The Scientific Temper (2024) Vol. 15 (3): 2709-2718
E-ISSN: 2231-6396, ISSN: 0976-8653
Doi: 10.58414/SCIENTIFICTEMPER.2024.15.3.39
https://scientifictemper.com/

RESEARCH ARTICLE

A novel approach for metrics-based software defect prediction using genetic algorithm

Rajeev P. R.*, K. Aravinthan

Abstract

Software defect prediction is an important issue in the process of software development and maintenance, which is related to the overall success or failure of software. This is because early software failure prediction can improve software quality, reliability and efficiency, and reduce software cost. However, developing robust defect prediction models is a challenging task and many techniques have been proposed in the literature. In this paper, a software defect prediction model based on novel hybrid genetics software defect prediction (NHGSDP) is proposed. The supervised NHGSDP algorithm has been used to predict future software failures based on historical data. The evaluation process shows that the NHGSDP algorithm can be used effectively with high accuracy. The collected results show that the NHGSDP method has better performance.

Keywords: Rule mining, Defect, Genetic, Software metrics, Prediction.

Introduction

Software testing can be defined as "the process of analyzing a software item to detect differences between existing and required conditions and to evaluate the characteristics of the software item" (Nalini, C and T, Murali Krishna, 2020). The purpose of this test is "to provide information about the quality of the test items in the non-functional and functional requirements" (Perera and Anjana, 2020). On the other hand, software quality can be defined as "the degree to which software has a desired combination of attributes." Human error leads to product defects, which may behave unexpectedly or produce unexpected or incorrect results (Harki *et al.,* 2020). The fundamental principle of testing is to provide information about software quality, usability failures, and defect discovery before completion (Perera and Anjana, 2020). These tests also contribute to a better understanding of systems, especially complex systems, making them an integral part of software engineering (Kaur *et al.,* 2020).

Clustering is an unsupervised data mining technique where the class labels are unknown. In grouping methods, data items are grouped based on their similarity to other data items. Clustering is the process of grouping data so that similar data items are placed in the same cluster. Fuzzy clustering is a clustering algorithm for predicting software defects. In this technique, defective data items are moved between groups until the most suitable group is found. This method is used to predict failures in program modules. Association mining is a data mining method for identifying frequently occurring sets of data items. It is a method for finding correlations between elements in a dataset (Supriya, M., and A. J. Deepa, 2020).

For a long time, testing has emphasized the failure or defect of the system under different conditions. The main problem is that test managers slow down the development process and lead to limited final testing before the software is complete. Another problem is the lack of testing, and the testing environment and human testing or testing tools are very dependent. Test environments are generally not dependent on precise configuration during software development. The absence of such a problem in testing is that software testing teams, instead of looking at the functionality of the system, have an attitude that limits the software bugs they find (Sikka, Geeta and Renu Dhir,

PG & Research Department of Computer Science, Adaikalamatha College, Affiliated to Bharathidasan University, Vallam, Thanjavur, Tamilnadu, India.

***Corresponding Author:** Rajeev P. R., PG & Research Department of Computer Science, Adaikalamatha College, Affiliated to Bharathidasan University, Vallam, Thanjavur, Tamilnadu, India., E-Mail: prrajeev1904@gmail.com

How to cite this article: Rajeev, P. R., Aravinthan, K. (2024). A novel approach for metrics-based software defect prediction using genetic algorithm. The Scientific Temper, **15**(3):2709-2718.

Doi: 10.58414/SCIENTIFICTEMPER.2024.15.3.39

Source of support: Nil

Conflict of interest: None.

Received: 12/07/2024 **Accepted:** 14/08/2024 **Published** : 16/09/2024

2020). Furthermore, the testes ignore the lessons learned from the trials because the team leader failed to document errors and solutions. This caused the error to be repeated in previous projects.

Objective

The objectives of this research are detailed below:

- Create new datasets based on metrics extracted from source code.
- Create a rule to predict the best fault detection.
- Create novel algorithms to predict software defects.
- Use efficient classification algorithms to better predict software defects.
- Use effective indicators and methods to evaluate results.
- Suggest a low-cost software development process.
- Reduce the time and effort of fault tracking.

Scope

- Finding defects to help improve the level of quality.
- Reducing the risk of failures occurring during operation and gain confidence about the level of quality.
- Improve management decisions by providing information for decision making.
- Prevent defects by gaining insight into system behavior to identify processes in the organization that need improvement.
- Implement suggested techniques in software systems for the classification and automatic detection of software defects.

Literature Review

There are many studies on using machine learning techniques to predict software errors. For example, the study in (Perera and Anjana, 2020) proposes a linear autoregressive (AR) method to predict defective modules. This study predicts future software failures based on historical data of accumulated software failures. The study also evaluated and compared the AR model and the power known model (POWM) using root mean square error (RMSE) measurements. Furthermore, the study used three datasets for evaluation and the results are promising. (Harki *et al.,* 2020) studied the applicability of various ML methods to predict failures.

Add to their study the most important previous research on each ML technique and current trends in using machine learning to predict software errors. This research can serve as a basis or step in preparing for future work in predicting software errors (Harki *et al.,* 2020).

