International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V73I6P127

Volume 73 Issue 6, 318-335, June 2025

Original Article

Optimized Horizontal and Vertical Dimension Selection using Hybrid Sampling and Quadratic Discriminant Analysis for Predicting Software Faults

Yuvaraj K[1], Balaji N V[2]

[1,2]*Department of Computer Science, Karpagam Academy of Higher Education, Coimbatore, Tamil Nadu, India.*

[1]*Corresponding Author : kyuvarajj@gmail.com*

Received: 14 March 2025 Revised: 12 May 2025 Accepted: 07 June 2025 Published: 28 June 2025

Abstract - *Software fault prediction is significant research intended to ascertain the faults in the software modules by analysing their various parameters. It aims to ensure maximum quality with minimum time, effort, cost, and usage of testing resources for the underlying software. Like any application, the quality of the data prominently stimulates the prediction result of the software fault. Intrinsically, several challenges, such as class imbalance, irrelevant and redundant attributes, and instance noise, exist in the software defect datasets. This irrelevant input slows the underlying prediction model's performance and produces erroneous prediction results. A data preprocessing methodology has been presented to address this problem by properly choosing the vertical and horizontal dimensions to ensure the quality of the input data. To handle data imbalance in the horizontal dimensions, hybrid sampling that uses SMOTE for oversampling and random under-sampling is applied to the data. It also uses the edited k nearest neighbour rule to remove noises. On the other hand, significant attributes from the vertical dimensions of the dataset are identified by applying the quadratic discriminant analysis. Several datasets have been used in the experimental study to assess the suggested preprocessing model's performance. The findings show that the suggested model performs better as it maintains the quality of the pre-processed dataset. The comparative analysis ensures that the suggested model overcomes the difficulties and performs well enough to forecast software module defects with improved quality up to 2.6% to 5.2% of AuC values.*

Keywords - *Class imbalance, Edited k-nearest neighbour rule, Quadratic discriminant analysis, Random sampling, Software defects, Software fault prediction.*

1. Introduction

In today's world, software has evolved as the strongest medium for automated systems and is truly ruling the entire world. Owing to technological growth and e-business, software has become the most significant entity for individuals and businesses [1]. In software development, identifying faults in the software at an early stage is one of the most difficult tasks. Classifying software errors is a branch of study in software quality assurance that focuses on testing, code inspection, and identifying faulty modules.

As the size of software continues to increase rapidly, the probability of faults associated with it is also growing substantially. Syntactic, semantic, service, communication, and exception failures within software modules are software defects that can lead to code complexity, reduced human understanding, and other issues [2]. These faults result in system failure and reduced quality and compromise software reliability, affecting organizational goodwill and causing financial loss. Because software development progresses through various lifecycle phases, errors in one phase can propagate to others. Detecting such errors in later stages requires reworking the entire process, increasing costs and manpower [3]. Consequently, increasing interest has been in predicting software flaws early in the development cycle [4].

Once faulty modules are identified early, it becomes relatively easier to build dependable and high-quality software. To improve software quality, several statistical models, machine learning algorithms, and software computing approaches are commonly used to forecast software defects using data from previously reported software defects [5]. Classification models are widely used to separate faulty modules from non-faulty modules. However, these models often face limitations in effectively predicting fault-prone software modules because of several persistent challenges in software fault prediction [4, 6]. One major concern in predicting software faults is the class imbalance problem because of the unequal distribution of faulty and non-faulty instances. This often leads to biased classification results [7, 8]. Another significant issue is selecting appropriate software metrics for inclusion in the fault-detection model.

A software metric is defined as a measurable characteristic or indicator of the software that helps evaluate its quality. These metrics are widely used to analyze software quality, assess execution time, detect defects during development, evaluate effectiveness, and control execution [9]. They can be broadly categorized into Halstead metrics, object-oriented metrics, code complexity metrics, size-based metrics using LOC, quality-based metrics, and others. The effectiveness of fault-prediction models is highly dependent on the relevance and selection of these metrics [10].

Although various studies have independently addressed the class imbalance issue or software metric selection challenge, few approaches have addressed both problems in a unified framework. The lack of an integrated data preprocessing methodology highlights a significant research gap in software fault prediction. Moreover, existing models often fail to account for the combined effect of noise in the data and irrelevant attributes, which further affects the performance and accuracy of classification models. Addressing the horizontal and vertical aspects of data preprocessing in a cohesive manner remains an underexplored area.

Despite the importance of class imbalance handling and software metric selection, limited work in the literature addresses these two challenges [11, 12]. This gap motivates the present study, which proposes a data preprocessing model to enhance the dataset by optimizing horizontal and vertical dimensions. Horizontal data preprocessing handles class imbalance using hybrid sampling methods-SMOTE oversampling and random undersampling -along with the edited k-nearest neighbour (ENN) for noise removal. Vertical data preprocessing focuses on selecting significant attributes from the full feature set using Quadratic Discriminant Analysis (QDA). Various datasets and classifiers were used in the experimental analysis to evaluate the effectiveness of the proposed preprocessing model. The study also compared the results with several existing models to validate the improvements.

The remainder of this article is structured as follows. A comprehensive review of the proposed study from the literature is presented in Section 2. The proposed model, which has a suitable architecture framework and horizontal and vertical data preprocessing in the following subsections, is discussed in Section 3. Section 4 deals with the experimental setup, dataset used, and performance metrics used in the evaluation. In Section 5, the results of the experiments are meticulously analyzed and compared with those of other currently used models. The paper concludes with the suggested future directions in Section 6.

2. Related Work

Software defect detection is the most significant research field for predicting fault-prone software modules. Several studies have focused on applying machine learning classifiers to discriminate software modules as defective and non-defective [13, 14], in addition to applying other techniques such as clustering [5, 15] and deep learning [16] for effective results. Usually, classifiers are trained using the training set for fault prediction [17].

However, the fault class will be a minority class with a minimum sample size, resulting in an imbalanced dataset. The prediction ability of the underlying model is always jeopardized when dealing with datasets with an imbalance in class. Thus, the superiority of the dataset is imperative to obtain better prediction results. The existence of irrelevant features and noisy instances exacerbates the problem of data imbalance by lowering the precision and generalizability of fault-prediction models. Consequently, both feature selection and instance quality need to be improved. Few researchers have contributed their research on instance reduction and feature selection, specifically suitable for fault predictions.

A two-phase data preprocessing method was proposed to minimize the number of samples of the defect datasets and select the important features. It utilizes symmetrical uncertainty and threshold-based clustering for selecting features that ensure high relevancy and low redundancy, and it applies random sampling to balance the class [11, 18]. The results are limited to the kNN, c4.5 decision tree, and Naive Bayes classifiers. A three-phase model comprising inter-quartile-range based noise removal, SMOTETomek for class balancing and voting-based ensemble feature selection was proposed to have high-grade preprocessing results [19].

Similarly, information gain-based feature selection, SMOTE-based resampling, and iterative noise filtration utilizing classifier fusion for noise elimination were proposed. However, the work lacks proof of its efficiency compared to other models [20]. Although these studies showed that combining several preprocessing methods can improve model performance, they frequently have limitations regarding validation, algorithm support, or benchmarking against other models.

