine learning models, achieving 100% accuracy in some datasets, such as the KC1 dataset [4].

Liang et al. [55] applied LSTM and word embeddings to open-source projects, improving prediction accuracy by capturing semantic information from code. Dam et al. [20] developed a tree-structured LSTM to learn from the hierarchical structure of source code, achieving high performance in fault prediction using the PROMISE and Samsung datasets.

Other deep learning approaches, such as DeepLineDP [71], focused on line-level predictions to enhance cost-effectiveness, while DeepJIT [35] extracted features from commit messages and code modifications. However, LSTM and BI-LSTM have shown limitations when applied to larger datasets with metric-based features, as noted by Batool and Khan [13], who used Radial Basis Function Networks (RBFN) to enhance fault prediction in conjunction with LSTM and BI-LSTM models.

Deep Learning (DL) usually requires considerable hyperparameter tuning and vast volumes of data to attain optimal performance, however it can be quite effective in certain cases, especially when datasets lack predetermined features or class labels. Machine Learning (ML) models are a more sensible option for SFP because datasets such as NASA and PROMISE are typically lower in size and already include organized attributes. Compared to DL models, ML models can handle smaller datasets more effectively, require less intricate tuning, and require less computing power. Furthermore, DL models frequently require a big quantity of training data in order to increase accuracy, which is not always possible in SFP applications because to the difficulty of gathering huge datasets. ML models, on the other hand, offer a more balanced answer for the particular requirements of SFP and work well with fewer datasets. They are also simpler to build.

Although feature selection and hyperparameter tuning have been individually shown to enhance machine learning models for SFP, there remains a research gap in the combined application of these techniques. Most studies have focused on optimizing either feature selection or hyperparameter tuning, but few have explored their synergistic effects on improving both model accuracy and computational efficiency.

Chapter 4

Dataset

4.1 Dataset

4.1.1 Fault Prediction Data

A crucial component of this thesis is the dataset, which provides the foundation for fault prediction. The fault prediction dataset is sourced from the Tera-PROMISE Repository [1], a publicly available repository that specializes in software engineering research datasets. This dataset has been widely adopted in the field of software fault prediction due to its comprehensive inclusion of software metrics and fault lables, making it representative of real-world software faults.

The dataset comprises seven open-source Java projects from the Tera-PROMISE Repository. Each project includes detailed information about the version numbers, the class names of each file, and, most importantly, the fault labels for each source file. To facilitate the analysis and model development, the individual datasets from the seven projects were combined into a unified dataset. This consolidation process resulted in a comprehensive dataset containing 6,981 rows, each representing a unique software module (or class) characterized by 21 distinct features. These fault labels indicate whether a particular source file contains faults, serving as the target variable for our prediction models. The dataset includes both continuous and discrete variables, many of which are skewed, as this visual analysis (fig. 5.2) makes clear.

4.1.2 Features

For each source file in these projects, 20 traditional fault prediction features are provided, covering various aspects of software structure and complexity. these features are important as they capture different dimensions of software, such as complexity, coupling, cohesion, inheritance, and size, where all of which are indicative of potential faults. These features include:

- **Lines of Code (LOC)**: Total lines of code in the file.
- **Weighted Methods per Class (WMC)**: A measure of the complexity of a class, calculated as the sum of complexities of all methods in the class.
- **Depth of Inheritance Tree (DIT)**: Indicates the inheritance levels from the object hierarchy top.

- **Number of Children (NOC)**: The number of immediate subclasses inheriting from a class.
- **McCabe Complexity Measures**:
  - **Maximum Cyclomatic Complexity (Max_CC)**: The highest cyclomatic complexity value among methods in the class.
  - **Average Cyclomatic Complexity (Avg_CC)**: The average cyclomatic complexity value of methods in the class.

These features are crucial as they capture different dimensions of the software's characteristics, such as complexity, coupling, cohesion, inheritance, and size, all of which are indicative of potential faults.

4.1.3 Data Preprocessing

The data preprocessing involved several steps to ensure the dataset was clean, balanced, and suitable for training machine learning models:

- **Deleting Duplicates**: Duplicate records were identified and removed to ensure each record was unique. Duplicate software modules can arise due to repeated code or multiple iterations of the same project version. Ensuring each module is represented only once maintains the itegrity and unbiased nature of the dataset.
- **Filling Missing Values**: Missing values were identified across several key metrics, likely due to incomplete data during software development. K-Nearest Neighbors (KNN) imputation was used to estimate and fill in missing values. This technique was chosen because it leverages the similarity between data points to provide accurate estimates, reducing the risk of skewed predictions caused by missing data.
- **Normalization**: To ensure optimal performance for machine learning algorithms, the dataset values were normalized using min-max scaling method form `sklearn. preprocessing import MinMaxScaler` module. This scaling method adjusts all features to fall within a uniform range (0 to 1), whihc prevents features with larger magnitudes from dominating th emodel training process and ensures smoother convergence for optimization algorithms.

4.1.4 Data Characteristics

The original dataset contains software modules from seven different Java projects. Each module is represented using multiple characteristics, as listed in Table 3.1, resulting in an extensive dataset for fault prediction. After combining the modules into a single file, the dataset was cleaned and balanced for faulty and non-faulty modules. The cleaning process involved removing invalid samples and ensuring that all necessary metric values were present. Additionally, the dataset was balanced using techniques like ADASYN (Adaptive Synthetic Sampling) to address the imbalance between faulty and non-faulty modules. This oversampling technique creates synthetic samples based on the existing data to balance out the class distributions.

The cleaned dataset comprises thousands of valid samples. To rigorously evaluate the performance of the model, a 10-fold stratified cross-validation approach was employed. In this method, the dataset was divided into ten equally sized subgroups (folds) while preserving the class distribution across each fold to address the inherent imbalance in the dataset. During each iteration of cross-validation, the model was trained on nine folds (90% of the data) and validated on the remaining fold (10% of the data). This process was repeated ten times, ensuring that each fold was used as the validation set exactly once. The final performance metrics were averaged across all ten iterations, providing a robust and unbiased estimate of the model's generalization capabilities.

Stratified cross-validation was chosen to ensure that each fold retained the same class distribution as the original dataset. This is particularly important for imbalanced datasets, such as those used in software fault prediction, where faulty modules often represent a minority class. Without stratification, some folds might lack representation of the minority class, leading to misleading evaluation metrics.

To obtain an accurate assessment of the model's generalization capability, the average performance metrics (e.g., accuracy, precision, and recall) across the ten folds were computed. This approach lowers the risk of overfitting and provides a more reliable evaluation of the model's output by testing it across the entire dataset.

4.1.5 Statistical Power

To ensure credible results, it is crucial to have adequate statistical power. The effect size represents the strength of the relationship between two variables. Given an effect size of $\gamma = 0.05$ and a desired statistical power of $\delta \geq 0.8$, the data groups minimally need 512 data points per group, amounting to 1024 data points for the entire experiment. The dataset exceeds this requirement with 4175 data points, ensuring adequate statistical power.

$$N = 2\left(\frac{\delta}{\gamma}\right)^2 = 2\left(\frac{0.8}{0.05}\right)^2 = 512$$

Where:

- N: Number of subjects
- δ: Statistical power
- γ: Effect size

The fault prediction dataset is robust and comprehensive, capturing various software metrics necessary for predicting faults. The preprocessing and feature selection steps ensure that the data is clean, normalized, and contains the most relevant features, making it suitable for training effective machine learning models. The statistical power of the dataset is adequate, ensuring that the results derived from the analysis will be credible and reliable.

Chapter 5
Research methodology

5.1 Research Questions

Research question 1: How do different feature selection techniques affect the performance of machine learning models in terms of accuracy and computational efficiency in software fault prediction?

Research question 2: Which search algorithm provides the most significant improvement in the accuracy of machine learning models for software fault prediction when tuning hyperparameters?

Research question 3: What is the combined effect of optimized feature selection and search algorithm-driven hyperparameter tuning on the efficiency and accuracy of ML models for software fault prediction?

5.1.1 Research Question 1

RQ1: *How do different feature selection techniques affect the performance of machine learning models in terms of accuracy and computational efficiency in software fault prediction?*

Effective feature selection is a critical process that involves selecting features from raw data. It directly influences a model's ability to learn from the data and make accurate predictions. By highlighting the most relevant information, feature selection can significantly enhance model performance.

Moreover, feature selection techniques can lower the computational effort by reducing the dimensionality of data. This leads to faster, more resource-efficient predictions on these systems.

5.1.2 Research Question 2

RQ2: *Which search algorithm provides the most significant improvement in the accuracy of machine learning models for software fault prediction when tuning hyperparameters?*

Hyperparameter tuning is a crucial aspect of the performance of machine learning models. It plays a significant role in how well these models can learn from data and predict outcomes. In the context of software fault prediction, finding the optimal set of hyperparameters is vital, given the complexity and variability of software fault data. This process can significantly enhance model accuracy, making predictions more reliable and actionable for software engineering tasks. Therefore, investigating the challenge of model optimization is crucial for improving the accuracy of software fault prediction models.

5.1.3 Research Question 3

RQ3: *What is the combined effect of optimized feature engineering and search algorithm-driven hyperparameter tuning on the efficiency and accuracy of ML models for software fault prediction?*

Combining feature selection and hyperparameter optimization strategies can uncover synergies that lead to more accurate and efficient machine learning models. This is especially important in real-world applications where resources are often limited, and software fault can have high stakes. By effectively combining these strategies, more reliable, scalable, and resource-efficient ML models for predicting software faults can be developed, contributing significantly to the advancement of software quality assurance methodologies.

5.2 Variables

Independent Variables:

- **Feature Selection Techniques:** Feature selection is a process that involves the use of Recursive Feature Elimination (RFE), L1 regularization, and Mutual Information(MI). These techniques are highly effective in enhancing model performance by getting rid of irrelevant features.
- **Machine Learning Models:** For this study, I have chosen three baseline machine learning models that are suitable for binary classification tasks commonly found in software fault prediction. These models are Support Vector Machine (SVM), Random Forest, and Logistic Regression. These models are selected due to their diverse mechanisms.
- **Search Algorithms for Hyperparameter Tuning:** To optimize the performance of the models, various search algorithms for hyperparameter tuning were utilized. The search algorithms used include Grid Search, Randomized Search, and Genetic Algorithm. Each algorithm represents a different approach to navigating the hyperparameter space, which is crucial for optimizing model performance.

Dependent Variables:

- The study focuses on two primary dependent variables - Accuracy and Computational Time, which give a direct assessment of the effectiveness and efficiency of the model. Alongside these, secondary metrics such as Precision, Recall, F1 Score, and Support are also monitored to evaluate the detailed performance characteristics of each model.

- Additional measures include the number of Correctly Classified Instances and Incorrectly Classified Instances, which provide insights into the models' prediction capabilities and errors.

Control Variables:

- In order to guarantee consistent and reliable outcomes in the study, several factors have been taken into consideration. These include the size and characteristics of the dataset used, the uniformity of the feature selection process, and the implementation of hyperparameter tuning techniques. This control is crucial to isolate the independent variables' effects and ensure that any observed results are solely due to the variations in machine learning techniques and optimization algorithms applied.

5.2.1 Metrics

It is not enough to rely solely on accuracy (eq. 5.1) when evaluating a software fault prediction model, particularly when dealing with imbalanced datasets where faults are a minority [7, 78].

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

For example, a model may achieve high accuracy by simply predicting "no-fault" for all modules. To gain a more nuanced understanding of the trade-off between different aspects, it is essential to consider other metrics such as precision (eq. 5.2), recall (eq. 5.3), f1 score (eq. 5.4), and support, which include correctly and incorrectly classified instances. By doing so, the model can be optimized according to the specific needs of software fault prediction [43, 86].

