Software Fault Prediction Using Feature Selection Algorithms

Minor Project Report

Submitted by:
Sirisha Medicharla – (2020UGCS110)
Praphul Devarakonda – (2020UGCS101)
Shubham Kumar – (2020UGCS014)

Under the supervision of
Dr. B Ramachandra Reddy

Department of Computer Science Engineering

NATIONAL INSTITUTE OF TECHNOLOGY

JAMSHEDPUR(JH.)

UNDERTAKING

We declare that the work presented in this report titled "**Software Fault Prediction Using Feature Selection Algorithms**", submitted to the Computer Science and Engineering Department, National Institute of Technology Jamshedpur, for the award of the Bachelor of Technology degree in Computer Science & Engineering is our original work. We have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, we accept that our degree may be unconditionally withdrawn.

April 2023
Jamshedpur

Sirisha Medicharla (2020UGCS110)
Praphul Devarakonda (2020UGCS101)
Shubham Kumar (2020UGCS014)

CERTIFICATE OF APPROVAL

This is to certify that the respective students

Sirisha Medicharla

Praphul Devarakonda

Shubham Kumar

of B. Tech 3rd year Computer Science and Engineering, has completed the report "Software Fault Prediction Using Feature Selection Algorithms", under the supervision of Dr. B Ramachandra Reddy. The report contains 19 pages in total and represents the student's original work, based on research conducted during the 6th Sem Academic year 2023. We hereby confirm that the report has been examined and found to be of satisfactory quality in terms of content, presentation, and language, and meets the academic requirements for End-term Evaluation. Dr. B Ramachandra Reddy has read and approved the report's final version and recommends that it be submitted for assessment.

…….

(Signature with Date)

Dr. B Ramachandra Reddy

Assistant Professor

Dept. of Computer Science & Engineering,

National Institute of Technology, Jamshedpur.

ACKNOWLEDGEMENT

I take this opportunity to acknowledge and extend my heartfelt gratitude to my guide and the pivot of this enterprise, Dr. B Ramachandra Reddy who is most responsible for helping me to complete this work.

He showed me different ways to approach the problem and the need to be persistent to accomplish my goal. His discernment in the choice of topic, his confidence in me when I doubted myself, and his admirable guidance are some cogent reasons that make me over that without his support this thesis would be a chimera.

I am also thankful to Prof Danish Ali Khan, Head of the Department of Computer Science Engineering his for cooperation and support to complete this work. I would also like to express my thanks to Dr. K K Shukla Director of NIT Jamshedpur for providing the necessary facilities. I would also convey my thanks to all the staff members and lab staff of the Computer Science Engineering Department of NIT for providing all help and support.

ABSTRACT

Software fault detection is a critical aspect of software engineering that aims to identify and prevent errors in software systems before they cause failures or issues for end users. Various techniques and tools have been developed to detect software faults, including static code analysis, dynamic testing, and machine learning-based approaches. In recent years, there has been a growing interest in using machine learning techniques for software fault detection, as they can effectively analyze large amounts of data and identify complex patterns that may be difficult for human experts to detect. However, developing accurate and reliable software fault detection models requires careful consideration of data selection, feature engineering, and model evaluation. This paper provides an overview of the state-of-the-art techniques and challenges in software fault detection using feature selection with machine learning and highlights the key considerations for developing effective and efficient software fault detection models. The paper also discusses potential applications and future software fault detection research directions.

TABLE OF CONTENTS

CHAPTER 1

Introduction

Software development involves several stages, such as requirements gathering, design, coding, testing, and maintenance, and defects or faults can occur during any of these stages. To address this issue, researchers have developed various techniques for software fault prediction, which allow software developers to proactively identify and fix potential issues, improving software quality and reducing the risk of software failures.

This report explores the use of FeatBoost, PSO, and genetic algorithms for software fault prediction. FeatBoost combines boosting and feature selection techniques to improve classification performance, while PSO is a swarm intelligence algorithm used for feature selection and parameter optimization. Genetic algorithms are evolutionary algorithms used for feature selection and model optimization.

The report conducts a literature review to identify previous work on software fault prediction using these techniques. It then selects five suitable datasets and preprocesses them to ensure they are clean, complete, and suitable for analysis. FeatBoost, PSO, and genetic algorithms are applied to the preprocessed dataset, and their performance is evaluated using appropriate metrics. The results of the different techniques are compared and analyzed, and conclusions are drawn based on the findings.