A linear kernel support vector machine (SVM) with recursive feature elimination (RFE) termed SVM-RFE was proposed and evaluated using an SVM classifier to anticipate defects in software datasets. However, the selected features are more numerous, which makes the classification process difficult [21]. A method for selecting defect forecasting models using decision tree logic and fault characteristics has been suggested [22]. An improved regularized linear discriminant analysis was employed to select significant features and was tested using a few DNA microarray gene expression datasets [23].

A hybrid feature selection model using chi-square, information gain, and correlation has been suggested [24].

However, the model lacks an experimental analysis. Many of these models exhibit potential in dimensionality reduction or feature selection accuracy; however, their scalability, lack of cross-domain testing, and insufficient evaluation limit their applicability to real-world software fault datasets. A partitioning filter with an iterative procedure was proposed to remove instances identified as noise [25].

However, the results were still less significant for many datasets. An analysis was conducted for sampling techniques in which random undersampling yielded better results [8]. An efficient dimensionality reduction model that utilizes Fisher linear discriminant analysis (FLDA) was proposed, which produces good results after resampling [26]. A survey of various sampling techniques and their categories was explicitly conducted for big data [27]. Despite these developments, a few methods offer a unified approach combining feature selection, class balance, and noise filtering into a single preprocessing pipeline for software failure prediction.

In addition to fault predictions, this study employed several data preprocessing approaches suggested for different classification models. A self-weighted supervised discriminative feature selection (SSD-FS) approach that utilizes sparsity-inducing regularization was proposed, which yields better results than many other sparse-based feature selections. However, the experiments were limited to kNN and SVM classifiers [28]. Recently, this work was extended by introducing a redundancy matrix-based framework to minimize redundancy [29].

A mean-weighted pattern score using attribute rank-based selection of features was proposed [30] with the idea of using weights for different patterns [31]; however, the model failed to apply the resampling process to balance the classes. A SMOTE-ENN algorithm for balancing the class ratio and CBoost, a cost-sensitive learning framework, was suggested, but it was experimented explicitly with bankruptcy detection [32]. Although hybrid- and ensemble-based models are becoming increasingly popular, their direct application to software failure datasets is frequently unproven or insufficiently evaluated, as these contributions demonstrate.

Generalizability across various classifiers and dataset properties has also not received enough attention. From the literature review, it is clear that only a few methods focus on imbalanced classes and software metric selection. This shows an immediate need for a novel yet effective data preprocessing model better suited for predicting faults from software fault datasets. Thus, this paper proposes a data preprocessing model suitable for effective software fault prediction.

3. Proposed Preprocessing Model

The proposed optimized horizontal and vertical dimension selection model is intended to improve the data quality through a series of data preprocessing steps, such as hybrid Sampling and QDA, specifically for predicting software faults. Software fault datasets contain a set of software faults and normal instances, which is a binary imbalanced dataset, and applying any model to predict the faults in such datasets will result in inaccurate results.

Thus, the model uses sampling strategies to balance the instances in the binary class, including random undersampling and SMOTE oversampling. It also applies the edited k-nearest neighbour approach to remove noise or outliers in dataset instances. This phase completely deals with the instances, which are the horizontal dimension of the dataset, in contradiction to the vertical dimension, that represents the set of attributes in the dataset.

As all the features may not contribute to the classification or prediction accuracy, the significant features that have more discriminant information with respect to the target class are selected using the quadratic discriminant function using the wrapper-based forward subset selection approach. The overall framework of the proposed data preprocessing model with horizontal and vertical dimension optimization approaches is shown in Figure 1. The phases of the proposed model are described in the following subsections.

3.1. Optimizing Horizontal Dimensions

The first step in the proposed data preprocessing model is to optimize the horizontal dimension. It includes hybrid sampling and noise removal by processing the instances in the underlying dataset.

In the proposed model, hybrid sampling was applied to adjust the instances of the underlying imbalanced dataset to create a balanced dataset. The model applies SMOTE oversampling and random undersampling techniques to take advantage of both models, thereby neutralizing its limitations.

Although balanced, the dataset may have outliers, leading to ineffective results. Thus, the proposed model utilizes the edited kNN rules to detect outliers and enhance data quality effectively. The process for maximizing the number of instances for the majority and minority class instances of the defect dataset is shown in Figure 2.

3.1.1. Class Balancing using Hybrid Sampling

Hybrid sampling is prevalent because an imbalanced dataset always provides a biased prediction of majority class instances.

Thus, for effective results, the proposed model utilizes the synthetic minority oversampling technique (SMOTE) algorithm for oversampling [33] and random undersampling to balance the binary class's instance count. A simple illustration of hybrid sampling is shown in Figure 3.