Precision indicates the model's ability to avoid false positives, meaning how many predicted faults were actually faults.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

Identifying actual faults in code using **recall** helps determine how good the model is at finding them. It can be used to find out how many faults our model identified in the code.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.3}$$

By combining precision and recall, **F1 Score** gives a more comprehensive picture of the model's performance.

$$F1 \text{ Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

Understanding the distribution of data and interpreting the other metrics in context becomes easier with the help of support. Correctly Classified Instances and Incorrectly Classified Instances provide a more intuitive understanding of how accurate the predictions of the model are.

High precision means that the model makes fewer false positive predictions, which is crucial to avoid unnecessary debugging efforts. High recall, on the other hand, means that the model identifies a good portion of the actual faults present, reducing the number of missed bugs. F1 Score balances precision and recall, providing a single score for overall model performance. Support indicates the number of data points in each class, whether they are faulty or non-faulty, and helps understand if the metrics are biased due to imbalanced datasets.

5.3 Experimental instrumentation

The goal of this research is to develop a new software fault prediction model using machine learning. The model will use feature selection methods to prevent overfitting and search optimization algorithms to reduce computational complexity. The PROMISE dataset will be used in the study, and the model will be written in Python programming language.

To extract features from the dataset, Logistic Regression with L1 Regularisation will be used in conjunction with the sci-kit-learn library. The baseline model will consist of three algorithms: RF, Logistic Regression, and SVM, which have shown promising results in previous research. Parameter tuning will be performed using Grid Search, Randomised Search, and Genetic Algorithm.

5.4 Hypotheses

5.4.1 Hypothesis 1

RQ1 can be answered with this hypothesis: Does feature selection impact the performance of machine learning models for software fault prediction?

To analyze the relationship between feature selection techniques and software fault prediction, three feature selection techniques are used for feature selection. As the dataset is not normally distributed, non-parametric one-way ANOVA Test is used to compare the performance of the models with feature selection methods and without feature selection.

The hypothesis 1 involves multiple independent groups (RFE, L1 Regularization, MI, CFS, and No Feature Selection), where the objective is to compare their mean accuracies. One-way ANOVA is specifically designed for comparing the means of multiple groups. The one-way ANOVA is well-suited for comparing the means of three or more independent groups to see if there is a statistically significant difference between them.

H_0: There is no significant difference in model accuracy across feature selection methods.

H_1: There is a significant difference in model accuracy across feature selection methods.

5.4.2 Hypothesis 2

Following are the hypotheses to know whether parameter tuning using search optimization leads to any improvement or not. Kruskal-Wallis Test, a non-parametric version of ANOVA is used to determine if there's a significant difference between them. ANOVA assumes that the data within each group follows a normal distribution; however, the data chosen for this research does not meet this criterion 5.2. In contrast, the Kruskal-Wallis Test is a non-parametric test that does not assume a specific distribution of the data. Instead, it ranks the data and evaluates whether the groups originate from the same distribution, making it more robust against non-normality.

Additionally, ANOVA requires homogeneity of variances, meaning the variances within each group should be approximately equal. The Kruskal-Wallis Test, on the other hand, can accommodate scenarios where the variance in model performance (accuracy) differs across various search algorithms, providing greater flexibility when variances are unequal. This makes Kruskal-Wallis Test more suitable for this hypothesis as it does not assume normality or equal variances.

H_0 : There is no significant improvement among different search algorithms (Grid Search, Randomised Search, Genetic Algorithm) used for hyperparameter tuning in machine learning for software fault prediction

H_1 : At least one search algorithm leads to a statistically significant improvement in the accuracy of machine learning models for software fault prediction compared to others.

5.4.3 Hypothesis 3

H_0: The combination of the Feature selection method and Parameter tuning method for software fault prediction reduces overfitting and computational efficiency

H_1: There is no improvement in computational efficiency and overfitting in the software fault prediction model after integrating the machine learning model with feature selection and parameter tuning

5.5 Experiment design

The objective here is to optimize the machine learning model's performance by selecting the best features (through feature selection) and then tuning the model's hyperparameters (via search optimization techniques) to achieve high accuracy, computational efficiency, and effective resource utilization.

The experiment is conducted as follows (fig. 5.1):

Data pre-processing

- As the dataset contains open-source Java systems of 2 consecutive releases in separate files, all the versions of different Java systems are merged for training and testing purposes.
- Data is being pre-processed, which includes cleaning, handling missing values, and eliminating duplicates and columns that are not necessary such as date.

Standardizing the data

- The data is scaled using the scikit-learn MinMaxScaler. This procedure scales the data to a certain range (usually [0, 1]) and calculates each feature's minimum and maximum values in the training folds.
- To maintain consistency, the validation fold also receives the same scaling. The test and validation sets are modified using the same scaling factors because the scaler is fitted to the training data. This reflects how the model would behave on new, untested data.

Splitting Data and K-Fold Stratified Cross-Validation

- Rather than using the standard train-test split to evaluate the model, K-fold stratified cross-validation is employed. The data is divided into ten subsets (folds) using this method. Nine folds are utilized for training and the last fold is used for validation in each iteration.
- To make sure that every data subset is utilized as a validation set exactly once, this process is performed ten times. By using the complete dataset for training and validation, this approach guarantees a thorough assessment of the model's performance at various folds.

Feature Selection Technique

- This research focuses on four feature selection methods: Recursive Feature Elimination, L1 regularization, Mutual Information, and Correlation-based Feature Selection(CFS). Each of these techniques is applied separately to baseline models, allowing for an evaluation of their performance in feature selection. The goal is to explore the importance of each feature selection method.
- The performance of feature selection models is evaluated using evaluation metrics, including computational complexity, and will proceed further for parameter tuning.

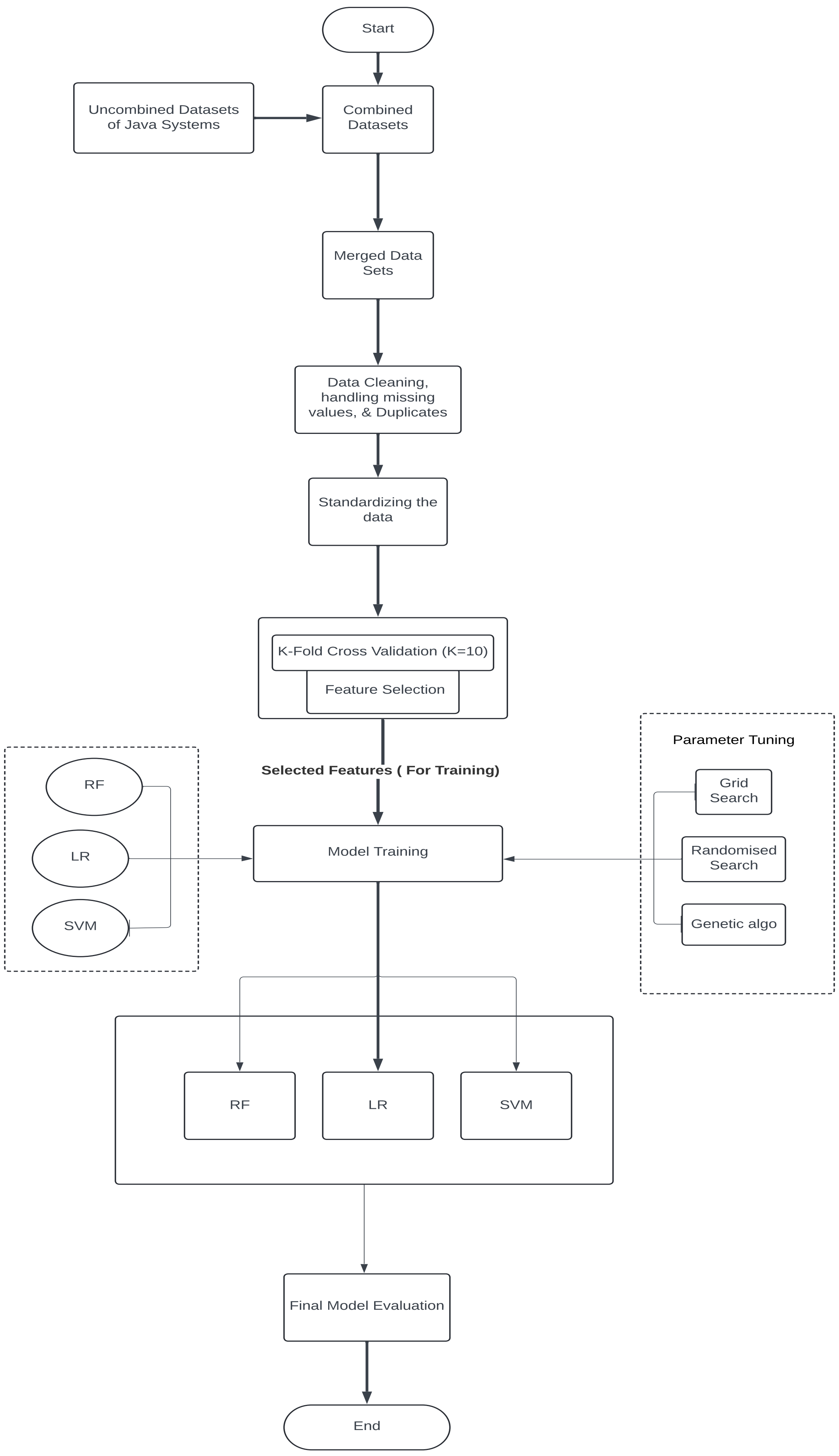


Figure 5.1: Flowchart

Baseline Model Training (With and Without Feature Selection)

- Random Forest, Logistic Regression, and Support Vector Machine (SVM) are used to train three different baseline models. Hyperparameter tuning is not done at this stage of training.

- Ten-fold stratified cross-validation is used to train and assess models. Each model is evaluated both with and without the feature selection procedure, and performance metrics like accuracy, precision, recall, F1 score, and confusion matrix are computed.

- This provides a baseline for comparison when more advanced techniques like hyperparameter optimization are applied.

Hyperparameter Optimization (Search Optimization Techniques)

- The hyperparameters of each model are adjusted using search optimization techniques once the baseline models have been assessed.

- Grid search, Randomized search, and Genetic Algorithm are the methods employed. To identify the ideal set of hyperparameters, these techniques methodically search the parameter space.

- This procedure includes K-fold cross-validation, which guarantees that each hyperparameter tuning's performance is assessed across several data subsets, preventing overfitting and guaranteeing that the model generalizes well to new data.

- In the background section of the parameter tuning models, tables are utilized to display the parameters taken into consideration for tuning each model and each optimization technique 2.5.

Final Model Training and Evaluation

- Upon feature selection and hyperparameter tuning, K-fold stratified cross-validation is used to identify the best features and hyperparameters for training the final models. Every cross-validation iteration after the models are fully trained assesses them on the validation fold.

- For every model, metrics for performance are collected, including Accuracy, F1 score, Precision, Recall, and values from the confusion matrix. To evaluate the overall performance of the model, these metrics are averaged over all the folds.

- Based on these metrics, the final comparison of the models (Random Forest, Logistic Regression, SVM) is performed with and without feature selection, using optimal hyperparameters.

5.6 Validity and reliability of the approach

The main objective was to reduce redundancy and increase computer productivity. This approach was chosen in addressing the challenges of open source Java process of different versions datasets, where high dimensionality and multicollinearity can significantly hamper model performance.