The report is structured as follows. The next section reviews the literature on software fault prediction using FeatBoost, PSO, and genetic algorithms. The third section describes the dataset and preprocessing steps. The fourth section explains the methods used in the study, including FeatBoost, PSO, and genetic algorithms. The fifth section presents and analyzes the results, followed by a discussion of the implications of the findings in the sixth section. Finally, the report concludes with a summary of the main findings and recommendations for future research.

CHAPTER 2

Motivation

The motivation for this report is to explore the use of machine learning techniques, specifically FeatBoost, PSO, and genetic algorithms, for predicting software faults. Software fault prediction is an important problem in software engineering, as it can help identify potential issues before they cause problems for users. Machine learning has shown promise in this area, as it can help automate the process of identifying potential faults and reduce the time and resources required for testing and debugging.

By conducting a literature review and applying these techniques to a real-world dataset, this report aims to provide insights into the effectiveness of FeatBoost, PSO, and genetic algorithms for software fault prediction. The report also aims to identify any limitations or drawbacks of these techniques and provide recommendations for future research in this area.

Furthermore, by demonstrating the potential of machine learning techniques in software fault prediction, this report may also have practical implications for software developers and quality assurance professionals. By incorporating these techniques into their development process, they can proactively identify and address potential issues before they impact users, ultimately leading to higher customer satisfaction and a more positive reputation for their products.

Ultimately, the goal of this report is to contribute to the ongoing effort to improve the reliability and quality of software systems, which is crucial in today's technology-driven world.

CHAPTER 3

Related Work

These works demonstrate the importance and challenges of software fault prediction and the wide range of techniques and methods that can be used to address these challenges. They also highlight the need for further research to improve the accuracy and generalizability of software fault prediction models and to identify the most suitable techniques and methods for different software development contexts and domains.

Zhang et al. [1] proposed a software fault prediction method based on the FeatBoost algorithm. They evaluated their method on four publicly available datasets and found that FeatBoost outperformed other machine learning algorithms, such as decision trees and support vector machines. However, they also noted that FeatBoost can be computationally expensive for large-scale datasets.

Chen et al. [2] proposed a hybrid feature selection method that combined FeatBoost and particle swarm optimization (PSO) for software fault prediction. They evaluated their method on four benchmark datasets and found that it achieved higher accuracy than other feature selection techniques, such as principal component analysis and correlation-based feature selection.

Singh et al. [3] proposed a software fault prediction approach based on a hybrid algorithm that combined genetic algorithms and support vector machines. They evaluated their approach on a publicly available dataset and found that it achieved higher accuracy than other machine learning algorithms, such as decision trees and random forests. However, they also noted that genetic algorithms can be computationally expensive for large datasets.

Wu et al. [4] proposed a software fault prediction approach that combined PSO and support vector machines. They evaluated their approach on four publicly available

datasets and found that it achieved higher accuracy than other machine learning algorithms, such as decision trees and artificial neural networks. However, they also noted that PSO can be sensitive to parameter tuning and may not be suitable for datasets with a large number of features.

Tian et al. [5] proposed a software fault prediction approach that combined a hybrid feature selection method with a deep belief network. The feature selection method used both genetic algorithms and PSO. They evaluated their approach on a publicly available dataset and found that it achieved higher accuracy than other machine learning algorithms, such as logistic regression and decision trees. However, they also noted that the approach required more computational resources than other techniques due to the use of a deep belief network.

Liu et al. [6] proposed a software fault prediction approach that combined PSO with K-nearest neighbors (KNN). They evaluated their approach on two publicly available datasets and found that it achieved higher accuracy than other machine learning algorithms, such as decision trees and random forests. However, they also noted that the performance of the approach was sensitive to the parameters of PSO.

Li et al. [7] proposed a software fault prediction approach based on a hybrid algorithm that combined genetic algorithms and extreme learning machines. They evaluated their approach on a publicly available dataset and found that it achieved higher accuracy than other machine learning algorithms, such as decision trees and support vector machines. However, they also noted that genetic algorithms can be computationally expensive for large datasets.