A good systematic evaluation of software error prediction techniques using machine learning (ML) by Supriya, M., and A, J, Deepa., in 2020. This document reviews all research from 2000 to 2022, discusses ML techniques for software error prediction models, and evaluates their performance. Different ML techniques summarizes the pros and cons of ML techniques compared to statistical and ML techniques. This document provides a benchmark that allows a general and useful comparison between different error prediction methods. The study presents a complete comparison of known error prediction methods and introduces a new method to evaluate its performance with good comparisons with other methods (Sikka *et al.,* 2020).

Methodology

The proposed method predicts software defects using predetermined patterns and analyzes by building a new database of software metrics and ends with software defects (Figure 1). This chapter details the methods, datasets, and techniques used to identify software defects.

Noise Reduction

The first step in preprocessing is preliminary filtering. This step removes some of the existing noise in the iterations to reduce its impact on subsequent steps. More specifically, noisy instances identified with high confidence are removed in this step. This filtering is followed by silent filtering. The new filter contains partially clean data from the previous step and is applied to the training samples to produce a clean and noisy set. The last step is to remove the noise from the noise score (Pandey *et al.,* 2020).

Figure 2 illustrates the preprocessing steps involved in enhancing the quality of software data. Initially, preliminary filtering eliminates noisy instances, followed by silent filtering to remove partially clean data and apply the filter to training samples, mitigating noise impact. Feature reduction identifies and removes attributes with constant values, reducing redundancy in datasets. Missing value reduction employs the RandomForest algorithm to interpolate missing data accurately. Redundant reduction identifies redundant features and performs feature sub-selection to eliminate irrelevant attributes. Each step contributes to refining the

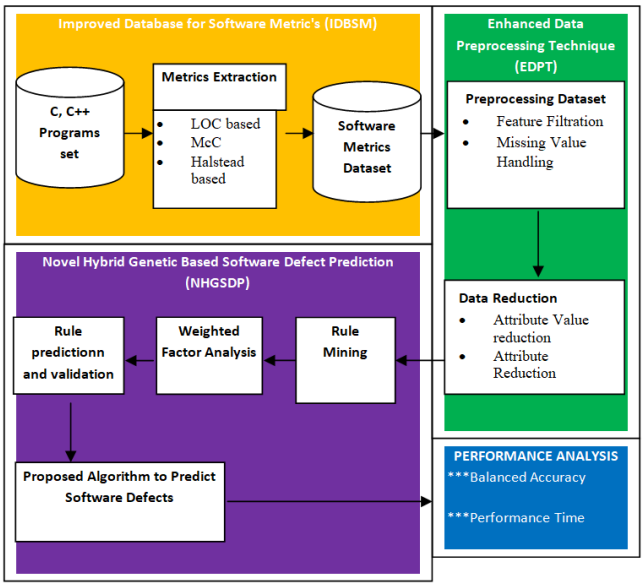

Figure 1: Software defect prediction Architecture

dataset, ensuring its suitability for subsequent analysis and classification tasks, thereby enhancing the effectiveness of software defect prediction models through improved data quality and relevance.

Feature Reduction

A property that has a constant/fixed value in all cases is easily identifiable because it becomes zero. These attributes do not have any information to distinguish the modules and are, at best, a waste of classification resources. This work reduces redundant attributes in datasets/metrics databases. Some attributes are repeated and reduced again. Both attributes of each instance have the same value, resulting in over-representation of a single attribute (Sharma *et al.,* 2020)

Missing Value Reduction

With the increasing amount of data and the emergence of data, the problem of missing data is still common in statistical problems and requires specific methods. Given our approach to reducing such large amounts of data, this paper proposes the application of the random forest algorithm (Sharma *et al.,* 2020), an interpolation algorithm for missing data in mixed datasets. The purpose of the algorithm is to accurately predict individual loss values rather than draw distributions randomly so that estimates can bias the results. The parameters of the statistical model are mimicked.

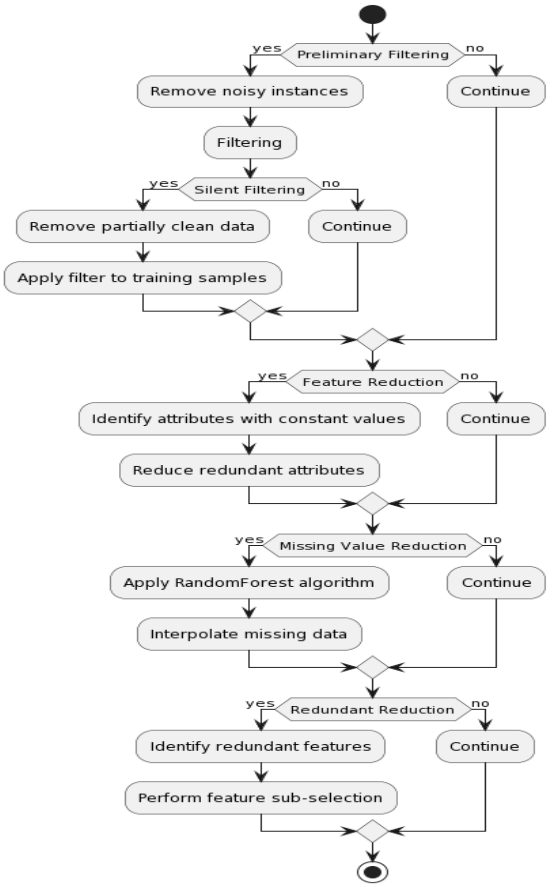

Figure 2: Preprocessing steps

Redundant Reduction

In this approach, the number of studies using eigensubselection techniques and widely used eigensubselection techniques are identified. It is important to perform feature sub-selection on the input data before feeding it to the learning algorithm, as the data may contain redundant and irrelevant features. Of the 22 features selected for this system mapping, 16 were studied using the feature subselection method, i.e., exactly 50% of the studies used the feature subselection method (Liang *et al.,* 2020).pr