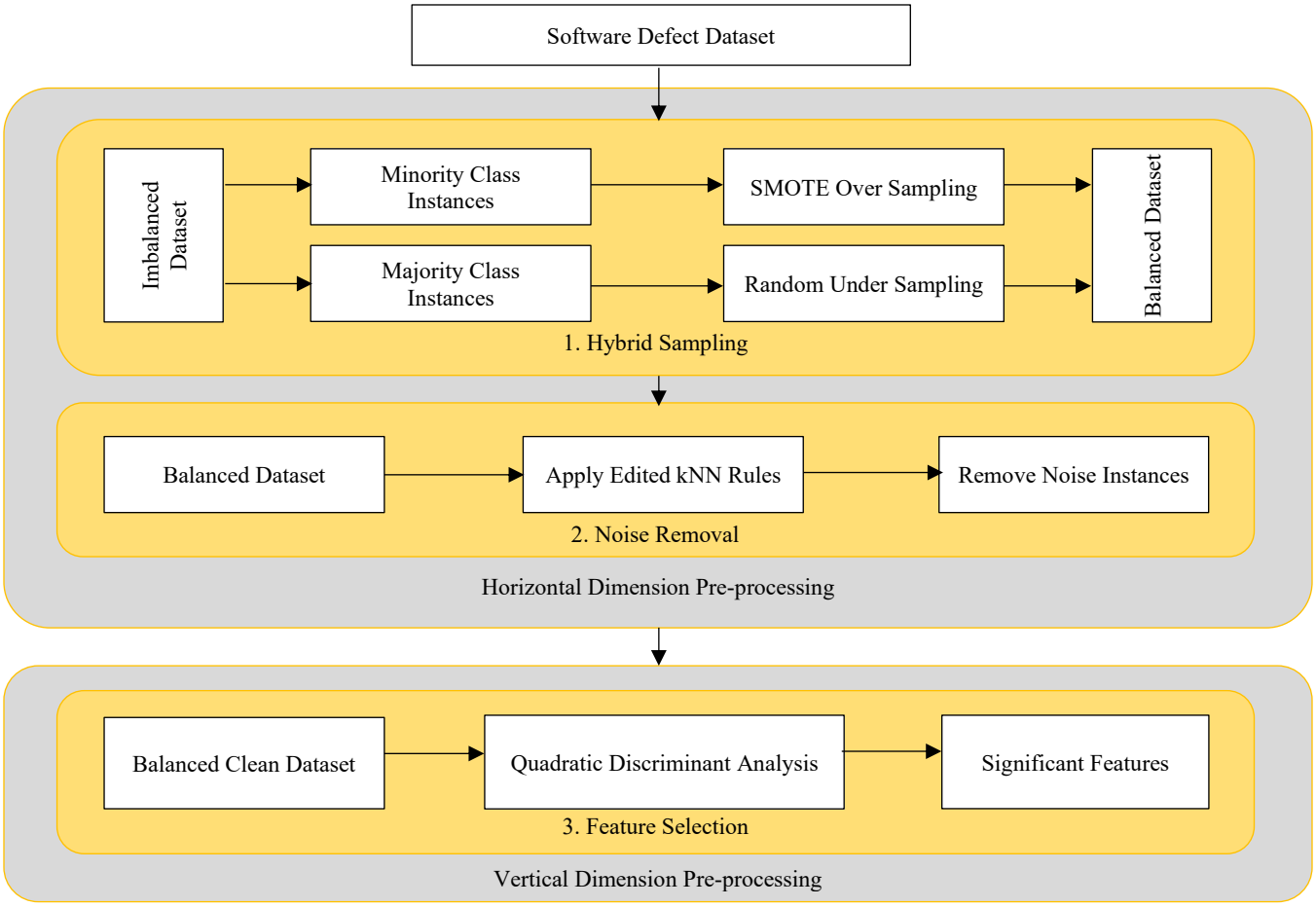

Fig. 1 Detailed structure of the proposed data preprocessing approach

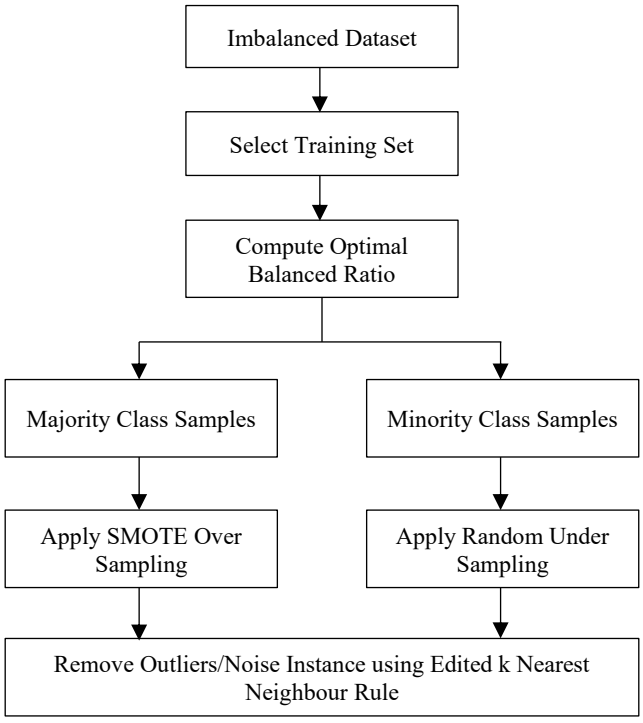

Fig. 2 Workflow for optimizing instances of the defect dataset

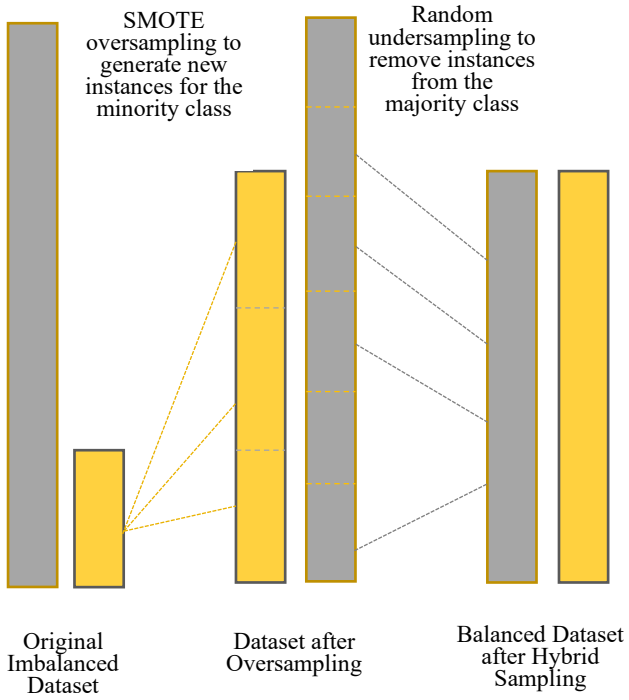

Fig. 3 Illustration of hybrid sampling

The oversampling and undersampling rates must be computed for hybrid sampling based on the training set. Half of the percentage difference between the instances of the dominant and fewer classes was used to calculate the sampling rate. Consider a dataset containing n instances, out of which n_1 samples belong to the majority class and n_2 is the sample count of the minority class instances. Consequently, Equation 1 was used to compute the sample rate.

$$S_{rate} = \frac{n_1 - n_2}{2n} * 100 \quad (1)$$

Thus, in the proposed model, SMOTE increases the instances, and random undersampling reduces the instances by S_{rate}. However, the number of instances (n_m) to be added and reduced can be given as $(n_1 - n_2)/2$ instances.

SMOTE Oversampling

The SMOTE algorithm uses the similarity between minority class instances to create new, nonidentical samples for the minority class. These added instances create a class-balanced dataset, thus avoiding class imbalance and overfitting issues. The k-nearest neighbors for each instance in the minority class were determined. Then, the distance between the feature vector of the instance and that of its N neighbours is computed. The new instance, a synthetic instance, is subsequently included in the feature vector after the distance is multiplied by an arbitrary number ranging from 0 to 1. This is shown in (2).

$$x_{new} = x_1 + [d(x_1, x_n) * rn] \quad (2)$$

Here, x_{new} represents the newly generated instance, x_1 is the feature vector of the instance in the minority group, $d(x_1, x_n)$ is the Euclidean distance between instance x_1 and it is a neighbour x_n, rn is a random number between 0 and 1. As per the suggested model, N is calculated as the ratio of the sample count to be added to the sample count in the minority class n_m/n_2 and k is the smallest number divisible by 5 greater than n_m/n_2.

Random Under-sampling

Random undersampling was utilized to decrease the sample count in the majority group. To balance the dataset, samples from the dominant class were randomly chosen. This iterative procedure is performed until the specified distribution is reached. Random sampling always offers better results than other undersampling techniques [9]. Instead of randomly deleting the n_m instances, the proposed model splits the entire set of samples into subsamples and then deletes the instances. Thus, for each k sample, N samples are deleted randomly, where k is a random number, and N can be computed as $(k * n_m)/n_1$. For example, if the sample count to be deleted is 4000 with a total sample size of 8000, and if 10 is selected as the k value, then N is 5. This implies that for every k (=10) instance, N (=5) instances were deleted randomly. The idea is to select instances for deletion in a distributed manner based on subsamples with k instances.

3.1.2. Noise Removal using Edited K Nearest Neighbour

The number of occurrences in the majority and minority groups in the dataset was balanced using hybrid sampling. However, the dataset may contain noise; thus, the noise can be identified and removed using the Edited K Nearest neighbour (ENN) approach for effective prediction results. A simple illustration of the noise removal using the ENN algorithm is shown in Figure 4.