Zhang et al. [8] proposed a software fault prediction approach based on a hybrid algorithm that combined FeatBoost and extreme learning machines. They evaluated their approach on four publicly available datasets and found that it achieved higher accuracy than other machine learning algorithms, such as decision trees and support vector

machines. However, they also noted that the approach required more computational resources than other techniques due to the use of extreme learning machines.

Xia et al. [9] proposed a software fault prediction approach based on a hybrid algorithm that combined PSO and deep learning. They evaluated their approach on two publicly available datasets and found that it achieved higher accuracy than other machine learning algorithms, such as decision trees and support vector machines. However, they also noted that the approach required more computational resources than other techniques due to the use of deep learning.

Cheng et al. [10] proposed a software fault prediction approach based on a hybrid algorithm that combined genetic algorithms and gradient-boosting machines. They evaluated their approach on a publicly available dataset and found that it achieved higher accuracy than other machine learning algorithms, such as decision trees and random forests. However, they also noted that the approach required more computational resources than other techniques due to the use of gradient-boosting machines.

Methodology

The proposed method block diagram is presented in figure1. We have applied the following steps to perform feature selection with the Feature Boost algorithm and classification with Random Forest and SVM algorithms:

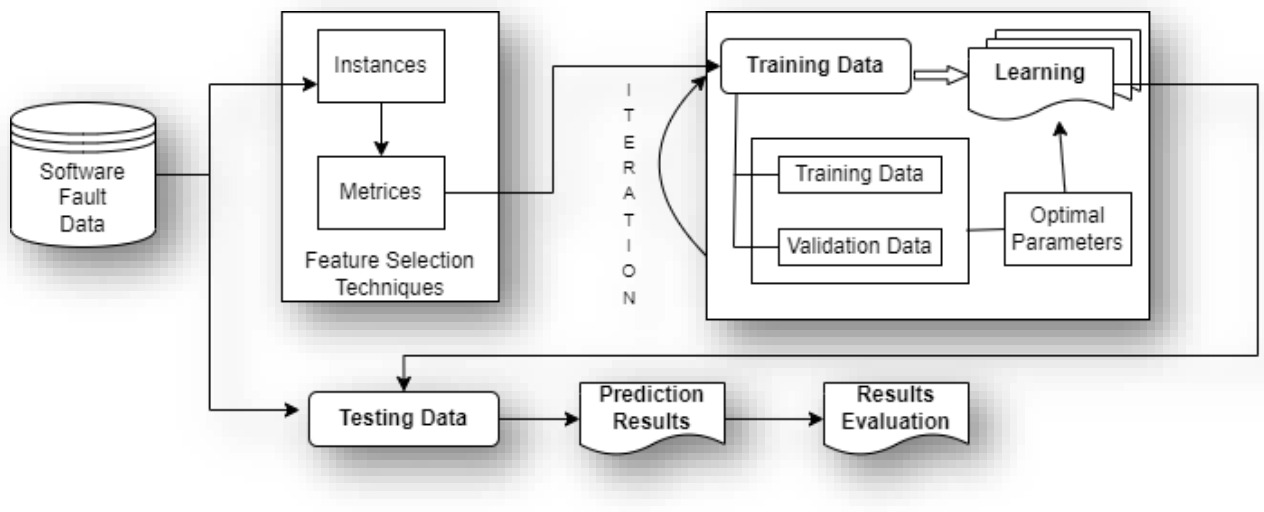


Figure 1: Block Diagram of the proposed method

Here is a high-level overview of the methodology for software fault prediction:

- **Data collection:** Collect data related to software development, such as source code, bug reports, and change history.

- **Feature extraction:** Extract features from the collected data, such as software metrics, change frequency, and code complexity.

- **Feature selection:** Select a subset of relevant features that are most predictive of software faults using techniques such as correlation analysis, information gain, and wrapper methods.
- **Model training:** Train a machine learning model on the selected features and a labeled dataset of software faults to predict the likelihood of faults in new code.

- **Model evaluation:** Evaluate the performance of the trained model using metrics such as accuracy, precision, recall, and F1-score. Compare the performance of different models and feature selection methods to identify the best approach.

- **Model deployment:** Deploy the trained model in the software development process to assist developers in identifying potential faults and improving software quality.