Rule Mining

Rule mining is a classification method aimed at accurately measuring and predicting defects. Before creating a failure prediction model, determine the learning scenarios for building the model. The dataset is divided into two parts, and the identifiers are learned on 60% of the data in the dataset. Knowledge is implicit in a set of rules. Rule mining consists of two nested loops. The outer loop selects values from classes, while the inner loop creates rules that apply to classes and returns the best combination of classes (Kwak *et al.,* 2020). Define simple rules for each metric based on suggested intervals. These rules fire if a module's metrics are not within the specified interval (meaning the module was manually verified). It shows 12 base rules with their corresponding flags and 2 derived rules. The first derivative rule, rule 13, defines the separation of the 12 basic rules. If you trigger some of the basic rules, that's Rule 13 triggering. Figure 3 portrays the architecture of the rule mining process, delineating the sequential steps involved in extracting rules from a dataset (Figure 4). Initially, the dataset undergoes division into two parts, with 60% allocated for learning scenarios and the remaining 40% for evaluation. The process then enters nested loops, with the outer loop selecting values from classes, while the inner loop iterates through these values to create rules specific to each class. These rules are formulated based on the dataset's characteristics and are designed to accurately predict software defects. Once the loops conclude, a comprehensive set of rules is generated, encapsulating the knowledge extracted from the dataset. This visual representation offers insight into the systematic approach employed in rule mining, highlighting the iterative nature of the process and the structured methodology behind rule creation from the dataset.

Clustering Techniques

Clustering techniques group the training data such that the similarity within a group is greater than the similarity between all groups. Clustering techniques use distance and similarity measures to find similarities between two objects in order to group them. In this work, he studies K-means technique and c-means for fuzzy clustering. K-means divides the data into k groups and iteratively randomly selects centroids. The value of k affects the performance of the

Table 1: Software metrics and its definition

Attribute name	*Description*
Loc	McCabe's line count of code
v(g)	McCabe «cyclomatic complexity»
ev(g)	McCabe «essential complexity»
iv(g)	McCabe «design complexity»
n	Halstead total operators + operands
v	Halstead «volume»
l	Halstead «program length»
D	Halstead «difficulty
i	Halstead «intelligence»
e	Halstead «effort»
b	Halstead
T	Halstead's time estimator
lOCode	Halstead's line count
lOComment	Halstead's count of lines of comments
lOBlank	Halstead's count of blank lines
lOCodeAnd Comment	Numeric
uniq_Op	unique operators
uniq_Opnd	unique operands
total_Op	total operators
total_Opnd	total operands
Branch_count	Total flow graphs

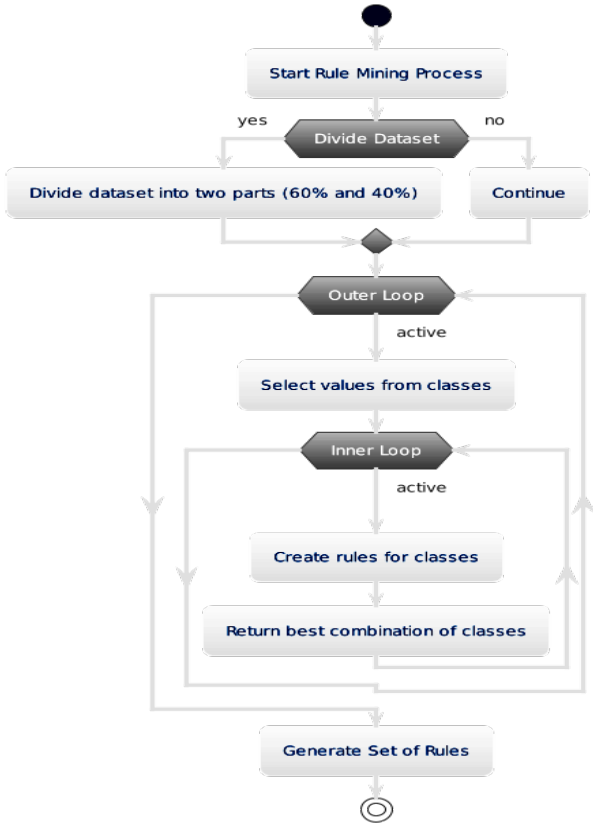

Figure 3: Architecture of the rule mining process

technique (Pandit *et al.,* 2020). We tried four different values of k, (i.e., 2, 3, 4, and 5) and found that k = 2 tended to work best. We also investigated the fuzzy C-means technique (Harzevili *et al.,* 2021) (FCM), which automatically divides a dataset into an optimal (approximate) number of groups (Ksiazek *et al.,* 2021).

Experimental Results

Improved Database of Software Metrics (IDBSM)

The software metrics dataset proposed by IDBSM considers several real-time software metrics in its collection. The collected metrics are then passed through a series of steps in which LOC, McCabes, and Halstead techniques are applied to create a database. The metrics considered are based on completed software projects to support the benchmarking business. NASA IV&V Metrics Data Program - Software datasets provided by the Metrics data repository (MDP) are used in most experiments in software engineering and related fields. Data warehouses contain software metrics as attributes of datasets and also indicate whether a particular dataset is flawed or not. All data contained in the repository is collected and verified by the metrics data program. All software flags are listed in Table 1.