The merging of datasets process required aligning the values to the same range across all data types using standardized procedures, ensuring that all data adhered to identical format specifications and this procedure was necessary to maintain consistency and reliability in the next data processing step.

Ten-fold stratified cross-validation was used to analyze the full dataset; training and testing sets were not separated beforehand. By putting the model through numerous rounds of training and validation, 10-fold stratified cross-validation offers a more thorough and dependable assessment of its performance. By using this method, the risk of overfitting is reduced and the model is thoroughly adjusted and assessed on all data subsets.

Every subset of the data is used for both training and validation via stratified cross-validation, guaranteeing that the model is assessed on unobserved data for every fold. The stratified cross-validation performance indicators yield a reliable approximation of the model's generalization capacity. Because stratified cross-validation assesses the model on several subsets of the data by nature, this approach also does away with the requirement for a separate test set during training.

Based on initial testing and validation, the model's features and parameters were selected to ensure that they maintained enough data for an objective assessment of performance and offered enough information for reliable training. By lowering the possibility of overfitting and enhancing the results' external validity, this classification procedure improves the model's dependability.

Parameter fine-tuning involved a combination of Grid Search, Randomized Search, and Genetic Algorithms to explore the parameter space effectively. These procedures are applied to three fundamental models: Random Forest, SVM, and Logistic Regression. These were chosen due to their adaptability and strength in managing both linear and non-linear data structures. The objective of using various algorithms is to enhance the trustworthiness of identifying the best configurations. The process of cross-validation within each method offered multiple evaluations of model performance, thereby further substantiating the sturdiness of the fine-tuned models.

A significant advantage of this methodology lies in its organized and methodical strategy for reducing overfitting by meticulously selecting features and optimizing parameters. Nonetheless, it could be constrained by the presumption that the chosen features will exhibit consistent performance in different Java systems. This assumption might not hold true if these systems differ in their foundational structures or the distribution of their data.

5.7 Analysis procedure

To determine if the dataset is normally distributed or not, the Shapiro-Wilk Test is taken for the normality check, along with histograms showing the distribution of data of each metric is also plotted to check visually(fig. 5.2). It is evident from the graph that variables are not symmetrically distributed around central values, distributions that are skewed, indicating a concentration of data at one end of the scale, or distributions that are multimodal, characterized by several peaks. These features imply that the data does not follow a normal distribution.

Each Research question has a primary hypothesis. The hypothesis of RQ1 is tested using the one-way ANOVA to compare the performance of the models with feature selection methods and without feature selection to observe the impact of feature selection on the machine learning model for software fault prediction.

Kruskal-Wallis Test, a non-parametric version of ANOVA is used for RQ2 hypothesis testing, to determine the difference between parametric tuning using search optimization algorithms impact.

In the same way, the hypothesis of RQ3 is also tested. All the significance level of the hypothesis are set to 0.05 ($\alpha = 0.05$)

5.8 Threats to Validity

5.8.1 Conclusion validity

The concept of conclusion validity pertains to the relationship between the various machine learning methods employed (treatment) and their effectiveness in predicting software faults (outcome).

The dataset used contained a significant class imbalance between faulty and non-faulty modules. This imbalance can cause the model to be biased towards the majority class, leading to poor predictive performance on the minority class fig. 5.2. To address this issue, the ADASYN (Adaptive Synthetic Sampling) algorithm was employed. It generated synthetic examples of the minority class, which in this case are faulty modules.

Statistical tests were carried out to determine if there was a significant relationship between the machine-learning treatments and the prediction outcomes. To test the null hypothesis that changes in the predictive accuracy of the models are due to random chance rather than the interventions, p-values were calculated. Different metrics such as accuracy, precision, recall, and the F1-score were employed to evaluate the effectiveness of the machine learning models. The use of these metrics ensures a comprehensive evaluation of model performance, taking into account both the prevalence of faults and the importance of accurately predicting them.

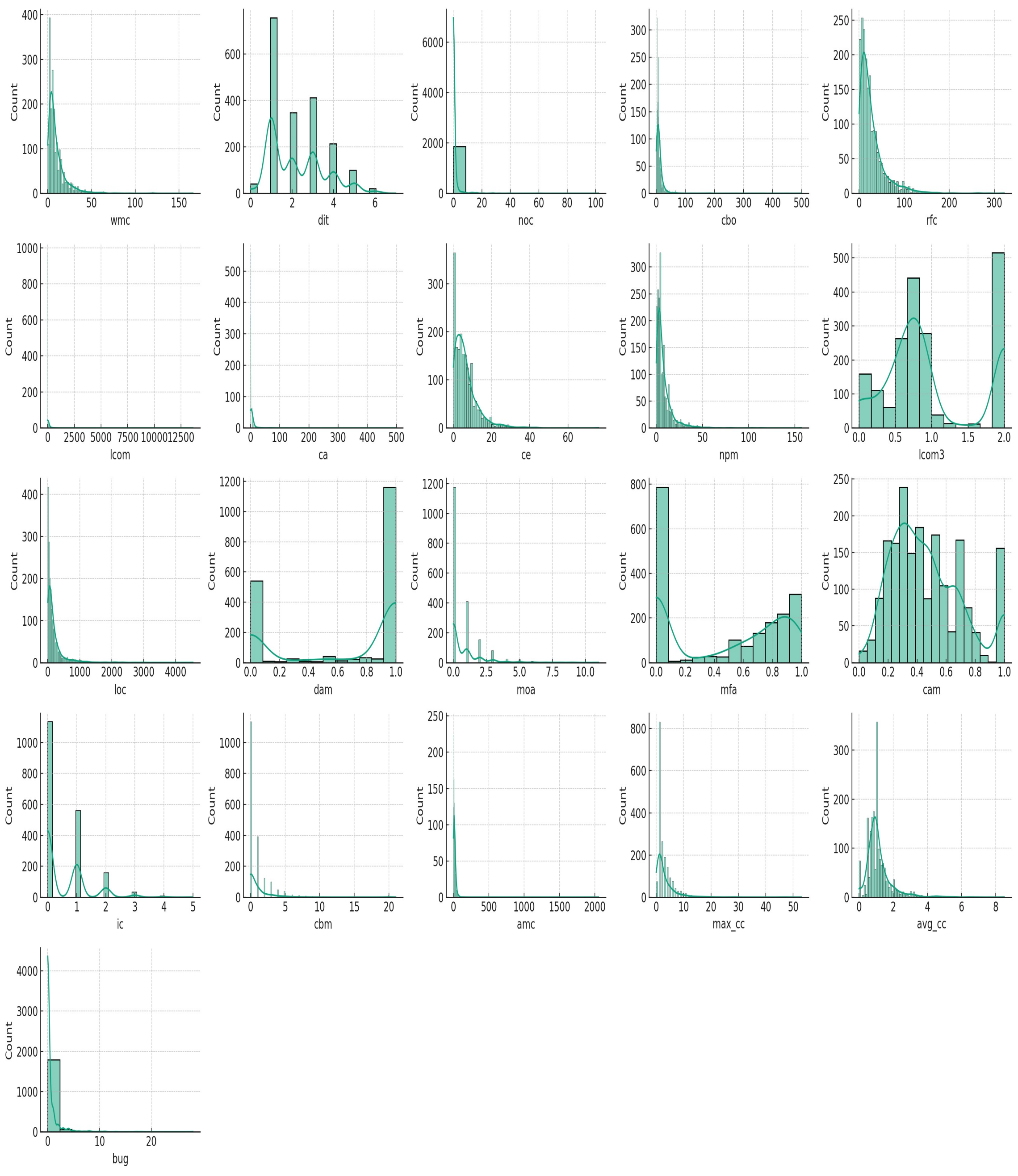


Figure 5.2: Data distribution

5.8.2 Internal validity

Internal validity refers to the degree of confidence that can be added to the relationship between the features, hyperparameter tuning, and the achieved model performance, including accuracy, efficiency, and resource utilization. It ensures that any observed improvements in the model's performance are solely due to the chosen methods, such as L1 regularization and search optimization techniques, and are not influenced by any external factors.

The process of selecting features used L1 regularization to promote model interpretability and prevent overfitting. This regularization shrinks less important feature coefficients to zero, leaving only the most significant predictors in the final model. The regularization parameter was chosen through a systematic evaluation of model performance on a dedicated validation dataset.

To optimize the model's hyperparameters, grid search, randomized search, and genetic algorithm methods are compared. Grid search exhaustively worked through predefined hyperparameters, while randomized search was more computationally efficient, albeit probabilistic. Inspired by natural selection principles, the genetic algorithm introduced innovative parameter combinations that could potentially outperform grid-based searches. Each method was optimized for computational efficiency and the final model's accuracy, confirmed through extensive performance evaluations.

5.8.3 Construct validity

The term "construct validity" refers to ensuring that the methods (Feature selection and Search Optimization Algorithms) used in a thesis accurately represent the principles of selecting features, model tuning, and performance optimization.

To evaluate the effectiveness of L1 regularization as a feature selection technique, the impact of selected features on model performance is analyzed across multiple datasets. The fact that L1 regularization successfully isolates the most important features and reduces model complexity without affecting accuracy confirms its construct validity.

The integration of hyperparameter optimization and feature selection should theoretically improve both the computational efficiency and accuracy of the model. The thesis provides evidence for the construct validity of this integrated method through experiments that demonstrate the superior performance of customized models with specific features over those without.

The practical applications and outcomes of the thesis provide real-world evidence of the effectiveness of these structures. Models incorporating these techniques can both improve processing efficiency and achieve high accuracy.

5.8.4 External validity

The term external validity refers to the extent to which the results obtained from machine learning models' optimization, through feature selection and hyperparameter tuning, can be applied to different populations, scenarios, or periods.

The key question here is whether machine learning models created for one type of software system can be used for another. Even if this study includes a variety of software systems with varying scales and complexities, the specific features of the systems being examined can still affect the study's external validity. In the future, researchers should aim to apply established models to a broader range of software systems, including those developed in different languages or architectures, to assess the stability and generalizability of model optimizations.

The feature selection and hyperparameter tuning strategies (L1 regularization, Grid Search, Randomized Search, and Genetic Algorithm) used in this thesis may not produce the same outcomes when applied to alternative datasets or models. The model complexity and underlying data distribution may impact how effectively these strategies perform. To validate the robustness and applicability of the results, a comparison study with alternative approaches should be included.

Chapter 6

Results and Analysis

The main objective of this research was to improve model performance by selecting features using feature selection methods and then optimizing model parameters using methods including Grid Search, Randomized Search, and Genetic Algorithms. These techniques were chosen to obtain high accuracy, computational efficiency, and efficient resource utilization.

The model performance metrics were obtained using 10-fold stratified cross-validation, where each metric accuracy, precision, recall, and F1 score was calculated for each fold. The table reports the average values of these metrics across the 10 folds to provide a robust measure of model performance.

From table 6.1, it is evident that Recursive Feature Elimination (RFE) has consistently improved model accuracy, particularly for Random Forest, where feature interactions play a significant role. Accuracy increased from 84.12% with Grid Search to 87.41% with the Genetic Algorithm, highlighting the effectiveness of the Genetic Algorithm in enhancing the reduced feature set selected by RFE. SVM has also benefited from RFE's straightforward approach to feature reduction, especially when combined with tuning methods. However, Logistic Regression showed only moderate gains due to its reliance on the quality of features rather than the quantity. Although RFE helped in removing irrelevant features and enhances model interpretability, the improvement in accuracy for Logistic Regression is modest compared to Random Forest.