The working principle of the ENN approach is that if the sample contains more neighbours from different classes, it can be considered an outlier and removed. Thus, for each instance x, it identifies the k nearest neighbours, says k=10, then x can be considered an outlier and removed if the neighbour count from other groups is greater than the same group. Thus, the output of the first phase after applying hybrid sampling and noise removal using ENN is a balanced, clean dataset without more variation in the sample count between binary groups. Algorithm 1 provides the pseudocode for the proposed horizontal dimension optimization.

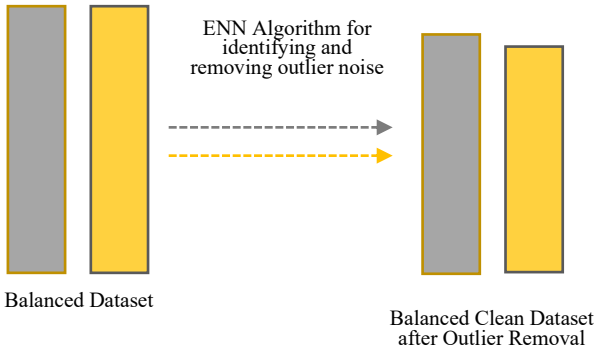

Fig. 4 Illustration of noise removal using ENN algorithm

Algorithm 1: Horizontal Dimension Optimization

Input: Imbalanced dataset D with n instances, n1 minority instances, n2 majority instances
Output: Balanced dataset
Begin hybrid_sampling()
  //Compute the sampling rate
  n_m = (n1-n2)/2 //Number of instances to be added or deleted
  S_{tate} = (n_m/n)*100; //Converting to sampling rate
  //SMOTE Oversampling
  N = n_m/n2//SMOTE percentage
  Identify the suitable k value
  For each instance, x in the majority class do
    N = n_m/n2
    Find the k nearest neighbours from minority classes
    While N ≠ 0 do
      Select a neighbour x_n

```
            Compute d(x, x_n) = |x- x_n| and select r_n ϵ(0,1)
            x_new = x_1 + [d(x_1,x_n )*r_n]
            Append the instance x_new in D
            Decrement N by 1
        End While
    End For
    //Random under-sampling
    Select a value for the variable k
    Compute N = (k*n_m)/n_1
    For each iteration, select k instances from the majority class without replacement. Do
        For i from 1 to N, do
            Remove one instance x randomly and update in D
        End For
    End For
    //ENN for noise removal
    For each instance x in D, do
        Select k nearest neighbours
        Compute n_sc and n_ds as neighbours from the same class and different class
        If n_sc < n_dc, then
            Remove instance x from D
        End If
    End For
End Procedure
```

3.2. Optimizing Vertical Dimensions

In the proposed model, the vertical dimensions represent the features of software fault datasets. Choosing the optimal elements influencing classification accuracy is crucial to detecting the fault-prone software module. The proposed model uses QDA to select important features from the underlying datasets. It uses a wrapper approach in the feature-selection process, in which the best subset with the minimum error is considered the significant feature set. To compute the error rate of the proposed model, the leave-one-out cross-validation error rate was used [34].

3.2.1. Quadratic Discriminant Analysis

QDA is a reproductively supervised feature-reduction model. Here, features that increase the space between classes are selected [35]. The QDA can be evaluated from simple probabilistic models using the class conditional probability of the data with respect to each value of the target variable. Thus, for each training sample x, class prediction k can be made using the Naive Bayes algorithm that maximizes the posterior probability, as given in Equation 3.

$$p(y = k|x) = \frac{p(x|y = k)p(y=k)}{p(x)} \quad (3)$$

However, QDA is modelled as a bivariate Gaussian distribution $p(x|y = k)$ With binary class k =2 and density d representing the number of features. The density ratio can then be computed, as shown in Equation 4.

$$d'(x) = \frac{|\sum_1|^{-1/2}}{|\sum_2|^{-1/2}} exp\left[-\frac{1}{2}(x-\mu_1)^t\sum_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^t\sum_2^{-1}(x-\mu_2)\right] \quad (4)$$

Here, μ_1 and μ_2 are the mean vectors of specific classes class1 and class 2, respectively, which can be computed by averaging the input variable of each specific class. The variables $\sum_1$ and $\sum_2$ specify the covariance matrix of specific classes, which is computed as the covariance of the variables of each specific class [36]. The expression $(x-\mu_1)^t\sum_1^{-1}(x-\mu_1)$ signifies the Mahalanobis distance between instance x and the mean of class [37]. Taking the natural logarithm on both sides of Equation 2 results in a quadratic function, as in Equation 5.

$$\log\big(d'(x)\big) = \frac{1}{2}\log\left(\frac{\sum_1}{\sum_2}\right) - \frac{1}{2}[(x-\mu_1)^t\sum_1^{-1}(x-\mu_1) - (x-\mu_2)^t\sum_2^{-1}(x-\mu_2)] \quad (5)$$

Applying the natural logarithm for the posterior probability given in Equation 1, substituting the values in Equation 5, and solving the equation results in Equation 6, in which $cl_1$and cl_2 represents classes 1 and 2.

$$qda(x) = \begin{cases} \text{x} \in \text{cl}_1 \text{ if } \log\big(d'(x)\big) > log\left[\frac{p_2 c(1|2)}{p_1 c(2|1)}\right] \\ \text{x} \in \text{cl}_2 \text{ if } \log\big(d'(x)\big) \leq log\left[\frac{p_2 c(1|2)}{p_1 c(2|1)}\right] \end{cases} \quad (6)$$

3.2.2. Feature Selection using QDA

Three major categories can be used to classify feature selection techniques: filter, wrapper, and embedding. In the proposed model, significant features are identified using the wrapper approach. Both forward selection and backward elimination can be used in the wrapper strategy. A feature is added to the subset with the least error at each iteration of the forward selection process, which begins with a null set and ends when the error remains constant. In contrast, backward elimination begins with a complete set of features, and at each iteration, the feature with the highest error is identified and eliminated. The iterations ended when the error rate did not change significantly. The proposed model uses the forward elimination approach to identify the features with the highest biased information with respect to the class using QDA. At each iteration, the discrimination function using QDA is computed for the features returned from the forward selection search. Here, each feature subset is evaluated using QDA, and the subset with the minimum error rate is considered the best subset to be selected. The proposed model utilizes the leave-one-out cross-validated error rate offered by [34] because it produces better results than other error rates. The model leaves each instance from the given dataset and applies a QDA discrimination function to the remaining instances. After evaluating the remaining instances, the class value was predicted for the instance on the left. This was performed for

all instances in each group. Equation 7 can then be used to obtain the error rate.

$$loo\_cv\_error = \frac{n_{1m}+n_{2m}}{n1+n2} \quad (7)$$

Here, $n_{1m}$and n_{2m} represents the number of misclassified instances in classes 1 and 2, respectively, and n_1 and n_2 specify the sample count in classes 1 and 2, respectively. The pseudocode for selecting the significant features from the given dataset with the wrapper approach-based forward selection search method using the QDA classifier and leave-one-out-cross validation error rate estimation is given in Algorithm 2.