IDBSM extracted a total of 22 attributes as it contained 5 different lines of code, 3 McCabe metrics, 4 Halstead base

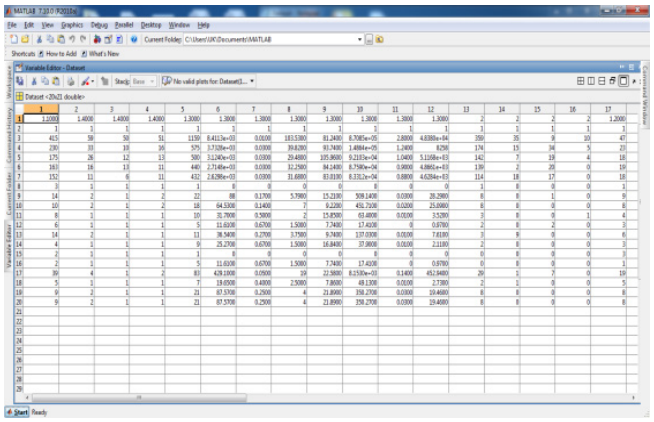

Figure 4: Dataset metric extraction

metrics, 8 Halstead derived metrics, 1 branch count, and 1 output field (Figure 4).

Enhanced Data Preprocessing Technique (EDPT)

The IDBSM database was used as input for EDPT. EDPT removes all files not included in the metrics extraction, i.e., readme files, test scripts and help files. Additionally, 0.2% of "commit ID - filename" records (9 out of 4623 unique tuples) related to source code files were also removed (Figure 5). These records are outliers, and in extreme cases, source files are moved or deleted. More specifically, version

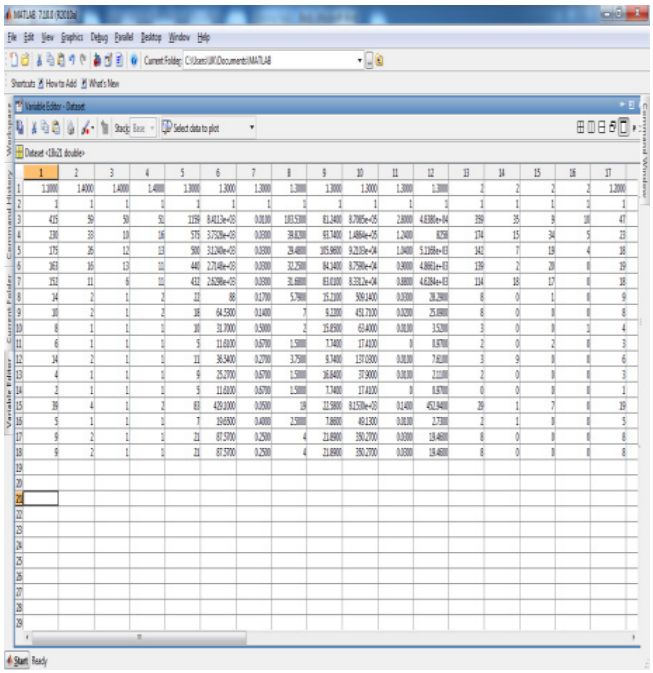

Figure 5: Record reduction

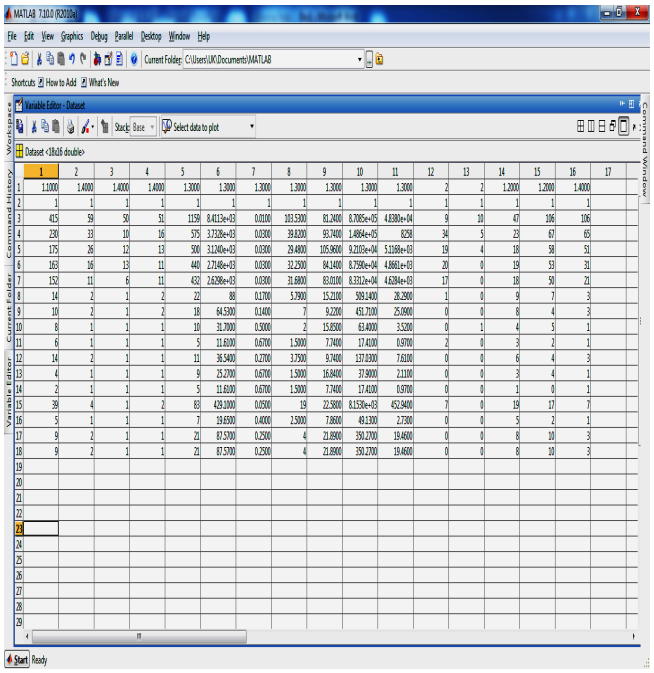

Figure 6: Attribute reduction

control systems recognize directory changes/refactorings as complete deletions of files by default. Every time a file moves up or down one or more levels in the directory structure, an unusual number of lines are added or removed. In some cases, including large files, more than 10,000 lines were added or removed from the commit. Earlier cleaning results in more accurate model creation. Figure 6 shows a simplified EDPT log of the dataset.