For Random Forest, which leverages ensemble learning, RFE effectively reduced noise, allowing the model to focus on feature interactions. This accounts for the high F1 score and low cross-validation variability ($\pm 1.2\%$) shown in table 6.5. For SVM, RFE improves the formation of decision boundaries by eliminating redundant features, which enhanced both accuracy and testing time. In contrast, Logistic Regression showed minimal gains, as it fundamentally relies on linear relationships, and RFE's iterative feature elimination may not fully take advantage of multicollinear data.

L1 Regularization is effective in sparse models such as Logistic Regression and Support Vector Machines (SVM) but proved less effective for Random Forest, where feature interactions are crucial. As shown in Table 6.2, while L1 Regularization successfully pruned features, its performance with Random Forest is inconsistent. This

Table 6.1: Performance metrics of machine learning models with Feature selection using RFE and search optimizations on balanced data

Recursive Feature Elimination						
Optimization	**F1 Score**	**Precision**	**Recall**	**Accuracy (%)**	**Training Time(s)**	**Testing Time(s)**
Random Forest						
Grid Search	0.85	0.86	0.84	84.12	10.3	15.8
Randomised Search	0.86	0.87	0.85	85.25	8.7	12.4
Genetic Algorithm	0.87	0.88	0.86	87.41	9.4	14.7
Logistic Regression						
Grid Search	0.82	0.84	0.81	81.74	4.2	6.8
Randomised Search	0. 83	0.85	0.82	83.21	3.9	6.1
Genetic Algorithm	0.84	0.85	0.83	83.92	5.0	7.4
SVM						
Grid Search	0.82	0.84	0.81	82.47	6.2	9.1
Randomised Search	0.84	0.85	0.83	83.85	5.8	8.3
Genetic Algorithm	0.85	0.86	0.85	84.62	6.7	9.9

inconsistency likely arises from the interactions between features that L1 Regularization may overlook. The accuracy is highest with Randomized Search, achieving 86.32% with Random Forest as the baseline model. Meanwhile, SVM exhibits similar accuracies across various tuning methods, peaking at 83.92% when optimized using the Genetic Algorithm. This improvement indicates that L1 Regularization enhances SVM's capability to manage irrelevant features, leading to simpler decision boundaries.

Examining the results when features are selected through L1 Regularization, as seen in Tables 6.2 and 6.5, we note that L1 Regularization struggled to leverage feature interactions within Random Forest, causing inconsistent performance. Its univariate approach was less compatible with the ensembled nature of Random Forest. Additionally, testing times were lower for SVM and Logistic Regression, reflecting their effective handling of sparse datasets.

Table 6.2: Performance metrics of machine learning models with Feature selection using L1 Regularization and search optimizations on balanced data

L1 Regularization						
Optimization	**F1 Score**	**Precision**	**Recall**	**Accuracy (%)**	**Training Time(s)**	**Testing Time(s)**
Random Forest						
Grid Search	0.75	0.71	0.82	80.16	9.4	14.7
Randomised Search	0.75	0.69	0.82	86.32	8.6	12.2
Genetic Algorithm	0.74	0.68	0.82	82.06	12.0	18.0
Logistic Regression						
Grid Search	0.75	0.84	0.70	81.26	3.8	6.4
Randomised Search	0.73	0.71	0.82	80.71	3.6	5.9
Genetic Algorithm	0.75	0.71	0.83	80.73	4.5	7.0
SVM						
Grid Search	0.74	0.68	0.82	80.20	5.9	8.5
Randomised Search	0.74	0.68	0.83	80.86	5.6	7.9
Genetic Algorithm	0.75	0.68	0.82	83.92	6.5	9.4

From table 6.3, MI improved accuracy, with Genetic Algorithm (87.00%) outperforming Grid Search (84.32%). MI's univariate approach has benefited from the flexibility of the Genetic Algorithm in exploring non-linear relationships. MI's limitation in capturing multivariate dependencies has resulted in lower accuracy gains compared to CFS or RFE (from table 6.4 and 6.1). MI enhanced SVM by focusing on high-value features, though its univariate nature might overlook feature interactions. MI has performed well in models like Random Forest, where nonlinear dependencies dominate, but its univariate nature limits its effectiveness in Logistic Regression and SVM.

CFS is the most effective feature selection method, consistently improving accuracy across all models and optimization algorithms, as shown in Table 6.4. When

Table 6.3: Performance metrics of machine learning models with Feature selection using MI and search optimizations on balanced data

Mutual Information (MI)						
Optimization	**F1 Score**	**Precision**	**Recall**	**Accuracy (%)**	**Training Time(s)**	**Testing Time(s)**
Random Forest						
Grid Search	0.84	0.85	0.83	84.3	10.1	14.9
Randomised Search	0.85	0.86	0.84	83.54	8.5	12.7
Genetic Algorithm	0.0.87	0.86	0.88	87.00	11.7	17.8
Logistic Regression						
Grid Search	0.81	0.82	0.80	80.98	4.3	6.9
Randomised Search	0.82	0.83	0.81	81.67	4.0	6.2
Genetic Algorithm	0.83	0.84	0.82	83.04	4.9	7.3
SVM						
Grid Search	0.81	0.82	0.80	81.56	6.3	9.3
Randomised Search	0.82	0.83	0.81	82.34	5.9	8.7
Genetic Algorithm	0.83	0.84	0.82	83.21	6.8	9.6

applied to Random Forest, it achieved the highest accuracy of 88.40% when tuned with a Genetic Algorithm. Its multivariate nature ensures that feature interactions are preserved. Even Logistic Regression saw modest accuracy improvements, peaking at 84.15% with CFS, indicating that performance is more sensitive to feature quality due to its linear nature.

Cross-validation results from Table 6.5 confirm the robustness of CFS and the Genetic Algorithm, showcasing the lowest standard deviation across folds. The evaluation of CFS based on feature correlation with the target and redundancy has allowed it to retain only the most informative features, resulting in the highest cross-validation accuracy (± 1.0% standard deviation) and minimal overfitting. For SVM, CFS effectively reduced redundant features, optimizing decision boundary formation and decreasing training and testing time.

Table 6.4: Performance metrics of machine learning models with Feature selection using CFS and search optimizations on balanced data

Correlation-Based Feature Selection (CFS)						
Optimization	**F1 Score**	**Precision**	**Recall**	**Accuracy (%)**	**Training Time(s)**	**Testing Time(s)**
Random Forest						
Grid Search	0.85	0.86	0.85	85.76	9.5	13.8
Randomised Search	0.87	0.88	0.86	86.90	8.4	11.9
Genetic Algorithm	0.88	0.89	0.87	88.40	12.5	17.6
Logistic Regression						
Grid Search	0.82	0.83	0.82	82.12	3.9	6.3
Randomised Search	0.83	0.84	0.83	83.04	3.7	5.8
Genetic Algorithm	0.84	0.85	0.84	84.15	4.7	6.9
SVM						
Grid Search	0.83	0.84	0.82	82.88	6.0	8.8
Randomised Search	0.84	0.85	0.83	83.76	5.7	8.2
Genetic Algorithm	0.85	0.86	0.85	85.12	6.9	9.8

The combination of Random Forest, CFS, and Genetic Algorithm (Table 6.4) achieved the highest accuracy of 88.40%, demonstrating its strong capability to detect faults more reliably. Both Logistic Regression and SVM also showed significant improvements with CFS, highlighting the importance of multivariate feature selection in enhancing fault prediction tasks. The highest F1 score of 0.88 was observed for Random Forest with CFS and Genetic Algorithm, reflecting its exceptional balance between precision and recall. This indicates the model's effectiveness in minimizing false positives and false negatives, which is critical for accurate fault prediction.

While the training time was highest for the Genetic Algorithm due to its exhaustive search process, the resulting gains in accuracy and generalization justify the computational cost. Additionally, testing times remained practical across all models, ensuring applicability for real-time fault prediction scenarios. Stratified cross-validation

Table 6.5: Validation results for accuracy from cross-validation, computed for Random Forest using feature selection method, and tuning algorithms. The values represent the mean accuracy and standard deviation across 10 validation folds.

Random Forest			
Feature Selection	**Optimization**	**Mean Accuracy**	**Std. Dev.**
RFE	Grid Search	84.12	±1.5%
	Randomised Search	85.25	±1.3%
	Genetic Algorithm	87.41	±1.2%
L1 Regularization	Grid Search	80.16	±1.6%
	Randomised Search	86.32	±1.4%
	Genetic Algorithm	82.06	±1.1%
MI	Grid Search	84.32	±1.7%
	Randomised Search	83.54	±1.5%
	Genetic Algorithm	87.00	±1.2%
CFS	Grid Search	85.76	±1.4%
	Randomised Search	86.90	±1.2%
	Genetic Algorithm	88.40	±1.0%

results further validated the robustness of Random Forest with CFS and Genetic Algorithm (Table 6.5), showing the lowest standard deviation (±1.0%) across folds. In contrast, higher variability in cross-validation metrics for Logistic Regression (Table 6.6) and SVM (Table 6.7) underscores their sensitivity to feature selection methods, reaffirming the importance of employing robust multivariate techniques like CFS for reliable performance.

6.1 Hypothesis Testing

6.1.1 Hypothesis 1

The hypothesis aims to address RQ1, which is to determine if there is a significant difference between machine learning models for software fault prediction with feature selection and machine learning models without feature selection. If $P < 0.05$, the null hypothesis can be rejected, which is defined as no significant differences between models with and without feature selection.

To evaluate the impact of different feature selection methods (RFE, L1 Regularization, MI, CFS, No Feature Selection) on machine learning performance, we conducted a one-way ANOVA and the results are presented in table 6.8.

Table 6.6: Validation results for accuracy from cross-validation, computed for Logistic Regression using feature selection method, and tuning algorithms. The values represent the mean accuracy and standard deviation across 10 validation folds.

Logistic Regression			
Feature Selection	**Optimization**	**Mean Accuracy**	**Std. Dev.**
RFE	Grid Search	81.74	±2.1%
	Randomised Search	83.21	±1.8%
	Genetic Algorithm	83.92	±1.6%
L1 Regularization	Grid Search	81.26	±1.9%
	Randomised Search	80.71	±1.6%
	Genetic Algorithm	80.73	±1.5%
MI	Grid Search	80.98	±2.3%
	Randomised Search	81.67	±2.0%
	Genetic Algorithm	83.04	±1.8%
CFS	Grid Search	82.12	±2.0%
	Randomised Search	83.04	±1.7%
	Genetic Algorithm	84.15	±1.5%

The hypothesis evaluates the effect of different feature selection methods on model performance. This involves multiple independent groups (RFE, L1 Regularization, MI, CFS, and No Feature Selection), where the objective is to compare their mean accuracies. One-way ANOVA is specifically designed for comparing the means of multiple groups. The one-way ANOVA is well-suited for comparing the means of three or more independent groups to see if there is a statistically significant difference between them.

In this study, the confidence level was set at 0.05, corresponding to a significance level of 5%. This means there is a 5% chance of incorrectly rejecting the null hypothesis (Type I error). A significance level of 0.05 is commonly used in scientific research, as it balances the risk of Type I errors—where false positives might dominate with the risk of Type II errors, which occur when failing to reject a false null hypothesis [49]. A p-value of less than 0.05 indicates statistical significance, suggesting that the observed results are unlikely to have occurred by chance.

The results indicate that fault prediction models that employ feature selection are more accurate than those that do not. Statistical significance of the observed variations in accuracy between models with and without feature selection using the One-

Table 6.7: Validation results for accuracy from cross-validation, computed for SVM using feature selection method, and tuning algorithms. The values represent the mean accuracy and standard deviation across 10 validation folds.