Algorithm 2: Vertical Dimension Optimization

```
Input: Dataset D with n features
Output: Significant Feature Subset
Begin QDA_feature_selection()
    Fs={ }
    While (variation exists in loo_cv_error), do
        For each feature i, i∉Fs do
            Fs = Fs∪i
            Apply the QDA model with the features in the
set Fs
        Estimate the error loo_cv_error as in Equation (7)
         End For
         Select the feature subset Fs with
min(loo_cv_error)
     End While
     Return the dataset with the best feature subset Fs
End Procedure
```

4. Experimental Analysis

The experimental evaluation and various result analyses conducted by simulating the experiments using the suggested model to demonstrate its efficacy are presented in this section. The experiment was performed on an Intel Core i3 processor with a speed of 1.70GHz and 4GB RAM running a 64-bit Windows Operating System. Software tools such as Weka and Orange Tools were used to analyse several models' correctness statistically, and Python was used to build the suggested preprocessing model.

4.1. Dataset Used

To assess the performance level of the proposed model, a few experiments and analyses of the obtained results were conducted using various real-time datasets collected from software projects, including NASA datasets [38] and the Bug dataset from the Eclipse project [39]. For our analysis, 11 datasets (CM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5) from NASA projects [40] and three datasets (Eclipse 2.0, Eclipse 2.1, and Eclipse 3.0) from eclipse projects were used. In software defence datasets, the features represent some software metrics that help classify the error-prone module.

The software metrics utilized in the NASA datasets include the LOC, Complexity, and Halstead metrics. The complexity measure was proposed in [41], in which an increase in the complexity of a path increases the possibility of fault. It comprises several metrics, including LOCs and cyclomatic, essential, and design metrics. Halstead (1977) [42] takes the readability of the code as a metric in which it is difficult to read the code, indicating a higher possibility of faults that can be divided into the base, derived, and LOC metrics. Attributes with continuous values were discretized for easy processing and data management. The Eclipse dataset contains a set of metrics that include code complexity metrics, syntax tree-based metrics, and abstract syntax tree-based measures. However, the model preprocesses the datasets by removing non-numeric features, thereby utilizing the numeric features and transforming the datasets into binary classes by updating various defect classes as defects. In addition, in any dataset, the attributes with a single value will not provide any information; therefore, they are also removed [11]. The details of the datasets used in this study are given in Table 1. The table presents the dataset's feature count, instance count, and percentage of defect instances.

Table 1. Description of the dataset used

Dataset	#Features	#Instances	% Defects
CM1	37	327	15.0
KC1	21	1162	28.1
KC3	39	194	18.6
MC1	38	1988	2.3
MC2	39	161	32.3
MW1	37	253	10.7
PC1	37	679	9.0
PC2	36	745	2.1
PC3	37	1077	12.4
PC4	37	1287	13.8
PC5	38	1711	27.5
Eclipse 2.0	155	6729	14.5
Eclipse 2.1	155	7888	10.8
Eclipse 3.0	155	10593	14.8

4.2. Evaluation Metrics

The only method to assess the effectiveness of a model is to use evaluation metrics to quantify its performance. This result specifies the quality of the underlying model. Several evaluation metrics are available in the literature for various applications [43]. The most frequently used evaluation metrics in many applications pertaining to different research fields are accuracy, precision, and error rate. This study uses some of the most widely used measures in the defect detection sector.

4.2.1. Classification Accuracy

This is the main assessment statistic that calculates the proportion of correctly classified examples out of all the occurrences. It is most effective for balanced datasets containing equal sample counts in all classes. Higher accuracy values indicate better performance.

4.2.2. Area under the Receiver Operating Characteristic (AUC)

This curve eventually applies the trapezoid rule to estimate the performance by plotting the true positive and false positive rates. It evaluates the distance between target variables. The value of AuC always lies between 0 and 1, representing the worst and best performances of the model, respectively.

4.2.3. Precision

This represents the rate of truly positive predictions that are positive. This can be measured as the ratio of true-positive instances to positive instances. The increase in the values indicates an increase in performance.

4.2.4. Recall

It is often used along with precision, which indicates the positive prediction rate that is successfully predicted. It can be measured as the number of true positives that are correctly identified. The increases in the values indicate an increase in performance.

4.2.5. F-Measure

This is a measure of the test accuracy that can be calculated as the harmonic mean of the precision and recall values.

4.2.6. Root Mean Square Error (RMSE)

This quantifiable metric computes the standard deviation of prediction errors. It was computed as the square root of the mean of the square of all errors.

4.2.7. Mean Absolute Error (MAE)

This evaluates the closeness between the predictions and actual outcomes-generally, the lower the errors, the higher the prediction model's performance. Thus, low values of RMSE and MAE indicate better performance.

5. Results

To evaluate the model's effectiveness, the proposed horizontal data preprocessing (HDP) and vertical data preprocessing (VDP) are individually experimented with various trials using 11 different classifiers for the software fault datasets presented in Table 1.

The classifiers used for the proposed study were Naive Bayes (NB), Multinomial Logistic Regression (MLR), multilayer perceptron (MLP), Support Vector Machine (SVM), AdaBoost (ADB), bagging (BAG), Additive Logistic Regression (ALR), stacking (STK), Logistic Model Tree (LMT) and Random Forest (RF). The classification accuracies of the classifiers after HDP, VDP, and without data preprocessing (WDP) for the classifiers using 14 datasets are presented in Table 2.

Table 2. Classification accuracy of different classifiers using the proposed model

Datasets	Method	Different Classifiers										Avg.
		NB	MLR	MLP	SVM	ADB	BAG	ALR	STK	LMT	RF	
CM1	WDP	79.20	84.10	78.29	87.16	86.54	87.16	84.10	87.16	85.93	85.63	84.53
	HDP	83.57	87.54	81.59	89.78	88.41	89.63	86.74	90.37	87.23	87.24	87.21
	VDP	84.23	86.96	83.17	91.71	89.67	90.12	85.78	89.71	88.11	89.47	87.89
KC1	WDP	72.70	75.32	75.82	74.05	73.80	76.33	74.47	73.46	75.32	76.92	74.82
	HDP	78.93	77.38	77.92	79.14	78.31	80.12	77.82	76.19	79.33	79.18	78.43
	VDP	77.11	79.23	76.72	78.91	77.59	79.97	78.93	77.59	78.45	80.28	78.48
KC3	WDP	78.87	77.84	76.80	81.96	82.47	78.87	81.96	81.44	79.38	81.44	80.10
	HDP	80.19	81.47	80.56	83.92	83.17	80.96	83.19	84.25	83.47	83.11	82.43
	VDP	81.28	83.72	82.96	85.17	84.23	82.77	84.52	85.16	84.44	83.69	83.79
MC1	WDP	91.41	95.82	97.14	97.49	97.02	97.14	97.26	97.49	97.49	97.37	96.56
	HDP	93.56	96.78	98.23	97.93	98.59	98.17	98.23	98.17	98.82	98.87	97.74
	VDP	95.22	97.49	98.71	98.18	97.99	98.29	98.59	98.39	98.98	98.57	98.04
MC2	WDP	72.00	61.60	70.40	68.80	69.60	69.60	66.40	64.80	64.80	70.40	67.84
	HDP	75.29	68.89	74.88	74.23	74.34	75.18	71.08	69.82	69.58	74.21	72.75
	VDP	76.82	70.29	76.18	76.23	73.87	76.31	72.53	70.18	70.24	75.35	73.80
MW1	WDP	81.42	86.96	86.56	89.33	87.75	89.33	90.12	89.33	90.51	88.54	87.98
	HDP	83.29	90.54	88.74	91.69	90.47	91.11	92.48	92.44	93.53	91.15	90.54
	VDP	84.29	91.47	89.52	90.11	91.28	92.47	94.55	92.69	94.78	92.66	91.38
PC1	WDP	88.09	91.49	91.91	91.21	90.92	91.49	91.49	91.35	90.78	92.06	91.08
	HDP	90.12	92.59	93.85	93.48	92.79	92.85	94.59	93.36	92.22	94.18	93.00
	VDP	91.15	92.69	94.43	94.32	92.93	93.18	95.57	93.66	93.78	94.28	93.60
PC2	WDP	90.74	96.64	97.58	97.85	97.85	97.85	97.45	97.85	97.85	97.85	96.95
	HDP	92.55	97.56	97.82	98.96	98.02	98.87	98.36	99.12	98.67	98.82	97.88
	VDP	94.28	97.29	98.11	98.55	98.49	98.82	98.63	98.85	98.29	98.11	97.94