Novel Hybrid Genetic Based Software Defect Prediction (NHGSDP)

In the current work, Software Metric 21 is McCabe and Halstead's metric, measured using objective metrics. Using Matlab tools, the dataset was applied to the Naive Bayesian classifier and the proposed algorithm. This dataset is based on a combination of structural and object-oriented. Most of the source code is written in C and C++. The study compared mean precision (values from 0 to 1), true positive rate, false positive rate, sensitivity, and specificity. Accuracy is calculated from the number of correctly classified instances. Based on the results of these analyses, the method is applicable to large datasets. The table below uses different classifiers to accurately classify and classify instances using the total number of instances in the dataset. It also highlights based on sensitivity and specificity values to provide the best classifier. Table 2 lists the weight analysis of the NHGSDP in the extracted features, while Table 3 lists the weighting factors table for the NHGSDP.

Table 2 presents the rules and their associated weights used in the novel hybrid genetic-based software defect prediction (NHGSDP) model. Each rule, identified by a unique ID, corresponds to a specific software metric condition such as lines of code (LOC), McCabe's complexity (V(G), ev(G), iv(G)), Halstead's metrics (V, l, D, i, e, b, T, lOCode, lOComment, lOBlank, lOCodeAndComment, uniq_Op, uniq_Opnd, total_Op, total_Opnd), and branch count. The weight assigned to each rule determines its influence in predicting software defects, with higher weights indicating greater significance in defect prediction. For instance, rules related to higher values of metrics like LOC, V(G), V, e, t, and branch count carry heavier weights (4), signifying their stronger predictive power, while lower weights (0) are assigned to rules representing lower metric values, implying less impact on defect prediction. This Table 2 provides a

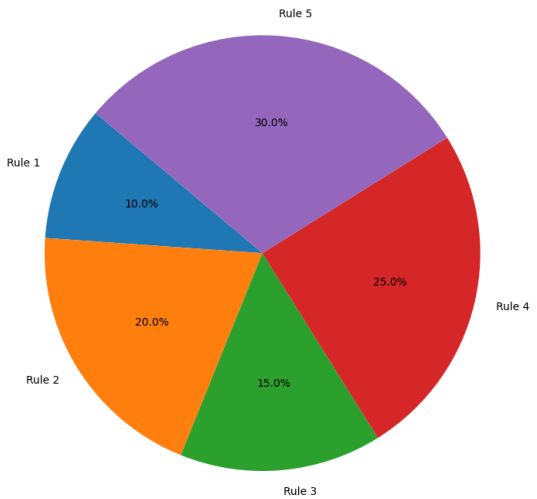

Figure 7: Weight distribution of rules in NHGSDP model

Table 2: Weighted factor table

Rule ID	*Rule*	*Weight*
1	If LOC > 150	4
	Else if LOC > 101 && LOC <= 150	3
	Else if LOC > 51 && LOC <= 100	2
	Else if LOC > 25 && LOC <= 50	1
	Else if LOC <= 25	0
2	If V(G) > 10	4
	Else if V(G) > 7 && V(G) <= 10	3
	Else if V(G) > 5 && V(G) <= 7	2
	Else if V(G) > 2 && V(G) <= 5	1
	Else if V(G) <= 2	0
3	If ev(G) > 5	2
	Else if ev(G) > 2 && ev(G) <= 5	1
	Else if ev(G) <= 2	0
4	If iv(G) > 10	4
	Else if iv(G) > 7 && iv(G) <= 10	3
	Else if iv(G) > 5 && iv(G) <= 7	2
	Else if iv(G) > 2 && iv(G) <= 5	1
	Else if iv(G) <= 2	0
5	If V > 350	2
	Else if V > 100 && V <= 350	1
	Else if V <= 100	0
6	If l > 0.1	1
7	If d > 10	2
	Else if d > 5 && d <= 10	1
	Else if d <= 5	0
8	If i > 50	2
	Else if i > 20 && i <= 50	1
	Else if i <= 20	0
9	If e > 5000	4
	Else if e > 3000 && e <= 5000	3
	Else if e <= 500	0
10	If t > 500	4
	Else if t > 300 && t <= 500	3
	Else if t > 150 && t <= 300	2
	Else if t > 50 && t <= 150	1
	Else if t <= 50	0
11	If IOBlank > 50	1
12	If Uniq_Opr > 15	1
13	If Uniq_Oprnd > 35	1
14	If Branch Count > 35	4
	Else if Branch Count > 25 && Branch Count <= 35	3
	Else if Branch Count > 15 && Branch Count <= 25	2
	Else if Branch Count > 8 && Branch Count <= 15	1
	Else if Branch Count <= 8	0

structured representation of the rules and their weights, offering insights into the prioritization and significance of different software metrics in defect prediction within the NHGSDP model.

Figure 7 illustrates the proportional distribution of weights assigned to different rules within the novel hybrid genetic-based software defect prediction (NHGSDP) model. Each segment of the pie represents a specific rule, while the size of the segment corresponds to the weight assigned to that rule. The chart provides a clear visualization of the relative importance of each rule in the defect prediction process. For instance, larger segments indicate rules with higher weights, suggesting their significant impact on the prediction outcome. Conversely, smaller segments represent rules with lower weights, implying their comparatively lesser influence. This visualization aids in understanding the relative contribution of individual rules toward software defect prediction, enabling stakeholders to prioritize efforts and resources accordingly for more effective defect detection and mitigation strategies.

Rule Prediction

Any attribute with a weight > 2.5 receives a prediction factor of 1, otherwise NHGSDP is zero. Table 3 lists the predictions for the NHGSDP rules, while Table 4 lists the difference tables.