SVM			
Feature Selection	**Optimization**	**Mean Accuracy**	**Std. Dev.**
RFE	Grid Search	85.12	±1.8%
	Randomised Search	86.25	±1.5%
	Genetic Algorithm	87.41	±1.4%
L1 Regularization	Grid Search	80.20	±1.5%
	Randomised Search	80.71	±1.3%
	Genetic Algorithm	80.73	±1.1%
MI	Grid Search	81.56	±2.1%
	Randomised Search	82.34	±1.8%
	Genetic Algorithm	83.21	±1.6%
CFS	Grid Search	82.88	±1.7%
	Randomised Search	83.76	±1.5%
	Genetic Algorithm	85.12	±1.3%

way ANOVA test was evaluated, as the model accuracy data did not follow a normal distribution (see Fig. 5.2). The ANOVA results revealed significant differences in accuracy, with a p-value of less than 0.001. This confirms the hypothesis that feature selection significantly impacts the performance of machine learning models for software fault prediction.

Table 6.8 shows that Correlation-based Feature Selection (CFS) significantly outperformed other methods, achieving the highest accuracy across Random Forest, Logistic Regression, and SVM. L1 regularization and Mutual Information (MI) followed in terms of performance.

Although the shortest training time was observed when no feature selection was applied for SVM, this method sacrificed accuracy. L1 regularization effectively balanced training time and accuracy, making it suitable for time-sensitive scenarios. Models without feature selection and those using L1 regularization had the fastest testing times across all models, but their accuracy was lower compared to those utilizing CFS and Recursive Feature Elimination (RFE).

The results support the hypothesis that feature selection significantly impacts the performance of software fault prediction. Among the methods evaluated, CFS con-

Feature Selection	**ML Model**	**Accuracy (%)**	**Training Time (ms)**	**Testing Time (ms)**
Recursive Feature Elimination (RFE)	Random Forest	76.2	6.0	0.15
	Logistic Regression	69.0	1.8	0.04
	SVM	71.8	5.5	0.12
L1 Regularization	Random Forest	80.16	3.5	0.08
	Logistic Regression	69.25	0.9	0.03
	SVM	70.002	3.0	0.06
Mutual Information(MI)	Random Forest	75.5	4.5	0.1
	Logistic Regression	68.4	1.2	0.03
	SVM	72.1	3.5	0.03
Correlation-based Feature Selection (CFS)	Random Forest	81.32	4.0	0.01
	Logistic Regression	70.04	1.5	0.035
	SVM	75.35	3.2	0.05
Without Feature selection	Random Forest	70.21	7.0	0.18
	Logistic Regression	60.47	6.0	0.15
	SVM	63.05	2.2	0.49

Table 6.8: Accuracy's of the models with and without Feature selection

sistently provided the best balance of accuracy and computational efficiency. The findings suggest that feature selection is essential for improving the robustness and scalability of fault prediction models.

6.1.2 Hypothesis 2

To address RQ2, three search optimization algorithms—Grid Search, Randomized Search, and Genetic Algorithm—were selected for parameter tuning. As shown in Table 6.9, after performing parameter tuning using search optimization algorithms, the accuracy of the software fault prediction model has significantly improved compared to models without feature selection and parameter tuning (refer to Table 6.9).

The hypothesis is to test whether there were differences in performance across different hyperparameter tuning methods (Grid Search, Randomized Search, Genetic Algorithm) which is different from Hypothesis 1 . ANOVA assumes that the data within each group follows a normal distribution; however, the data chosen for this research does not meet this criterion 5.2. In contrast, the Kruskal-Wallis Test

Model	**Grid Search Accuracy (%)**	**Randomized Search Accuracy (%)**	**Genetic Algorithm Accuracy (%)**	**Accuracy without Tuning (%)**
Random Forest	80.16	86.32	82.06	70.21
Logistic Regression	81.26	81.27	83.26	60.47
SVM	81.53	81.53	82.32	63.05

Table 6.9: Comparison of Model Accuracies With and Without Parameter Tuning

is a non-parametric test that does not assume a specific distribution of the data. Instead, it ranks the data and evaluates whether the groups originate from the same distribution, making it more robust against non-normality.

Additionally, ANOVA requires homogeneity of variances, meaning the variances within each group should be approximately equal. The Kruskal-Wallis Test, on the other hand, can accommodate scenarios where the variance in model performance (accuracy) differs across various search algorithms, providing greater flexibility when variances are unequal. This makes Kruskal-Wallis Test more suitable for this hypothesis as it does not assume normality or equal variances.

The Kruskal-Wallis Test was used to perform a statistical verification. In this instance, the groups examined were the hyperparameter tuning techniques (Grid Search, Randomized Search, Genetic Algorithm), and the Kruskal-Wallis Test statistic of 7.26 indicates a significant difference between them. The p-value of 0.0265 exceeds the standard alpha threshold of 0.05. This suggests that the null hypothesis can be rejected, indicating a statistically significant variation in the improvement of machine learning model accuracy using different search algorithms. This implies that when optimizing model parameters for software fault prediction, different tuning techniques perform differently.

6.1.3 Hypothesis 3

The results presented in Tables 6.2, 6.3, 6.4, 6.1, and 6.10 indicate that both feature selection and tuning significantly impact model efficiency and accuracy. CFS consistently yields the highest accuracy across models, particularly when paired with the Genetic Algorithm, achieving an accuracy of 88.4%. Additionally, L1 Regularization combined with Randomized Search emerges as the most time-efficient method.

From Table 6.8, it is clear that feature selection methods enhance accuracy, with CFS and RFE being the most effective. Furthermore, the Genetic Algorithm produces higher accuracy compared to both Grid Search and Randomized Search. Simpler techniques like L1 Regularization effectively reduce training time, and Randomized Search also decreases training time relative to Grid Search and the Genetic Algorithm. Ultimately, the combination of L1 Regularization and Randomized Search minimizes training time, while MI paired with Randomized Search proves to be the most efficient for testing.

For the Random Forest model, the combination of CFS and the Genetic Algorithm achieves the highest accuracy of 88.4%, although it increases computational costs, as seen in Table 6.4. For applications where time is critical, Logistic Regression with L1 Regularization and Randomized Search strikes a good balance between accuracy and efficiency. The findings reveal that the interaction of feature selection and hyperparameter tuning results in better outcomes than using either technique independently.

In conclusion, optimized feature selection and tuning are essential for improving software fault prediction performance. The results suggest that different combinations can be tailored based on the application's priorities: for accuracy, use CFS combined with the Genetic Algorithm; for speed, use L1 Regularization with Randomized Search.

Model	**Training Time (s)**	**Prediction Time (s)**	**Training Accuracy (%)**	**Testing Accuracy (%)**
RF	7.0	0.18	81.52	70.21
LR	6.0	0.15	73.67	60.47
SVM	2.2	0.49	75.24	63.05

Table 6.10: Machine learning models' performance without feature selection and search optimizations to analyze overfitting.

Chapter 7

Discussion

7.1 Analysis of Results and Their Significance

7.1.1 Key Findings

- CFS consistently delivered the highest accuracy (88.40% with Random Forest and Genetic Algorithm) by preserving multivariate feature interactions and minimizing redundancy, showcasing its strength for complex models like Random Forest and SVM.
- Genetic Algorithm achieved the best hyperparameter tuning results, particularly when paired with CFS, optimizing model configurations to enhance accuracy and robustness
- Randomized Search provided a practical alternative for time-sensitive applications, achieving competitive accuracy with significantly lower computational costs compared to Grid Search and Genetic Algorithm.
- The integration of feature selection and hyperparameter tuning resulted in models with reduced overfitting, higher testing accuracy, and consistent cross-validation performance, particularly for Random Forest and SVM.
- Feature selection and tuning had varying impacts on models, with Random Forest excelling due to its reliance on feature interactions, SVM benefitting from kernel-specific tuning, and Logistic Regression showing moderate improvements with sparse feature selection.

7.1.2 Does using Feature Selection and Parameter Tuning worth it?

7.1.2.1 Feature Selection Analysis

The study evaluated four feature selection methods: L1 Regularization, Mutual Information (MI), Recursive Feature Elimination (RFE), and Correlation-Based Feature selection (CFS), and analyzed their impact on model performance metrics such as accuracy, training time, and testing time. Each method selected a unique subset of features based on its evaluation criteria, which significantly influenced the performance of the three baseline models (Random Forest, Logistic Regression, and SVM).

L1 Regularization:
L1 Regularization proved effective in identifying relevant features by assigning positive weights to features like wmc, ce, cam, ic, cbm, amc, and avg_cc, which positively contributed to fault prediction, while also selecting negatively weighted features such as cbo, rfc, and loc, which showed high predictive importance despite inverse relationships (Table 2.2). This method achieved moderate accuracy across models, with a peak accuracy of 86.32% when combined with Random Forest and Randomized Search (Table 6.2). L1 Regularization was computationally efficient, achieving notably shorter training and testing times, particularly for simpler models like Logistic Regression and SVM (e.g., 0.9 seconds training time for Logistic Regression).

L1 Regularization's ability to create sparse feature sets by assigning zero weights to irrelevant features benefits models that thrive on fewer, high-impact attributes, such as Logistic Regression and SVM. Furthermore, its focus on linear relationships aligns well with features like wmc, cbo, and loc, explaining its moderate success in these models. The smaller feature set also reduced computational overhead, making L1 Regularization suitable for scenarios requiring computational efficiency. However, L1 Regularization struggled with models like Random Forest, which rely on feature interactions. Its univariate approach limited its ability to capture these interactions, leading to inconsistent accuracy gains and highlighting its limitations for ensemble-based models (Table 6.2, 2.2, 6.8).

Mutual Informtaion(MI):
Mutual Information (MI) selected a diverse set of features (from Table 2.3), with top-ranked attributes including rfc, loc, wmc, cbo, ce, lcom3, amc, dam, max_cc, and cam, while excluding lower-ranked features such as ca, dit, noc, and ic due to their weak associations with the target variable. MI demonstrated competitive accuracy, particularly with Random Forest, achieving 87.00% when combined with Genetic Algorithm (table 6.3). The training and testing times for MI were higher than those of L1 Regularization but comparable to Recursive Feature Elimination (RFE).

As a univariate method, MI evaluates features individually based on their dependency with the target variable, effectively identifying strong predictors such as rfc, loc, and wmc, which contributed to improved model performance. However, its univariate nature limited its ability to capture feature interactions, leading to the exclusion of features like ca and noc, which could have improved performance in interaction-reliant models like Random Forest and SVM (from Table 2.3, 6.3). Despite this, MI excelled in identifying nonlinear dependencies, as evidenced by features like dam and max_cc, making it particularly effective for models capable of handling such relationships, such as Random Forest and SVM. Nevertheless, MI was less effective for Logistic Regression, which relies on linear dependencies, as its inability to consider multivariate relationships limited its potential, causing its accuracy to trail behind CFS and RFE.

Recursive Feature Elimination(RFE):
Recursive Feature Elimination (RFE) proved to be an effective feature selection method, retaining critical attributes such as wmc, rfc, ce, cbo, cam, and loc, which

significantly contribute to software complexity and fault-proneness (Table 2.1). Meanwhile, features like noc, mfa, and moa were excluded due to their weaker correlations with the target variable. This selective process led to notable accuracy improvements, with RFE achieving 87.41% accuracy for Random Forest when paired with Genetic Algorithm tuning (Table 6.1). However, RFE's iterative nature resulted in higher training times, as demonstrated by the 10.3 seconds required for Random Forest with Grid Search.