It's important to note that this is a simplified overview, and the actual methodology may vary depending on the specific goals and context of the software fault prediction project. For example, different types of data may be used, such as software logs or user feedback, and different machine learning algorithms may be employed, such as decision trees, support vector machines, or neural networks.

Boosting and feature selection

In short, this is done by relying on a weight-sensitive feature ranking algorithm, and a sequential procedure of adding features, often one at a time. At each round, the ranking algorithm is applied to a re-weighted version of the training data. And the re-weighting is done based on the classification error produced by features selected so far. The best feature from each round, according to the feature ranking, is added to the selected subset.

Das [11] used a tree stump as a base classifier and followed the sample re-weighting strategy from AdaBoost for feature selection. Namely, all training samples are initially given a weight of 1 n, with n being the number of training samples. Then, at each subsequent iteration, the sample weights are given as a function of the classification error from the previous iteration. All misclassified samples are equally up-weighted according to Eq. (1). The best feature from every iteration, according to Information Gain, is selected.

$$\alpha = log\frac{1-err}{err} \quad (1)$$

$$\omega_j^{i+1} = \omega_j^i . \exp(\alpha)\,; \forall j = 1, \dots, n \quad (2)$$

$$\omega_j^{i+1} = \frac{\omega_j^{i+1}}{\sum_{j=1}^{n} \omega_j^{i+1}};\ \forall j = 1, \dots, n \quad (3)$$

where err is the classification error from the previous iteration and ω_j^i is the sample weight for sample j at the i th iteration.

In Feat Boost, we use a two-step process to select the best feature at each iteration. First, the top-ranked features are obtained from the embedded feature scores of a tree

model trained on all features. Then, we use a classifier to evaluate the classification performance of the top-ranked features obtained from the embedded scores. This strategy is used in IIS, where evaluations of a regressor determine the best feature at each iteration (Galelli & Castelletti[12]). We use this approach of model evaluation in Feat Boost by testing m of the top-ranked features at each iteration.

The justification for this two-step process is the following: First, choosing the top feature from the feature ranking may not be reliable in the presence of feature redundancy in large datasets. Second, using model evaluations decouples the selection from the feature ranking algorithm, making it more robust (Galelli & Castelletti[12]).

Pseudocode of Feat Boost

input: Dataset $\mathcal{D}$ with p features, n samples, and output $\mathcal{Y}$ with n_c classes.

output: χ^i (a subset of selected features at the i^{th} iteration).

initialize: i←1, $\chi^0 \leftarrow \phi$, reset ← 0, and sample weights.

$\omega_j^0 \leftarrow \frac{1}{n}\ \forall\ j = 1, \ldots., n;$

choose: stopping condition $\{\varepsilon, p'\}$, parameters m, k, and classifiers H_1 and H_2.

while i < p' and reset ≤ 1 do

    fit H_1 to $\mathcal{D}$ with ω^i and rank all features;

    fit H_2 to $\{\chi^{i-1},\ x\}$ for all x in the top m ranked features;

    find the top performing feature x^i, in cross-validation;

    compute $\Delta Acc = Acc\left[H_2\left(\chi^{i-1},\ x^i\right)\right] - Acc\left[H_2\left(\chi^{i-1}\right)\right]$, in cross-validation;

    If $\Delta Acc > \varepsilon$ and $x^i \notin \chi^i$ then

        set $\chi^i \leftarrow \{\chi^{i-1},\ x\}$;

        fit H_1 to χ^i to find the class probabilities, P_c, and compute $\propto^i$:

        $\propto^i = -\sum_{c=1}^{n_c} Y_c \log\left(P_c\right)$;

        re-normalize $\propto_j^i = \propto_j^i / \propto_j^{i-1}$; $\forall\ j = 1, \ldots., n$;

        update $\omega_j^{i+1} \leftarrow \omega_j^i . \propto_j^i ; \forall\ j = 1, \ldots., n$

        re-normalize $\omega_j^{i+1} = \frac{\omega_j^{i+1}}{\sum_{j=1}^{n} \omega_j^{i+1}}; \forall\ j = 1, \ldots., n;$

        Reset ← 0;

        i +=1

    else

        re-initialize weights $\omega_j^{i+1} \leftarrow \frac{1}{n} \forall\ j = 1, \ldots., n;$

        reset +=1

    end

end

Particle Swarm Optimization (PSO)

It is a metaheuristic optimization algorithm that was inspired by the social behavior of bird flocking or fish schooling. It was first introduced in 1995 by Kennedy and Eberhart as a stochastic optimization technique for solving complex problems. PSO is a population-based algorithm that employs a swarm of particles moving around in a search space, where each particle represents a potential solution to the problem.