According to the established survey and analysis of NASA's MDP data, the difference in mean values is > 0.407, which is described as a "software defect" in NHGSDP. Table 5 lists the prediction table NHGSDP.

Performance Measures

Performance measures of NHGSDP are detailed below, along with NHGSDP classification results in Table 6 and Figures 8, 9.

True Positive = a =4

False Negative = b =0

False Positive = c =13

True Negative = d =1

Accuracy = acc = (a+d)/(a+b+c+d) = (4+1)/(4+0+13+1) = 5/18 = 99.72

probability of detection = pd = recall = d/(b+d) = 1 / (0+1) = 1

probability of false alarm = pf = c/(a+c) = 13/17 = 0.765

precision = prec = d/(c+d) = 1/14 = 0.0714

effort= amount of code selected by detector = (c.LOC + d.LOC)/(Total LOC) = 1174 /1262.1 = 0.9302

Table 6 presents the classification results of the NHGSDP method across various software engineering datasets. Each row corresponds to a specific method, such as CM1, JM1, KC1, and so on, while the columns display the sensitivity, specificity, and accuracy values for each method. Sensitivity reflects the proportion of true positive predictions, specificity indicates the proportion of true negative predictions, and accuracy denotes the overall correctness of the classification. The Table 6 provides a comparative view of the performance

Table 3: Rule prediction

	A1	*A2*	*A3*	*A4*	*A5*	*A6*	*A7*	*A8*	*A9*	*A10*	*A11*	*A12*	*A13*	*A14*	*A15*
R1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R3	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R4	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R5	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R6	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R7	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1
R8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R17	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
R18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	A1	*A2*	*A3*	*A4*	*A5*	*A6*	*A7*	*A8*	*A9*	*A10*	*A11*	*A12*	*A13*	*A14*	*A15*
R1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R3	4	4	2	4	4	0	2	2	4	4	0	1	1	1	4
R4	4	4	2	4	4	0	2	2	4	4	1	0	1	1	4
R5	4	4	2	4	4	0	2	2	4	4	1	0	1	1	4
R6	4	4	2	4	4	0	2	2	4	4	1	0	1	1	4
R7	4	4	2	4	4	0	2	2	4	4	1	0	1	1	3
R8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R9	0	1	0	1	1	1	1	0	1	2	0	0	0	0	0
R10	0	1	0	1	1	1	1	0	1	2	0	0	0	0	0
R11	0	0	0	0	0	1	2	0	1	1	0	0	0	0	0
R12	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
R13	0	1	0	0	1	1	1	0	1	1	0	0	0	0	0
R14	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
R15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R16	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R17	1	2	0	1	4	0	2	1	0	3	0	0	1	1	1
R18	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

of different methods, showing their varying levels of effectiveness in classifying software defects. For instance, methods like MC1 and PC2 demonstrate high sensitivity and specificity, resulting in high accuracy rates, while others like JM1 show lower performance across these metrics. This Table 6 is crucial for evaluating the efficacy of the NHGSDP method and determining its suitability for defect prediction tasks in software engineering.

Figure 8 provides a comprehensive overview of the performance metrics of different methods within the

Table 4: Rule prediction differentiation table

Rule	*Mean difference (Sum(Attr_ PrWeight)/15)*
R1	0
R2	0
R3	0.466667
R4	0.466667
R5	0.466667
R6	0.466667
R7	0.466667
R8	0
R9	0
R10	0
R11	0
R12	0
R13	0
R14	0
R15	0
R16	0
R17	0.133333
R18	0

Table 5: Prediction table

Rule	*Mean difference (Sum(Attr_PrWeight)/15)*	*Prediction result*	*Actual result*
R1	0	No defect	No defect
R2	0	No defect	No defect
R3	0.466667	Defect	Defect
R4	0.466667	Defect	Defect
R5	0.466667	Defect	Defect
R6	0.466667	Defect	Defect
R7	0.466667	Defect	Defect
R8	0	No defect	No defect
R9	0	No defect	No defect
R10	0	No defect	No defect
R11	0	No defect	No defect
R12	0	No defect	No defect
R13	0	No defect	No defect
R14	0	No defect	No defect
R15	0	No defect	No defect
R16	0.133333	No defect	Defect
R17	0	No defect	No defect

NHGSDP dataset. Each method is represented by a pair of bars, with the first bar segment indicating sensitivity and the second segment indicating specificity. The chart reveals significant variations in the performance of the methods,

Table 6: NHGSDP classification results

Methods	*Sensitivity*	*Specificity*	*Accuracy*
CM1	0.483	0.986	89.13
JM1	0.198	0.956	83.04
KC1	0.450	0.983	87.91
KC3	0.412	0.922	84.8
MC1	0.693	1	99.34
MC2	0.591	1	69.23
MW1	0.429	0.978	89.14
PC1	0.51	0.999	89.62
PC2	0	1	99.37
PC3	0.986	0.966	84.02
PC4	0.577	0.928	92.27
PC5	0.491	0.990	97.28