PC3	WDP	80.97	86.58	85.03	86.97	86.84	86.58	86.19	86.97	85.68	87.35	85.92
	HDP	84.96	88.23	87.89	89.37	89.73	88.17	88.56	88.28	87.28	88.49	88.10
	VDP	83.75	88.19	86.87	88.96	89.27	89.85	89.17	88.59	88.11	89.08	88.18
PC4	WDP	85.91	90.23	90.48	89.00	89.12	89.74	90.85	87.89	89.86	90.73	89.38
	HDP	89.21	92.18	93.18	91.58	91.8	91.28	92.35	89.28	92.63	93.12	91.66
	VDP	88.48	92.52	93.13	92.65	92.82	92.64	92.75	89.08	91.11	91.22	91.64
PC5	WDP	73.72	76.03	73.21	74.49	73.08	75.51	74.49	72.69	74.87	76.54	74.46
	HDP	78.59	79.96	76.31	76.63	78.58	79.29	78.2	76.59	78.72	79.28	78.22
	VDP	79.18	79.28	77.29	77.24	77.67	78.63	77.98	76.12	78.12	79.09	78.06
Ecl.2.0	WDP	79.78	82.56	77.69	86.23	85.28	86.98	83.29	86.48	84.72	84.42	83.74
	HDP	80.29	86.36	80.87	88.09	87.49	88.07	85.59	89.87	88.19	86.98	86.18
	VDP	82.69	87.29	82.75	89.69	88.27	89.34	86.69	89.18	88.27	87.08	87.13
Ecl.2.1	WDP	82.18	83.29	79.18	87.28	90.28	87.18	85.19	88.19	85.34	86.59	85.47
	HDP	83.29	85.57	82.19	89.21	91.25	89.09	87.09	90.37	89.49	88.92	87.65
	VDP	84.25	86.25	83.58	89.09	92.25	90.17	87.54	91.52	89.73	89.64	88.40
Ecl.3.0	WDP	83.29	84.49	80.93	88.09	91.48	90.18	87.57	90.18	88.93	89.08	87.42
	HDP	86.94	86.97	83.21	91.48	93.09	92.05	88.49	92.63	91.37	92.82	89.91
	VDP	86.08	86.28	85.74	91.83	93.35	91.18	89.27	92.58	91.19	92.36	89.99

By analyzing the obtained results, the proposed vertical data preprocessing and horizontal data preprocessing provided better results individually regarding classification accuracy than without data preprocessing. The overall accuracy of the classifiers with different datasets without applying any instance reduction or feature selection was 84.73%, whereas those with only HDP and VDP were 87.26% and 87.74%, respectively. Thus, the upsurges in the proposed HDP and VDP accuracy are approximately 3% and 4%, respectively. More specifically, the maximum increase in accuracy can be seen for dataset MC2, with an HDP of 7.24% and VDP of 8.79%. In contrast, the minimum increase in accuracy with HDP and VDP can be seen through dataset PC2 as 0.95% and MC1 as 1.21%, respectively. This minimum variation was due to the higher range of imbalances in the datasets. More precisely, datasets PC2 and MC1 have a very minimal number of defect instances of 2.1% and 2.3%, respectively; thus, resampling the instances in the majority and minority classes may not create much difference. This is illustrated in Figure 5.

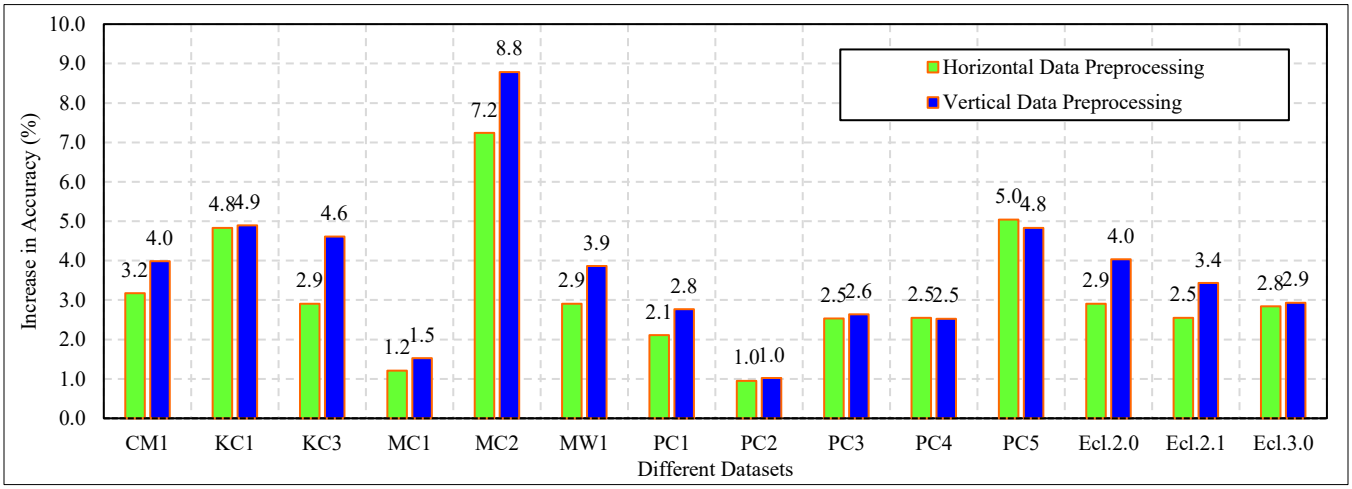

Fig. 5 Increase in accuracy for different datasets

The classification accuracy obtained for various classifiers and datasets is presented as a distribution in Figure 6. Consequently, the average rankings of several classifiers over a range of datasets under the three approaches (Vertical, Horizontal, and Without Data Preprocessing) given in Table 2 are displayed in Figure 7. These rankings are calculated by assigning each classifier in each dataset a rank determined by its performance and then averaging the rankings for each preprocessing method over all datasets. Here, lower ranks indicate a better performance. The robustness of RF, BAG, and SVM is evident from their consistent ranking among the top-performing techniques. This is followed by classifiers such as ADB, ALR, STK, and LMT, which demonstrate moderate performance. Although MLR, MLP, and NB had the

highest rankings, reflecting lower accuracy, the overall classifier performance was marginally improved through horizontal and vertical preprocessing, with vertical preprocessing offering greater consistency. This implies that data preparation enhances the effectiveness of classifiers, particularly in ensemble- and margin-based approaches.