By systematically removing the least important features, RFE prioritized high-impact attributes such as wmc and cbo, which enhanced the accuracy of models like Random Forest and SVM (Table 6.1). Moreover, its ability to preserve feature interactions made it particularly effective for models that rely on complex relationships between features. Despite these advantages, the computational expense of the iterative process limits RFE's practicality for real-time or large-scale datasets, making it less efficient compared to methods like L1 Regularization and MI (Table 6.2, 6.3).

Correlation-Based Feature Selection (CFS):
Correlation-Based Feature Selection (CFS) demonstrated superior performance by consistently selecting features with high relevance to the target variable while minimizing redundancy. Key features such as wmc, cbo, loc, rfc, and ce were retained due to their strong correlation with fault prediction, whereas redundant features like noc, ca, and lcom3 were excluded (from Table 2.4). CFS achieved the highest accuracy of 88.40% when paired with Random Forest and Genetic Algorithm, significantly outperforming other methods (Table 6.4). Training times for CFS were moderate, exemplified by 12.5 seconds for Random Forest with Genetic Algorithm, while testing times remained efficient, ensuring practicality in real-time applications.

The multivariate nature of CFS allowed it to evaluate feature subsets based on both relevance and redundancy, effectively preserving critical feature interactions necessary for models like Random Forest and SVM. Features such as wmc, cbo, and ce enhanced Random Forest's ensemble learning by emphasizing interaction-driven predictors. Furthermore, CFS demonstrated robustness, with the lowest variability in cross-validation results ($\pm 1.0\%$) (Table 6.5), confirming its reliability and generalizability across different data splits. While CFS required slightly higher computational resources compared to simpler methods like MI and L1 Regularization, its superior accuracy and robustness justified the additional computational cost.

From the research, it is evident that multivariate methods like CFS and RFE consistently outperformed univariate methods like MI and L1 Regularization, particularly for interaction-reliant models like Random Forest. The ability of CFS to eliminate redundant features while preserving interactions led to the highest accuracy gains across all models.

Univariate methods (L1 Regularization, MI) required less computational effort, leading to shorter training times. Conversely, the iterative nature of RFE and the multivariate evaluation of CFS increased training times but delivered superior performance.

7.1.2.2 Parameter Tuning using Search Optimization techniques

The model's performance depends on hyperparameters, which can be adjusted using search optimization techniques. The main methods examined are Grid Search, Randomized Search, and Genetic Algorithms.

Grid Search:
Grid Search provided robust and reliable results by systematically evaluating all possible combinations of hyperparameters. For instance, it achieved an accuracy of 85.76% for Random Forest with CFS 6.4 but required 9.5 seconds of training time, making it more computationally expensive compared to Randomized Search. The exhaustive nature of Grid Search ensures that the best configuration within the predefined parameter grid is identified, providing a dependable baseline for comparison.

However, the higher computational cost of Grid Search limits its practicality for large-scale tasks. In this study, the method demonstrated its strength in simpler models like Logistic Regression, where it identified optimal parameters 2.6 such as C = 0.1, penalty = l2, and solver = liblinear, achieving consistent performance across different feature selection methods. For SVM, Grid Search optimized parameters 2.7 like C = 10 and kernel = rbf, enhancing non-linear pattern detection. Despite its computational demands, Grid Search remains a valuable tool for scenarios requiring exhaustive evaluation and accuracy assurance.

Randomised Search:
Randomized Search offered a balanced trade-off between accuracy and computational efficiency, making it suitable for scenarios with limited computational resources. For example, with Random Forest and RFE, Randomized Search achieved an accuracy of 85.25% in just 8.7 seconds 6.1 of training time, outperforming Grid Search in efficiency. By sampling hyperparameters randomly from specified distributions, Randomized Search reduces computational costs while maintaining accuracy levels close to exhaustive methods.

For Logistic Regression, Randomized Search found configurations like C = 0.1, penalty = l1, and solver = liblinear, which enhanced model sparsity and interpretability. This led to efficient training times, particularly for datasets processed with L1 Regularization 6.2 or MI 6.3. Similarly, for SVM, Randomized Search optimized parameters such as gamma = 0.01 and C = 5, which enabled the model to balance complexity and generalization effectively. While Randomized Search may miss the absolute best configuration due to its stochastic nature, it remains a practical choice for resource-constrained environments or real-time applications.

Genetic Algorithm:
Genetic Algorithm consistently achieved the highest accuracy across all feature selection methods and models, albeit at the cost of increased training times. For instance, when combined with CFS, it delivered an accuracy of 88.40% for Random Forest 6.4,

the highest among all tuning methods. This performance stems from the iterative nature of the Genetic Algorithm, which explores a broad hyperparameter space and optimizes configurations to maximize accuracy. The flexibility of the Genetic Algorithm allowed it to align seamlessly with the multivariate nature of CFS, enabling the selection of parameters such as n_estimators = 300, max_depth = 40, and criterion = entropy. These parameters leveraged the ensemble learning capabilities of Random Forest, ensuring robust performance.

However, the computational cost of the Genetic Algorithm is higher due to its evolutionary optimization process, which evaluates multiple generations of hyperparameter combinations. Despite this, its ability to discover high-performing configurations makes it indispensable for high-stakes tasks requiring maximum accuracy and reliability. For SVM, the Genetic Algorithm's fine-tuning of parameters such as C = 20 and gamma = 0.005 resulted in improved decision boundary formation, leading to superior performance compared to simpler tuning methods.

7.1.3 Impact of feature selection + parameter tuning

Parameter tuning acted as a catalyst by enhancing the strengths of feature selection methods and compensating for their limitations. This synergy allowed models to achieve higher accuracy, reduced overfitting, and better computational efficiency. For example, Recursive Feature Elimination (RFE), known for iteratively retaining key features such as wmc, rfc, and loc, critical predictors of software faults benefited significantly from parameter tuning. While RFE's iterative nature is computationally expensive and prone to diminishing returns on its own, efficient tuning strategies, particularly the Genetic Algorithm, magnified its impact 6.1. By optimizing hyperparameters such as n_estimators, max_depth, and min_samples_split, the Genetic Algorithm amplified RFE's contribution, enabling Random Forest to achieve an accuracy of 87.41%, a substantial improvement over its default configurations.

For linear models like Logistic Regression, which inherently rely on linear relationships, parameter tuning played an even more pivotal role in adapting feature selection outputs. Although methods like RFE and MI provided modest improvements in accuracy 6.8, parameter tuning optimized their utility 6.3, 6.1. For instance, Randomized Search effectively identified configurations such as C = 0.1 and penalty = l1, which enhanced model sparsity and reduced overfitting. This tuning-step synergy was especially evident with L1 Regularization, which provided a sparse feature set and minimized computational overhead. As a result, Logistic Regression achieved efficient training times (e.g., 0.9 seconds with Randomized Search) without sacrificing accuracy.

Dimensionality reduction through feature selection shortened training times 6.8, but parameter tuning ensured that the selected features were leveraged to their full potential. While univariate methods like MI and L1 Regularization reduced noise, their lack of consideration for feature interactions limited their impact. In contrast, multivariate methods such as Correlation-Based Feature Selection (CFS) not only preserved critical feature interactions but also aligned exceptionally well with advanced tuning techniques like the Genetic Algorithm 6.4. This alignment allowed Random Forest with CFS to achieve the highest accuracy of 88.40%, demonstrating the complementary nature of feature selection and parameter tuning.

Additionally, parameter tuning significantly stabilized the performance of multivariate feature selection methods. Models tuned with the Genetic Algorithm consistently exhibited lower variability across cross-validation folds 6.5, 6.6, 6.7. For instance, Random Forest combined with CFS and Genetic Algorithm achieved a standard deviation of only ±1.0% in accuracy, showcasing its reliability across different data splits. This stability stems from CFS's ability to preserve interaction-driven features, which the Genetic Algorithm further optimized to balance exploration and exploitation of the hyperparameter space.

In contrast, univariate feature selection methods such as MI and L1 Regularization exhibited higher variability in ensemble models like Random Forest, where feature interactions are critical. For example, MI's inability to capture multivariate dependencies caused it to overlook important interactive features. Although Randomized Search compensated for some of these limitations by identifying optimal hyperparameter configurations, the variability remained higher compared to multivariate methods like CFS.

Critical Balance Between Computational Cost and Performance: Multivariate methods like CFS and RFE delivered higher accuracy and robustness but incurred higher computational costs due to their more complex evaluation strategies. However, advanced tuning techniques like Genetic Algorithm justified these costs by optimizing configurations that maximized the utility of these feature sets. For instance, even though CFS-based models had moderate training times (e.g., 12.5 seconds for Random Forest with Genetic Algorithm), their testing times remained practical (e.g., 17.6 ms), making them suitable for real-time applications 6.4.

On the other hand, univariate methods such as MI and L1 Regularization, when paired with simpler tuning strategies like Randomized Search, achieved shorter training times and computational efficiency. However, this came at the expense of accuracy and stability in interaction-reliant models like Random Forest. These findings underscore the importance of tailoring feature selection and parameter tuning combinations to the specific requirements of the model and dataset.

7.2 Limitations

While this research introduces an enhanced machine learning framework for software fault prediction, it is important to acknowledge certain limitations that may influence the findings and applicability of the results.

Dataset: The models developed in this study were evaluated on a dataset containing approximately 6,981 rows. While this size is adequate for academic research, in real-world applications, software projects may contain much larger datasets, some-times with millions of lines of code. The scalability of the models and the computational overhead

Generalization: The models developed in this research were trained and validated on data from specific Java projects, and while stratified cross-validation was used to ensure the robustness of the results, there are inherent challenges in generalizing the findings across different software domains. The software metrics used in this study may have varying importance depending on the type of software being developed (e.g., web applications versus embedded systems).

Computational Resources: The application of search optimization techniques such as Genetic Algorithms and Randomized Search can be computationally expensive, especially when dealing with large datasets or complex models. While the study achieved promising results, the resource demands for hyperparameter tuning could be prohibitive in real-time or resource-constrained environments.

Chapter 8

Conclusions and Future Work

The aim of this thesis was to evaluate the combined impact of feature selection and search optimization strategies on the performance of machine learning models for software fault prediction. The findings demonstrate that these strategies not only enhance accuracy but also improve computational efficiency and model robustness when tailored to specific use cases.

This study uniquely addresses the interaction between feature selection and parameter tuning in software fault prediction, an area often overlooked in previous research. By combining multivariate feature selection methods with iterative tuning strategies, the thesis demonstrates how to achieve both high accuracy and computational efficiency, bridging a critical gap in the field.

The thesis established that multivariate feature selection methods, such as Correlation-Based Feature Selection (CFS), are particularly effective in preserving interaction-driven features, enabling ensemble models like Random Forest to achieve the highest accuracy (88.40%) when paired with the Genetic Algorithm. This combination also showed exceptional robustness, with a minimal cross-validation variability of ± 1.0In contrast, univariate methods such as Mutual Information (MI) and L1 Regularization prioritized computational efficiency. The integration of Randomized Search with sparse methods like L1 Regularization achieved substantial reductions in training times (e.g., 0.9 seconds for Logistic Regression), while maintaining accuracy within 2-3% of more exhaustive methods. This highlights the feasibility of efficient hyperparameter tuning in real-time fault prediction scenarios. However, the inability of univariate methods to preserve feature interactions limited their performance in models that rely on such relationships, such as Random Forest.

A key finding is the importance of tailoring feature selection and parameter tuning combinations to the specific requirements of the model. While Random Forest benefitted most from multivariate methods due to its reliance on feature interactions, Logistic Regression achieved the best results with sparse methods like L1 Regularization. SVM demonstrated substantial improvements when paired with kernel-specific tuning, particularly with Genetic Algorithm optimizing parameters such as gamma and C.