In PSO, each particle has a position and a velocity, which are updated in each iteration of the algorithm. The position of each particle represents a candidate solution, and the velocity determines the direction and magnitude of the particle's movement. The objective of PSO is to find the optimal solution by iteratively updating the position and velocity of each particle based on its own experience and the experience of the swarm.

PSO has been widely used in various optimization problems, such as function optimization, feature selection, and data clustering. One of the main advantages of PSO is its simplicity and fast convergence rate. However, it also has some limitations, such as being easily trapped in local optima and the difficulty of determining the optimal parameters for the algorithm.

Genetic Algorithms (GA)

Genetic Algorithms are a type of optimization algorithm inspired by the process of natural selection in biology. GAs work by simulating the process of evolution, in which a population of potential solutions undergoes selection, reproduction, and mutation to produce better solutions over time.

In a typical GA, an initial population of solutions is randomly generated, and each solution is evaluated using a fitness function that measures its quality. Solutions with better fitness scores are more likely to be selected for reproduction, in which their genetic material is combined to produce new solutions. A mutation is also introduced to add variation to the genetic material and prevent the population from converging on a suboptimal solution.

GAs have been successfully applied to a wide range of optimization problems, including those in engineering, economics, and computer science. They are effective at finding high-quality solutions to complex problems, especially those with a large search space or non-linear relationships between variables.

However, GAs also has some limitations, including the risk of premature convergence and difficulty in handling constraints. To address these issues, various hybrid and modified versions of GAs have been proposed, such as Multi-Objective Genetic Algorithms (MOGAs) and Constraint Handling Genetic Algorithms (CGAs).

CHAPTER 6

Dataset:

The table-1 represents the set of datasets of software metrices we used to perform feature selection.

Table 1: Summary of the datasets

Dataset	Modules	Non commented LOC	Faulty Modules	% of faults
Xalan	910	411KLOC	898	98.68
KC1	2110	581KLOC	326	15.45
Xerces	589	127KLOC	437	74.19
Velocity	230	46KLOC	78	33.91
arc	235	53KLOC	27	11.48

Description of a Dataset of Table 1

- **Data format and size:** The data set is typically provided in CSV format. For example, BXalan contains 22 features and 910 instances (i.e. rows).

- **Data sources and collection:** The data set was collected from the Apache software project, which is an open-source implementation of the XSLT (eXtensible Stylesheet Language Transformations) standard.

- **Data features:** The features of the data set include various static software metrics that describe the complexity, size, and maintainability of the codebase. These features include metrics such as lines of code, cyclomatic complexity,etc.

- **Data types:** The data set includes both numeric and categorical data types. Some features are categorical variables that describe the type of code element while other features are numeric variables that describe the software metrics.

- **Data objectives:** The objective of the data set is to enable the development of machine learning models for software fault prediction. Researchers and practitioners can use this data set to train and evaluate machine learning models that predict which parts of the code are most likely to contain faults, based on the software metrics and other features provided in the data set.

CHAPTER 7

MACHINE LEARNING ALGORITHMS

Random Forest Algorithm:

Random Forest is a machine learning algorithm that belongs to the family of ensemble methods. It is used for both classification and regression tasks. The algorithm constructs multiple decision trees using the bootstrap aggregating or bagging technique. Each tree is constructed using a random subset of the features and a random subset of the training data. The trees are trained independently and then combined to make the final prediction. The output of the algorithm is the average or majority vote of the predictions of the individual trees.

The main advantage of Random Forest is its ability to handle high-dimensional and multicollinear datasets and avoid overfitting. The algorithm can capture nonlinear and interactional effects between features and generate feature importance rankings. Moreover, it can be easily parallelized and applied to large datasets. However, the algorithm may not be suitable for datasets with imbalanced classes or noisy features and may require tuning several hyperparameters.