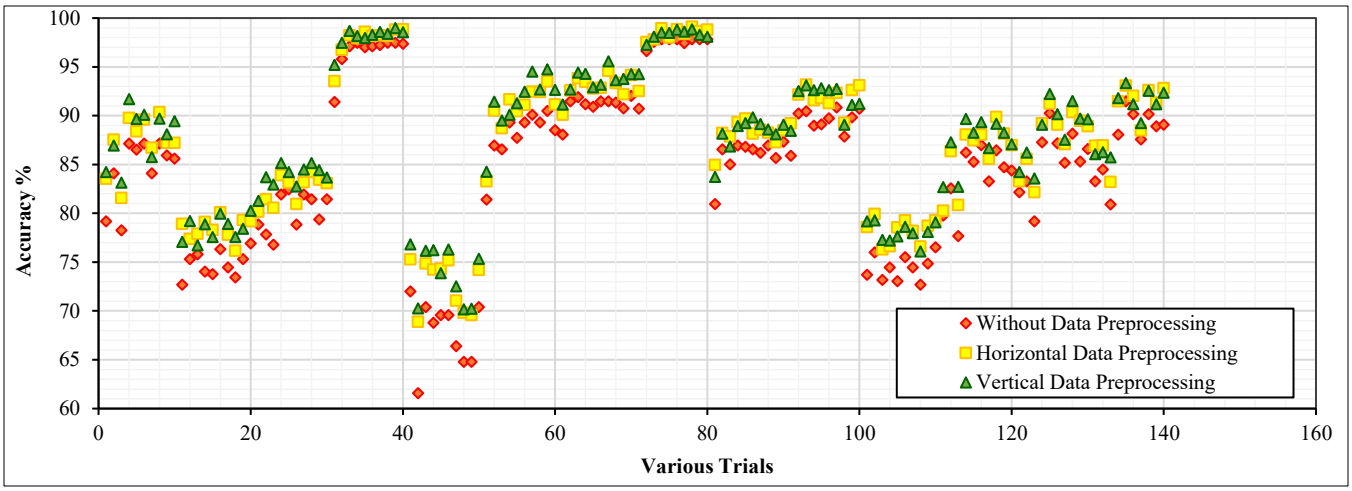

Fig. 6 Accuracy distribution with various models

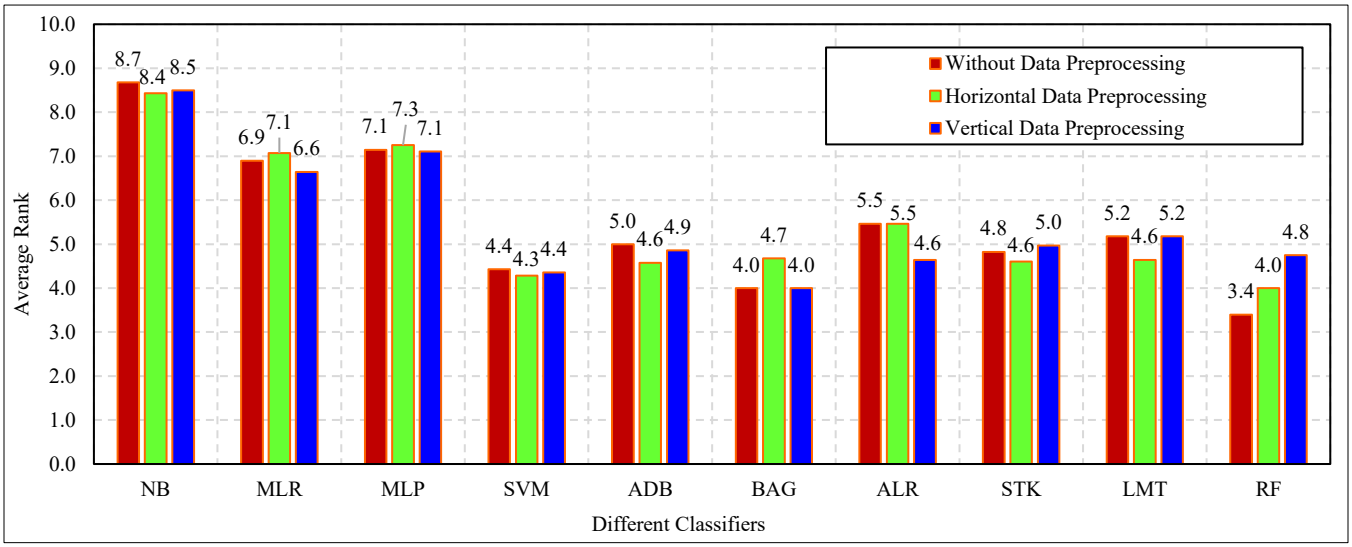

Fig. 7 Average ranking of different preprocessing methods

Specifically, the proposed HDP and VDP performance was also analyzed individually using AUC as the performance metric by applying the 11 classifiers used in the study. The average obtained values for HDP, VDP, and WDP with AUC for the 14 software fault datasets are shown in Figure 8 for ease of interpretation. Vertical and horizontal preprocessing consistently yielded the highest AUC scores across almost all datasets. The performance was the lowest when preprocessing was omitted. This demonstrates that data preparation, particularly through horizontal and vertical methods, substantially improves the classification performance and model reliability, underscoring its importance in enhancing generalization and predictive accuracy.In addition, using the 11 classifiers employed in this study, the performance of the suggested HDP and VDP was further examined separately using a number of other metrics, including precision, recall, f-measure, AuC, RMSE, and MAE. The average values obtained for HDP, VDP, and without any data preprocessing (WDP) with the above metrics for 14 software fault datasets are presented in Table 3. From the obtained results, the proposed HDP had an average value of 80.7% precision, 85.11% recall, 82.75% f-measure, 73.75% AuC, 26.61% RMSE, and 14.14% MAE, whereas the proposed VDP had an average of 80.8% precision, 85.81% recall, 8316% f-measure, 75.44% AuC, 23.39% RMSE, and 13.34% MAE.