The thesis also highlights a critical trade-off between computational cost and predictive performance. While Genetic Algorithm consistently delivered the best results by exploring a broader hyperparameter space and aligning with feature selection outputs, its computational overhead made it less suitable for time-sensitive applications. Conversely, Randomized Search offered a practical balance, achieving near-optimal accuracy with significantly reduced training times, as seen in Random Forest with RFE (85.25% accuracy in 8.7 seconds). The accuracy of 88.40% achieved by combining CFS with Genetic Algorithm represents a notable improvement in the context of fault prediction datasets characterized by high dimensionality and multicollinearity. This finding underscores the potential of tailored feature selection and tuning strategies to address the unique challenges of software fault prediction.

This study fills a significant gap in the literature by exploring the combinatorial effects of feature selection and parameter tuning in software fault prediction. The findings underscore the importance of aligning hyperparameter configurations with feature selection outputs to achieve robust and generalizable models. For high-stakes scenarios requiring maximum accuracy, the use of multivariate methods like CFS with advanced tuning strategies is recommended. In contrast, for resource-constrained environments, univariate methods combined with efficient search techniques like Randomized Search provide a viable alternative.

8.1 Future Work

The thesis examines the impact of search optimization and feature selection on the performance of machine learning models. It has demonstrated significant improvements in model accuracy and robustness by implementing strategies such as Grid Search, Randomized Search, and Genetic Algorithms for hyperparameter tuning, as well as using L1 Regularization, CFS, MI, and RFE for feature selection.

Future work should explore the application of these techniques on larger and more diverse datasets, including datasets from different programming languages and software architectures.

As fault prediction models are used in critical software engineering processes, improving their interpretability is vital. Future studies could Employ explainable AI (XAI) methods like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to identify the contribution of selected features to predictions. Focus on visualization tools that demonstrate the impact of feature selection and hyperparameter tuning on the decision-making process of the models.

To further enhance model performance, future studies could explore combining advanced feature selection techniques like Mutual Information, Recursive Feature Elimination (RFE), and hybrid methods that integrate filter and wrapper approaches. When paired with ensemble optimization techniques such as Particle Swarm Optimization or Bayesian Optimization, these methods could make feature selection and hyperparameter tuning more comprehensive and effective. Additionally, hybrid optimization strategies—such as a combination of Bayesian Optimization and Genetic

Algorithms—could improve the efficiency and effectiveness of the search process by balancing exploration and exploitation of the hyperparameter space.

While using Bayesian Optimization offers potential benefits, they were not implemented in this thesis due to their high computational complexity and the practical time constraints of the research. Bayesian Optimization (BO) builds a probabilistic model, often a Gaussian Process (GP), to approximate the objective function [16]. This process is inherently non-linear due to the use of covariance functions in the GP. However, in the context of simpler baseline models like Linear Regression or Logistic Regression, which are inherently linear and involve fewer hyperparameters, the computational overhead of BO may not be justified [97, 99]. Implementing these advanced and computationally intensive methods would require substantial computational resources and extended experimental time, which may be beyond the scope of this study. Future work could focus on leveraging these advanced techniques to develop more robust, interpretable, and high-performing models for software fault prediction in contexts where resources permit.

References

[1] "Software defect," 2018. [Online]. Available: https://dx.doi.org/10.21227/H2K078

[2] A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu, "Better software analytics via "duo": Data mining algorithms using/used-by optimizers," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2099–2136, 2020.

[3] T. N. AL-Hadidi and S. O. Hasoon, "Software defect prediction using extreme gradient boosting (xgboost) with optimization hyperparameter," *AL-Rafidain Journal of Computer Sciences and Mathematics*, vol. 18, no. 1, 2024.

[4] O. Al Qasem and M. Akour, "Software fault prediction using deep learning algorithms," *International Journal of Open Source Software and Processes (IJOSSP)*, vol. 10, no. 4, pp. 1–19, 2019.

[5] M. Ali, T. Mazhar, T. Shahzad, Y. Y. Ghadi, S. M. Mohsin, S. M. A. Akber, and M. Ali, "Analysis of feature selection methods in software defect prediction models," *IEEE Access*, vol. 11, pp. 145 954–145 974, 2023.

[6] S. A. Ali, N. R. Roy, and D. Raj, "Software defect prediction using machine learning," in *2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)*, 2023, pp. 639–642.

[7] B. ALSANGARI and G. BİRCİK, "Performance evaluation of various ml techniques for software fault prediction using nasa dataset," in *2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2023, pp. 1–7.

[8] H. Alsghaier and M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier," *Software: Practice and Experience*, vol. 50, no. 4, pp. 407–427, 2020.

[9] F. AlShaikh and W. Elmedany, "Estimate the performance of applying machine learning algorithms to predict defects in software using weka," in *4th Smart Cities Symposium (SCS 2021)*, vol. 2021, 2021, pp. 189–194.

[10] F. Amini and G. Hu, "A two-layer feature selection method using genetic algorithm and elastic net," *Expert Systems with Applications*, vol. 166, p. 114072, 2021.

[11] A. Auger and B. Doerr, "Theory of randomized search heuristics: Foundations and recent developments," 2011.

[12] A. O. Balogun, S. Basri, S. A. Jadid, S. Mahamad, M. A. Al-momani, A. O. Bajeh, and A. K. Alazzawi, "Search-based wrapper feature selection methods

in software defect prediction: an empirical analysis," in *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9*. Springer, 2020, pp. 492–503.

[13] I. Batool and T. A. Khan, "Software fault prediction using deep learning techniques," *Software Quality Journal*, vol. 31, no. 4, pp. 1241–1280, 2023.

[14] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[15] G. P. Bhandari and R. Gupta, "Machine learning based software fault prediction utilizing source code metrics," in *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, 2018, pp. 40–45.

[16] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, no. 2, p. e1484, 2023.

[17] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011.

[18] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, no. 8, pp. 1040–1058, 2009.

[19] C. Cui, B. Liu, and G. Li, "A novel feature selection method for software fault prediction model," in *2019 Annual Reliability and Maintainability Symposium (RAMS)*, 2019, pp. 1–6.

[20] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, "A deep tree-based model for software defect prediction," *arXiv preprint arXiv:1802.00921*, 2018.

[21] H. Das, B. Naik, and P. D. H. Behera, "Optimal selection of features using artificial electric field algorithm for classification," *Arabian Journal for Science and Engineering*, vol. 46, 03 2021.

[22] S. Das, A. Paramane, S. Chatterjee, and U. M. Rao, "Recursive feature elimination aided accurate fault classification in power transformers using dissolved gas data," in *2022 IEEE 6th International Conference on Condition Assessment Techniques in Electrical Systems (CATCON)*, 2022, pp. 192–195.

[23] R. Dhanya, I. R. Paul, S. Sindhu Akula, M. Sivakumar, and J. J. Nair, "A comparative study for breast cancer prediction using machine learning and feature selection," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, 2019, pp. 1049–1055.

[24] S. Feng, J. Keung, X. Yu, Y. Xiao, K. E. Bennin, M. A. Kabir, and M. Zhang, "Coste: Complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology*, vol. 129, p. 106432, 2021.

[25] W. Fu, T. Menzies, and X. Shen, “Tuning for software analytics: Is it really necessary?” *Information and Software Technology*, vol. 76, pp. 135–146, 2016.

[26] H. Ghinaya, R. Herteno, M. R. Faisal, A. Farmadi, and F. Indriani, “Analysis of important features in software defect prediction using synthetic minority oversampling techniques (smote), recursive feature elimination (rfe) and random forest,” *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 3, pp. 276–288, 2024.

[27] S. Glamocak, “Feature importance in imbalanced binary classification with ensemble methods,” Ph.D. dissertation, Technische Universität Wien, 2023.

[28] N. Gopika and A. M. Kowshalaya M.E., “Correlation based feature selection algorithm for machine learning,” in *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*, 2018, pp. 692–695.

[29] S. Goyal, “Effective software defect prediction using support vector machines (svms),” *International Journal of System Assurance Engineering and Management*, vol. 13, no. 2, pp. 681–696, 2022.

[30] K. Guo, X. Wan, L. Liu, Z. Gao, and M. Yang, “Fault diagnosis of intelligent production line based on digital twin and improved random forest,” *Applied Sciences*, vol. 11, no. 16, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/16/7733

[31] R. Gupta, “Software defects prediction using support vector machine,” *International Journal of Computer Science and Software Engineering*, vol. 1, no. 1, pp. 39–48, 2015.

[32] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[33] M. A. Hall, “Correlation-based feature selection for machine learning,” Ph.D. dissertation, The University of Waikato, 1999.

[34] T. Hall and D. Bowes, “The state of machine learning methodology in software fault prediction,” in *2012 11th international conference on machine learning and applications*, vol. 2. IEEE, 2012, pp. 308–313.

[35] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, “Deepjit: an end-to-end deep learning framework for just-in-time defect prediction,” in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 34–45.

[36] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, “Mifs-nd: A mutual information-based feature selection method,” *Expert systems with applications*, vol. 41, no. 14, pp. 6371–6385, 2014.

[37] A. Iqbal, S. Aftab, U. Ali, Z. Nawaz, L. Sana, M. Ahmad, and A. Husen, “Performance analysis of machine learning techniques on software defect prediction using nasa datasets,” *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, 2019.

[38] A. Iqbal, S. Aftab, I. Ullah, M. S. Bashir, and M. A. Saeed, "A feature selection based ensemble classification framework for software defect prediction," *International Journal of Modern Education and Computer Science*, vol. 10, no. 9, p. 54, 2019.

[39] M. Jagtap, P. Katragadda, and P. Satelkar, "Software reliability: Development of software defect prediction models using advanced techniques," in *2022 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 2022, pp. 1–7.

[40] Y. Jiang and B. Cukic, "Misclassification cost-sensitive fault prediction models," in *Proceedings of the 5th international conference on predictor models in software engineering*, 2009, pp. 1–10.

[41] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, pp. 561–595, 2008.

[42] A. Jindal, A. Gupta *et al.*, "Comparative analysis of software reliability prediction using machine learning and deep learning," in *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*. IEEE, 2022, pp. 389–394.

[43] S. Kaliraj, A. M. Kishoore, and V. Sivakumar, "Software fault prediction using cross-project analysis: A study on class imbalance and model generalization," *IEEE Access*, vol. 12, pp. 64 212–64 227, 2024.

[44] A. Khan, A. Azim, R. Liscano, K. Smith, Q. Tauseef, G. Seferi, and Y.-K. Chang, "Machine learning-based test case prioritization using hyperparameter optimization," in *2024 IEEE/ACM International Conference on Automation of Software Test (AST)*, 2024, pp. 125–135.

[45] V. Kreinovich, C. Quintana, and O. Fuentes, "Genetic algorithms: what fitness scaling is optimal?" *Cybernetics and Systems*, vol. 24, no. 1, pp. 9–26, 1993.

[46] H. Kumar and H. Das, "Software fault prediction using wrapper based feature selection approach employing genetic algorithm," in *2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON)*, 2023, pp. 1–7.

[47] K. V. Kumar, P. Kumari, A. Chatterjee, and D. P. Mohapatra, "Software fault prediction using random forests," in *Intelligent and Cloud Computing*, D. Mishra, R. Buyya, P. Mohapatra, and S. Patnaik, Eds. Singapore: Springer Singapore, 2021, pp. 95–103.