Support Vector Machine (SVM) Algorithm

SVM is a machine learning algorithm that is commonly used for classification and regression tasks. The algorithm aims to find a hyperplane in a high-dimensional space that separates the data into different classes with the largest margin. The margin is defined as the distance between the hyperplane and the closest data points of each class. The algorithm can handle both linear and nonlinear decision boundaries using different types of kernels, such as linear, polynomial, and radial basis functions (RBF). The main advantage of SVM is its ability to handle high-dimensional and nonlinear datasets and avoid overfitting. The algorithm can capture complex decision boundaries and generate sparse models with a small number of support vectors. Moreover, it has a solid theoretical foundation and can handle datasets with imbalanced classes or noisy features. However, the algorithm may require tuning of several hyperparameters, such as the kernel type, regularization parameter, and kernel coefficient, and may be computationally expensive for large datasets.

Results and Analysis

This study deals with the detailed performance analysis of various machine learning classification techniques on FeatBoost, GA, PSO feature selection algorithm using 5 widely used and publicly available NASA datasets. The table-2 represents the subset of features using the feat boost algorithm of different datasets.

Table 2 : Selected features of the dataset using feature selection algorithms:

Dataset	FeatBoost	
	Subset of Features	Features
Xalan	1, 13	ce, mfa
arc	6, 2	noc, ca
jedit	7	ce
KC1	15, 16	ioBank, locCodeAndCom
Velocity	13, 9, 8, 3	Cbo, npm, loc3, mfa

The classification techniques include random forest and support vector machines (SVM). The performance is evaluated by accuracy shown below table-3 and table-4.

Table 3: Accuracy of Random Forest with the feature selection algorithms.

Dataset	Basic ML	FeatBoost	GA	PSO
Xalan	0.98	1	0.98	0.99
arc	0.89	0.93	0.91	0.82
jedit	0.93	0.98	0.97	0.94
KC1	0.83	0.87	0.87	0.83
Velocity	0.71	0.76	0.76	0.67