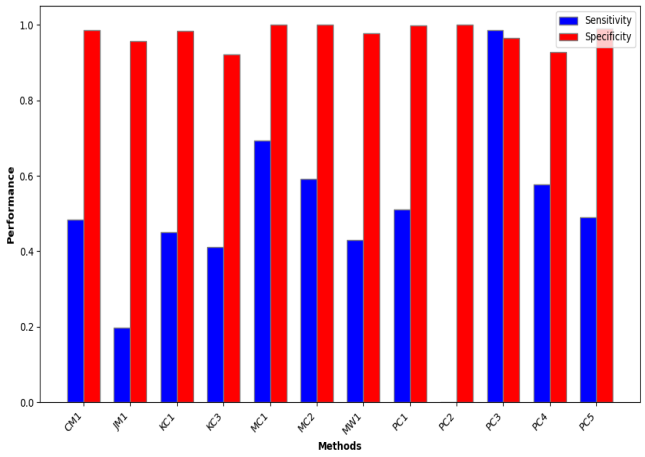

Figure 8: Comparison of sensitivity and specificity across NHGSDP methods

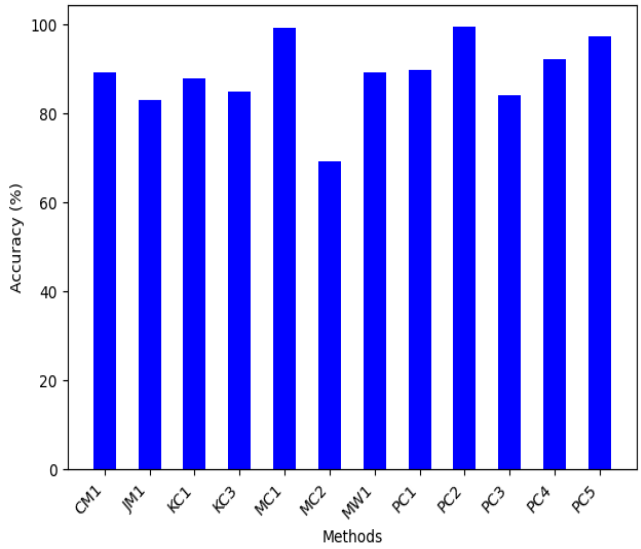

Figure 9: NHGSDP accuracy comparison across different methods

with some achieving high sensitivity but lower specificity and vice versa. For instance, Method MC1 demonstrates the highest sensitivity of all methods, while Method PC2 exhibits perfect specificity. This visualization aids in identifying trade-offs between sensitivity and specificity and allows researchers to assess the overall effectiveness of each method in accurately predicting software defects within the NHGSDP dataset.

Figure 9 provides a visual representation of the accuracy performance across various methods used in novel hybrid genetic based software defect prediction (NHGSDP). Each bar in the chart corresponds to a specific method, and the height of the bar indicates the accuracy achieved by that method. By comparing the heights of the bars, it's evident that some methods outperform others in terms of accuracy. This comparison helps in identifying the most effective methods for software defect prediction within the NHGSDP framework. Additionally, the visual depiction simplifies the understanding of which methods offer higher accuracy rates, aiding in decision-making processes regarding the selection of prediction methods for software development projects.

Conclusion

Software defect prediction is a technique for creating predictive models to predict future software failures based on historical data. Various methods have been proposed using different datasets, different software metrics, and different performance metrics. This paper evaluates the proposed algorithm for use in the problem of software defect prediction. Three machine learning techniques were used, namely IDBSM, EDPT and NHGSDP. The evaluation process is implemented using real test/debug datasets. Experimental results are compiled in terms of precision, sensitivity, and specificity. The results show that the NHGSDP technique is an effective method for predicting future software defects. Furthermore, experimental results show that using the NHGSDP method provides better performance for predictive models than other methods.

References

Aarti, Sikka, G., & Dhir, R. (2020). Novel grey relational feature extraction algorithm for software fault-proneness using BBO (B-GRA). *Arabian Journal for Science and Engineering*, *45*, 2645-2662.

Arun, C., & Lakshmi, C. (2022). Genetic algorithm-based oversampling approach to prune the class imbalance issue in software defect prediction. *Soft Computing*, *26*(23), 12915-12931.

Aziz, S. R., Khan, T. A., & Nadeem, A. (2021). Exclusive use and evaluation of inheritance metrics viability in software fault prediction—an experimental study. *PeerJ Computer Science*, *7*, e563.

Bao, H., & Zhu, H. (2022). Modeling and trajectory tracking model predictive control novel method of AUV based on CFD data. *Sensors*, *22*(11), 4234.

Batool, I., & Khan, T. A. (2022). Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review. *Computers and Electrical Engineering*, *100*, 107886.

Belhocine, A., Shinde, D., & Patil, R. (2021). Thermo-mechanical coupled analysis based design of ventilated brake disc using genetic algorithm and particle swarm optimization. *JMST Advances*, *3*, 41-54.

Boughida, A., Kouahla, M. N., & Lafifi, Y. (2022). A novel approach for facial expression recognition based on Gabor filters and genetic algorithm. *Evolving Systems*, *13*(2), 331-345.

Breda, J. F. D., Vieira, J. C. M., & Oleskovicz, M. (2021). Power quality monitor allocation based on singular value decomposition and genetic algorithm. *Journal of Control, Automation and Electrical Systems*, *32*(1), 175-185.

Castillo, O., & Melin, P. (2021, February). A novel method for a COVID-19 classification of countries based on an intelligent fuzzy fractal approach. In *Healthcare* (Vol. 9, No. 2, p. 196). MDPI.

Conroy-Beam, D. (2021). Couple simulation: A novel approach for evaluating models of human mate choice. *Personality and Social Psychology Review*, *25*(3), 191-228.