[48] T. S. Kumar and B. Booba, "A systematic study on machine learning techniques for predicting software faults," in *2021 IEEE Mysore Sub Section International Conference (MysuruCon)*. IEEE, 2021, pp. 133–136.

[49] S. Kwak, "Are only p-values less than 0.05 significant? a p-value greater than 0.05 is also significant!" *Journal of Lipid and Atherosclerosis*, vol. 12, no. 2, p. 89, 2023.

[50] K. Lakra and A. Chug, "Development of efficient and optimal models for software maintainability prediction using feature selection techniques," in *2021 8th*

International Conference on Computing for Sustainable Global Development (INDIACom), 2021, pp. 798–803.

[51] A. Lambora, K. Gupta, and K. Chopra, “Genetic algorithm- a literature review,” in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 380–384.

[52] K. Larsen, J. H. Petersen, E. Budtz-Jørgensen, and L. Endahl, “Interpreting parameters in the logistic regression model with random effects,” *Biometrics*, vol. 56, no. 3, pp. 909–914, 2000.

[53] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” *IEEE transactions on software engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[54] J. Li, P. He, J. Zhu, and M. R. Lyu, “Software defect prediction via convolutional neural network,” in *2017 IEEE international conference on software quality, reliability and security (QRS)*. IEEE, 2017, pp. 318–328.

[55] H. Liang, Y. Yu, L. Jiang, and Z. Xie, “Seml: A semantic lstm model for software defect prediction,” *IEEE Access*, vol. 7, pp. 83 812–83 824, 2019.

[56] W. Liu, S. Liu, Q. Gu, X. Chen, and D. Chen, “Fecs: A cluster based feature selection method for software fault prediction with noises,” in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2, 2015, pp. 276–281.

[57] R. Mahajan, S. K. Gupta, and R. K. Bedi, “Design of software fault prediction model using br technique,” *Procedia Computer Science*, vol. 46, pp. 849–858, 2015.

[58] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Applied Soft Computing*, vol. 27, pp. 504–518, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494614005857

[59] R. Malhotra and M. Cherukuri, “A systematic review of hyperparameter tuning techniques for software quality prediction models,” *Intelligent Data Analysis*, no. Preprint, pp. 1–19.

[60] R. Malhotra and A. Negi, “Reliability modeling using particle swarm optimization,” *International Journal of System Assurance Engineering and Management*, vol. 4, 09 2013.

[61] S. Mehta and K. S. Patnaik, “Improved prediction of software defects using ensemble machine learning techniques,” *Neural Computing and Applications*, vol. 33, no. 16, pp. 10 551–10 562, 2021.

[62] Meiliana, S. Karim, H. L. H. S. Warnars, F. L. Gaol, E. Abdurachman, and B. Soewito, “Software metrics for fault prediction using machine learning approaches: A literature review with promise repository dataset,” in *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, 2017, pp. 19–23.

[63] S. Mondal, A. K. Sahu, H. Kumar, R. M. Pattanayak, M. K. Gourisaria, and H. Das, "Software fault prediction using wrapper based ant colony optimization algorithm for feature selection," in *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*, 2023, pp. 1–6.

[64] R. Muhammad, A. Nadeem, and M. A. Sindhu, "Vovel metrics—novel coupling metrics for improved software fault prediction," *PeerJ Computer Science*, vol. 7, p. e590, 2021.

[65] M. Nevendra and P. Singh, "Empirical investigation of hyperparameter optimization for software defect count prediction," *Expert Systems with Applications*, vol. 191, p. 116217, 2022.

[66] N. Nikravesh and M. R. Keyvanpour, "Parameter tuning for software fault prediction with different variants of differential evolution," *Expert Systems with Applications*, vol. 237, p. 121251, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423017530

[67] H. Osman, M. Ghafari, and O. Nierstrasz, "Automatic feature selection by regularization to improve bug prediction accuracy," in *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, 2017, pp. 27–32.

[68] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: A survey," *Expert Systems with Applications*, vol. 172, p. 114595, 2021.

[69] K. Phung, E. Ogunshile, and M. E. Aydin, "Enhancing software fault prediction with error-type metrics: A risk-based approach," *Available at SSRN 4529342*.

[70] T. M. Phuong Ha, D. Hung Tran, L. T. My Hanh, and N. Thanh Binh, "Experimental study on software fault prediction using machine learning model," in *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, 2019, pp. 1–5.

[71] C. Pornprasit and C. K. Tantithamthavorn, "Deeplinedp: Towards a deep learning approach for line-level defect prediction," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 84–98, 2022.

[72] F. Provost, "Machine learning from imbalanced data sets 101," in *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, vol. 68, no. 2000. AAAI Press, 2000, pp. 1–3.

[73] N. Pudjihartono, T. Fadason, A. W. Kempa-Liehr, and J. M. O'Sullivan, "A review of feature selection methods for machine learning-based disease risk prediction," *Frontiers in Bioinformatics*, vol. 2, p. 927312, 2022.

[74] O. A. Qasem, M. Akour, and M. Alenezi, "The influence of deep learning algorithms factors in software fault prediction," *IEEE Access*, vol. 8, pp. 63 945–63 960, 2020.

[75] S. K. Rath, M. Sahu, S. P. Das, S. K. Bisoy, and M. Sain, "A comparative analysis of svm and elm classification on software reliability prediction model," *Electronics*, vol. 11, no. 17, p. 2707, 2022.

[76] S. S. Rathore and Kumar, “Software fault prediction based on the dynamic selection of learning technique: findings from the eclipse project study,” *Applied Intelligence*, vol. 51, no. 12, pp. 8945–8960, 2021.

[77] S. S. Rathore and S. Kumar, “A study on software fault prediction techniques,” *Artificial Intelligence Review*, vol. 51, pp. 255–327, 2019.

[78] S. S. Rathore, S. S. Chouhan, D. K. Jain, and A. G. Vachhani, “Generative oversampling methods for handling imbalanced data in software fault prediction,” *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 747–762, 2022.

[79] S. Riaz, A. Arshad, and L. Jiao, “Rough noise-filtered easy ensemble for software fault prediction,” *IEEE Access*, vol. 6, pp. 46 886–46 899, 2018.

[80] Y. Saeys, I. Inza, and P. Larrañaga, “A review of feature selection techniques in bioinformatics,” *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 08 2007. [Online]. Available: https://doi.org/10.1093/bioinformatics/btm344

[81] A. M. Salem, K. Rekab, and J. A. Whittaker, “Prediction of software failures through logistic regression,” *Information and Software Technology*, vol. 46, no. 12, pp. 781–789, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584903002155

[82] E. Scornet, “Tuning parameters in random forests,” *ESAIM: Proceedings and surveys*, vol. 60, pp. 144–162, 2017.

[83] R. T. Selvi and P. Patchaiammal, “Fault prediction for large scale projects using deep learning techniques,” in *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 2022, pp. 482–489.

[84] S. Shi, “Feature selection and case selection methods based on mutual information in software cost estimation,” 2014.

[85] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *Advances in neural information processing systems*, vol. 25, 2012.

[86] J. Sohn, Y. Kamei, S. McIntosh, and S. Yoo, “Leveraging fault localisation to enhance defect prediction,” in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 284–294.

[87] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, “Software defect association mining and defect correction effort prediction,” *IEEE Transactions on Software Engineering*, vol. 32, no. 2, pp. 69–82, 2006.

[88] I. Syarif, A. Prugel-Bennett, and G. Wills, “Svm parameter optimization using grid search and genetic algorithm to improve classification performance,” *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 14, no. 4, pp. 1502–1509, 2016.

[89] T. Tahir, Ç. Gencel, G. Rasool, U. Tariq, J. Rasheed, S. F. Yeo, and T. Cevik, “Early software defects density prediction: training the international software benchmarking cross projects data using supervised learning,” *IEEE Access*, 2023.

[90] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “Automated parameter optimization of classification techniques for defect prediction models,” in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 321–332.

[91] N. S. Thomas and S. Kaliraj, “An improved and optimized random forest based approach to predict the software faults,” *SN Computer Science*, vol. 5, no. 5, p. 530, 2024.

[92] R. S. Wahono, N. S. Herman, and S. Ahmad, “Neural network parameter optimization based on genetic algorithm for software defect prediction,” *Advanced Science Letters*, vol. 20, no. 10-11, pp. 1951–1955, 2014.

[93] V. Walunj, G. Gharibi, R. Alanazi, and Y. Lee, “Defect prediction using deep learning with network portrait divergence for software evolution,” *Empirical Software Engineering*, vol. 27, no. 5, p. 118, 2022.

[94] H. Wang and T. M. Khoshgoftaar, “A study on software metric selection for software fault prediction,” in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019, pp. 1045–1050.

[95] P. Wang, C. Jin, and S.-W. Jin, “Software defect prediction scheme based on feature selection,” in *2012 Fourth International Symposium on Information Science and Engineering*, 2012, pp. 477–480.

[96] T. Wang, Z. Zhang, X. Jing, and L. Zhang, “Multiple kernel ensemble learning for software defect prediction,” *Automated Software Engineering*, vol. 23, pp. 569–590, 2016.

[97] X. Wang, Y. Jin, S. Schmitt, and M. Olhofer, “Recent advances in bayesian optimization,” *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–36, 2023.

[98] X. Xiaolong, C. Wen, and W. Xinheng, “Rfc: a feature selection algorithm for software defect prediction,” *Journal of Systems Engineering and Electronics*, vol. 32, no. 2, pp. 389–398, 2021.

[99] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[100] Z. Yang and H. Qian, “Automated parameter tuning of artificial neural networks for software defect prediction,” in *Proceedings of the 2nd International Conference on Advances in Image Processing*, ser. ICAIP ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 203–209. [Online]. Available: https://doi.org/10.1145/3239576.3239622

[101] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, “A comparison of optimization methods and software for large-scale l1-regularized linear classification,” *The Journal of Machine Learning Research*, vol. 11, pp. 3183–3234, 2010.

[102] A. Zakrani, M. Hain, and A. Idri, “Improving software development effort estimating using support vector regression and feature selection,” *IAES International Journal of Artificial Intelligence*, vol. 8, no. 4, p. 399, 2019.

[103] R. Zebari, A. Abdulazeez, D. Zeebaree, D. Zebari, and J. Saeed, “A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction,” *Journal of Applied Science and Technology Trends*, vol. 1, no. 1, pp. 56–70, 2020.

[104] C. Zhang, P. Soda, J. Bi, G. Fan, G. Almpanidis, S. García, and W. Ding, “An empirical study on the joint impact of feature selection and data resampling on imbalance classification,” *Applied Intelligence*, vol. 53, no. 5, pp. 5449–5461, 2023.

[105] C. Zhang, Y. Li, Z. Yu, and F. Tian, “Feature selection of power system transient stability assessment based on random forest and recursive feature elimination,” in *2016 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, 2016, pp. 1264–1268.

Appendix A
Supplemental Information

SDLC	- Software Development Life Cycle
ML	- Machine learning
SFP	- Software Fault Prediction
RF	- Random Forest
LR	- Logistic Regression
SVM	- Support Vector Machine
GA	- Genetic Algorithm
ANN	- Artificial Neural Network
RFE	- Recursive Feature Elimination
MI	- Mutual Information
CFS	- Correlation-Based Feature Selection
OOB	- Out-of-bag
NGEN	- Number of Generations
CXPB	- Crossover Probability
MUTPB	- Mutation Probability
ADASYN	- Adaptive Synthetic Sampling
TP	- True Positive
TN	- True Negative
FP	- False Positive
FN	- False Negative
ANOVA	- Analysis Of Variance