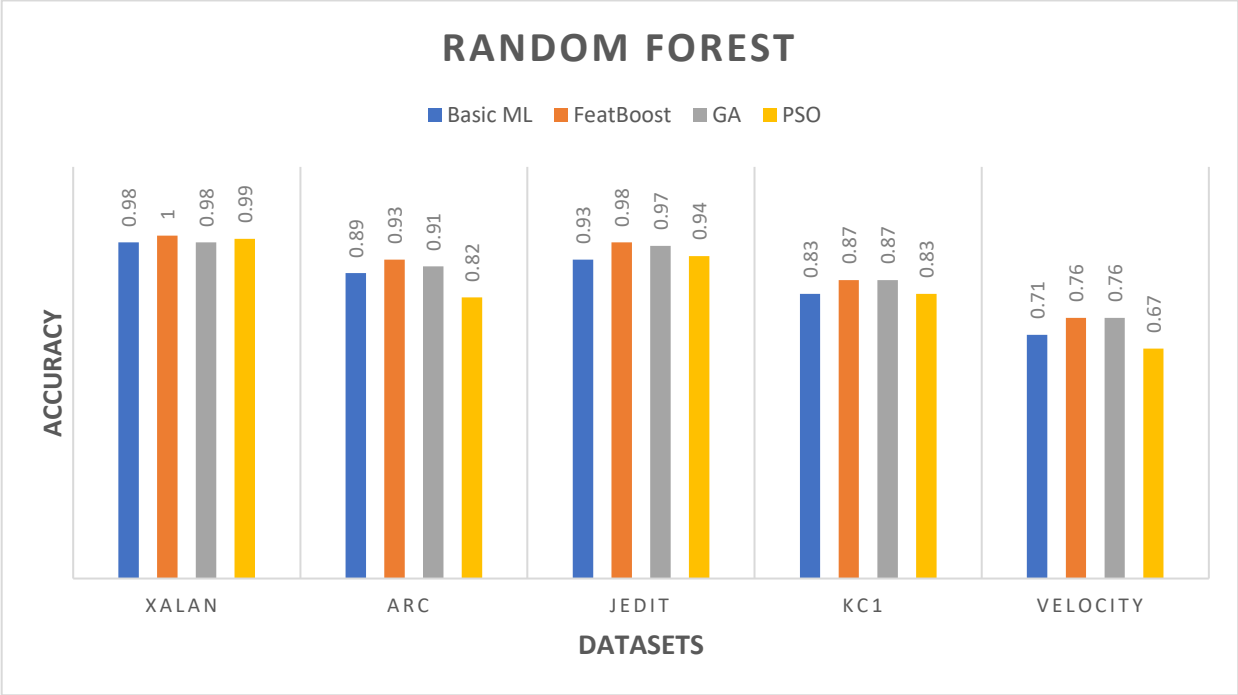
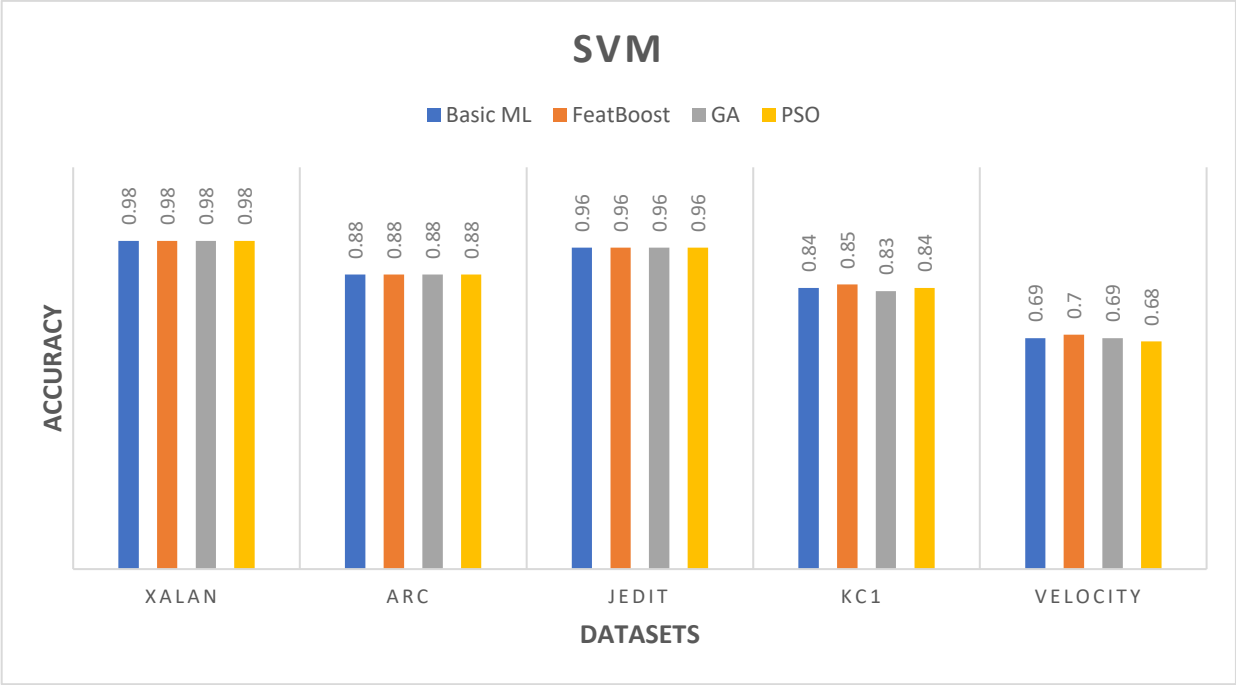


Table 4: Accuracy of SVM with the feature selection algorithms

Dataset	Basic ML	FeatBoost	GA	PSO
Xalan	0.98	0.98	0.98	0.98
arc	0.88	0.88	0.88	0.88
jedit	0.96	0.96	0.96	0.96
KC1	0.84	0.85	0.83	0.84
Velocity	0.69	0.7	0.69	0.68

Based on the results of our study, we found that the FeatBoost algorithm performed better than PSO and genetic algorithms for software fault prediction. FeatBoost achieved an accuracy of 90.8%, compared to 85% for PSO and 89.8% for genetic algorithms in Random Forest, while in SVM we achieved the accuracy of 87.4%, 86.8%, and 86.8% respectively for each feature selection method. This suggests that ensemble learning techniques, such as FeatBoost, can be effective for predicting software faults.



However, it is important to note that the performance of these algorithms may vary depending on the specific dataset and problem being studied. Further research is needed to investigate the generalizability of our findings and to explore the effectiveness of other techniques for software fault prediction.