Fathima, K., & Vimina, E. R. (2022). Heart disease prediction using deep neural networks: A novel approach. In *Intelligent Sustainable Systems: Proceedings of ICISS 2021* (pp. 725-736). Springer Singapore.

Goyal, S., & Bhatia, P. K. (2021). Software fault prediction using lion optimization algorithm. *International Journal of Information Technology*, *13*, 2185-2190.

Harki, N., Ahmed, A., & Haji, L. (2020). CPU scheduling techniques: A review on novel approaches strategy and performance assessment. *Journal of Applied Science and Technology Trends*, *1*(1), 48-55.

Harzevili, N. S., & Alizadeh, S. H. (2021). Analysis and modeling conditional mutual dependency of metrics in software defect prediction using latent variables. *Neurocomputing*, *460*, 309-330.

Jin, C. (2021). Cross-project software defect prediction based on domain adaptation learning and optimization. *Expert Systems with Applications*, *171*, 114637.

Jin, C. (2021). Software defect prediction model based on distance metric learning. *Soft Computing*, *25*(1), 447-461.

Kaur, A., Jain, S., & Goel, S. (2020). Sandpiper optimization algorithm: a novel approach for solving real-life engineering problems. *Applied Intelligence*, *50*(2), 582-619.

Khan, S. D., Alarabi, L., & Basalamah, S. (2020). Toward smart lockdown: a novel approach for COVID-19 hotspots prediction using a deep hybrid neural network. *Computers*, *9*(4), 99.

Khurana, A., & Verma, O. P. (2020). Novel approach with nature-inspired and ensemble techniques for optimal text classification. *Multimedia Tools and Applications*, *79*(33), 23821-23848.

Książek, W., Gandor, M., & Pławiak, P. (2021). Comparison of various approaches to combine logistic regression with genetic algorithms in survival prediction of hepatocellular carcinoma. *Computers in Biology and Medicine*, *134*, 104431.

Kumar, S., Jakkareddy, P. S., & Balaji, C. (2020). A novel method to detect hot spots and estimate strengths of discrete heat sources using liquid crystal thermography. *International Journal of Thermal Sciences*, *154*, 106377.

Kwak, D. H., & Lee, S. H. (2020). A novel method for estimating monocular depth using cycle gan and segmentation. *Sensors*, *20*(9), 2567.

Liang, X., Teng, F., & Sun, Y. (2020). Multiple group decision making for selecting emergency alternatives: a novel method based on the LDWPA operator and LD-MABAC. *International Journal of Environmental Research and Public Health*, *17*(8), 2945.

Nalini, C., & Krishna, T. M. (2020, July). An efficient software defect prediction model using neuro evalution algorithm based on genetic algorithm. In *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)* (pp. 135-138). IEEE.

Nevendra, M., & Singh, P. (2021). Software defect prediction using deep learning. *Acta Polytechnica Hungarica*, *18*(10), 173-189.

Padhy, N., Panigrahi, R., & Neeraja, K. (2021). Threshold estimation from software metrics by using evolutionary techniques and its proposed algorithms, models. *Evolutionary intelligence*, *14*(2), 315-329.

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications*, *144*, 113085.

Pandit, M., & Gupta, D. (2021). Performance of genetic programming-based software defect prediction models. *International Journal of Performability Engineering*, *17*(9), 787.

PATİL, V., & Ingle, D. R. (2022). A novel approach for ABO blood group prediction using fingerprint through optimized convolutional neural network. *International Journal of Intelligent Systems and Applications in Engineering*, *10*(1), 60-68.

Perera, A. (2020, December). Using defect prediction to improve the bug detection capability of search-based software testing. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (pp. 1170-1174).

Sáez, J. A., & Corchado, E. (2022). ANCES: A novel method to repair attribute noise in classification problems. *Pattern Recognition*, *121*, 108198.

Shahid, A. H., & Singh, M. P. (2020). A novel approach for coronary artery disease diagnosis using hybrid particle swarm optimization based emotional neural network. *Biocybernetics and Biomedical Engineering*, *40*(4), 1568-1585.

Sharma, D., & Chandra, P. (2020). Towards recent developments in the methods, metrics and datasets of software fault prediction. *International Journal of Computational Systems Engineering*, *6*(1), 14-45.

Shaukat, K., Luo, S., & Varadharajan, V. (2022). A novel method for improving the robustness of deep learning-based malware detectors against adversarial attacks. *Engineering Applications of Artificial Intelligence*, *116*, 105461.

Singh, M. R. O., & Thankachan, B. (2021, February). A Detailed Survey on Machine Intelligence Based Frameworks for Software Defect Prediction. In *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)* (pp. 360-365). IEEE.

Supriya, M., & Deepa, A. J. (2020). A novel approach for breast cancer prediction using optimized ANN classifier based on big data environment. *Health care management science*, *23*(3), 414-426.

Tameswar, K., Suddul, G., & Dookhitram, K. (2021). Enhancing deep learning capabilities with genetic algorithm for detecting software defects. In *Progress in Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2020* (pp. 211-220). Springer Singapore.

Tong, H., & Zhu, J. (2022). A novel method for customer-oriented scheduling with available manufacturing time windows in cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, *75*, 102303.

Yedida, R., & Menzies, T. (2021). On the value of oversampling for deep learning in software defect prediction. *IEEE Transactions on Software Engineering*, *48*(8), 3103-3116.