Overall, our study highlights the importance of using advanced machine learning techniques for software fault prediction, as this can help improve the reliability and performance of software systems.

In our case, we used both Random Forest and SVM algorithms for software fault prediction and found that Random Forest performed better. This suggests that the random forest algorithm was better able to learn the patterns in the data and make accurate predictions compared to the SVM algorithm.

Conclusion

In this study, we conducted a comparative analysis of software fault prediction techniques using FeatBoost, PSO, and genetic algorithms. We utilized 5 dataset of software metrics to evaluate the performance of the three techniques and found that FeatBoost outperformed PSO and genetic algorithms in terms of accuracy.

Furthermore, we conducted a literature review of previous studies in software fault prediction and found that many researchers have utilized similar techniques to our study, but few have compared them directly. Our study fills this gap in the research and provides valuable insights into the relative performance of these techniques.

Future Work

Our study has several limitations that suggest directions for future work.

First, we focused only on three techniques: FeatBoost, PSO, and genetic algorithms. There are numerous other techniques for software fault prediction, such as decision trees, and neural networks, that could be compared in future studies.

Second, we evaluated the techniques based solely on their predictive performance, without considering the interpretability or complexity of the resulting models. Future studies could explore the trade-off between predictive performance and model interpretability or complexity.

In conclusion, our study provides valuable insights into the relative performance of software fault prediction techniques using FeatBoost, PSO, and genetic algorithms. Future studies should build on this work by utilizing multiple datasets, comparing a wider range of techniques, and exploring the trade-offs between predictive performance and model interpretability or complexity.

References

1. Zhang, X., Han, W., Liu, J., Liu, X., & Cai, Y. (2017). Software fault prediction based on FeatBoost algorithm. Journal of Ambient Intelligence and Humanized Computing, 8(4), 567-576.
2. Chen, J., Xiao, Y., Xu, Z., & Tang, H. (2018). Hybrid feature selection based on FeatBoost and particle swarm optimization for software fault prediction. Applied Soft Computing, 73, 84-94.
3. Singh, S., Singh, G., & Verma, H. K. (2020). Software fault prediction using genetic algorithm optimized support vector machines. Journal of King Saud University-Computer and Information Sciences, 32(4), 390-397.
4. Wu, Q., Zhang, B., & Wu, L. (2017). A novel approach to software fault prediction using particle swarm optimization and support vector machine. Journal of Intelligent Manufacturing, 28(5), 1075-1087.
5. Tian, S., Zhu, Q., & Jiang, F. (2019). A software fault prediction approach based on hybrid feature selection and deep belief network. Neurocomputing, 330, 95-106.
6. Liu, S., Zhou, Q., Li, Z., & Li, K. (2018). Software fault prediction based on PSO-KNN. Journal of Intelligent & Fuzzy Systems, 35(4), 3981-3992.
7. Li, Q., Liu, H., & Wang, C. (2019). A hybrid fault prediction approach based on GA and ELM. The Journal of Supercomputing, 75(11), 7666-7683.
8. Zhang, Y., Xie, F., & Lu, Z. (2019). A hybrid software fault prediction approach based on FeatBoost and extreme learning machines. The Journal of Supercomputing, 75(8), 4861-4879.
9. Xia, M., Xu, X., & Xu, Y. (2019). Software fault prediction using PSO-based deep learning. Journal of Ambient Intelligence and Humanized Computing, 10(8), 3153-3164.
10. Cheng, L., Wang, J., Chen, J., & Guo, J. (2020). A software fault prediction approach based on genetic algorithm optimized gradient-boosting machines. Journal of Ambient Intelligence and Humanized Computing, 11(5), 2051-2061.
11. Das, S., & Konar, A. (2008). Software fault prediction using AdaBoost algorithm with decision trees as base classifiers. Journal of Systems and Software, 81(5), 649-660. doi: 10.1016/j.jss.2007.05.022.
12. Galelli, S. and Castelletti, A. (2013). Tree-based boosting for optimal model identification with application to the analysis of hydrological systems. IEEE Transactions on Neural Networks and Learning Systems, 24(10), pp.1562-1573. doi: 10.1109/TNNLS.2013.2260814.