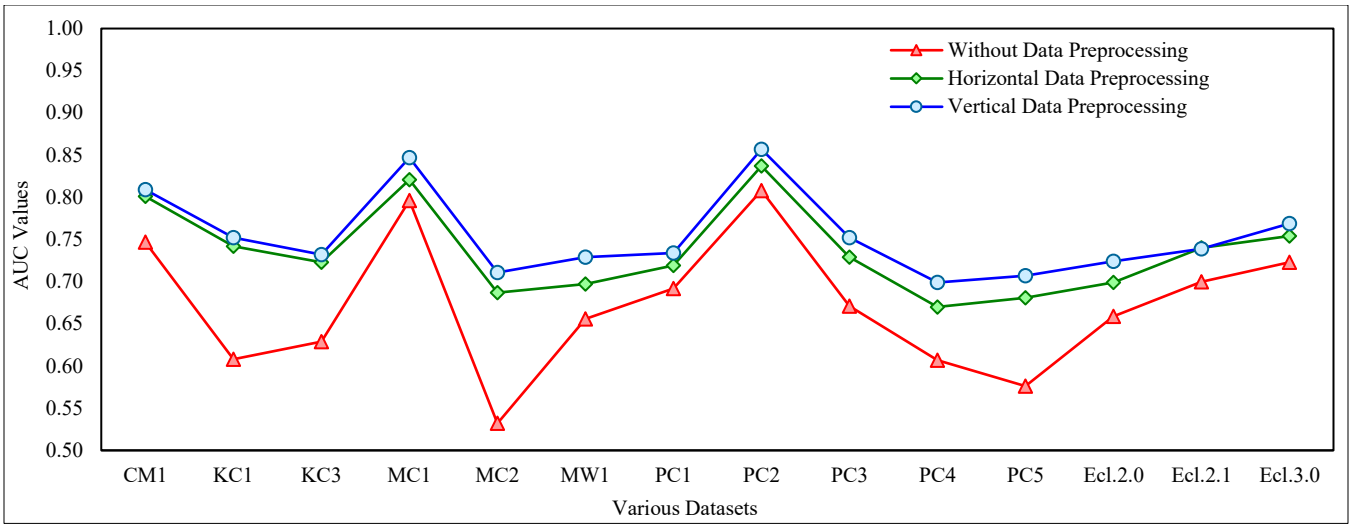

Fig. 8 Average AUC values across different datasets

Table 3. Performance analysis with different metrics

Metrics	Models	Software Defect Datasets													
		CM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5	Ecl.2.0	Ecl.2.1	Ecl.3.0
Precision	WDP	0.817	0.654	0.696	0.944	0.558	0.755	0.803	0.961	0.740	0.715	0.624	0.714	0.753	0.787
	HDP	0.867	0.786	0.785	0.955	0.702	0.783	0.816	0.977	0.797	0.762	0.725	0.743	0.784	0.814
	VDP	0.857	0.775	0.778	0.966	0.712	0.799	0.816	0.979	0.808	0.770	0.739	0.746	0.760	0.807
Recall	WDP	0.794	0.748	0.801	0.966	0.678	0.880	0.911	0.970	0.809	0.894	0.745	0.775	0.769	0.789
	HDP	0.814	0.789	0.836	0.978	0.731	0.897	0.923	0.982	0.848	0.914	0.795	0.795	0.789	0.824
	VDP	0.825	0.791	0.847	0.987	0.744	0.907	0.937	0.989	0.867	0.910	0.814	0.791	0.787	0.817
F-Measure	WDP	0.805	0.698	0.745	0.955	0.613	0.813	0.854	0.965	0.773	0.794	0.679	0.743	0.761	0.788
	HDP	0.840	0.787	0.810	0.966	0.716	0.836	0.866	0.980	0.822	0.831	0.758	0.768	0.786	0.819
	VDP	0.841	0.783	0.811	0.976	0.728	0.850	0.872	0.984	0.836	0.834	0.775	0.768	0.773	0.812
AuC	WDP	0.747	0.608	0.629	0.796	0.532	0.656	0.692	0.808	0.671	0.607	0.576	0.659	0.700	0.723
	HDP	0.801	0.742	0.723	0.821	0.687	0.697	0.719	0.837	0.729	0.670	0.681	0.699	0.740	0.754
	VDP	0.809	0.752	0.732	0.847	0.711	0.729	0.734	0.857	0.752	0.699	0.707	0.724	0.739	0.769
RMSE	WDP	0.340	0.444	0.404	0.191	0.499	0.319	0.279	0.199	0.379	0.291	0.439	0.535	0.478	0.407
	HDP	0.287	0.347	0.248	0.068	0.395	0.314	0.201	0.041	0.287	0.211	0.317	0.397	0.314	0.299
	VDP	0.271	0.317	0.232	0.048	0.371	0.080	0.193	0.032	0.241	0.209	0.342	0.342	0.331	0.265
MAE	WDP	0.231	0.331	0.247	0.065	0.369	0.162	0.131	0.089	0.239	0.145	0.315	0.241	0.251	0.217
	HDP	0.174	0.199	0.152	0.025	0.257	0.091	0.065	0.054	0.140	0.074	0.193	0.193	0.199	0.164
	VDP	0.163	0.197	0.141	0.021	0.244	0.081	0.051	0.027	0.121	0.078	0.174	0.197	0.201	0.171

The increases in the precision, recall, f-measure, and AuC rate for the suggested HDP were 7.37%, 3.35%, 5.45%, and 9.53%, respectively, compared to those without data preprocessing.

Typically, the decreases in the values of error rates, such as RMSE and MAE for HDP, are highly remarkable at 28.40% and 34.72%, respectively. Correspondingly, the hike in precision, recall, f-measure, and AuC for the proposed VDP are appreciable with 7.52%, 4.19%, 5.98%, and 12.30%, respectively, compared with no data preprocessing.

Additionally, the error rate decreases, such as RMSE and MAE for VDP, are notable, with decreases of approximately 37.08% and 38.44%, respectively. Figure 9 shows the average of the data in Table 3 as a bar chart.

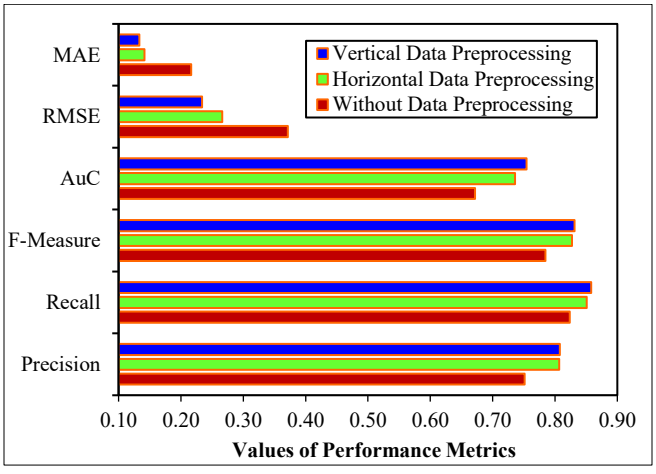

Fig. 9 Analysis of the proposed model with performance metrics

Table 4 presents the classification accuracy of the classifiers across various software defect datasets using the horizontal and vertical data preprocessing techniques proposed in this chapter. The top-performing model varied according to the dataset. SVM achieved the highest accuracy for CM1 (95.59%), whereas RF performed best on KC1 (83.46%). AB outperformed the others on KC3 (87.80%), Eclipse 2.1 (95.57%), and Eclipse 3.0 (97.31%). MLP led to MC1 (99.79%) and PC4 (96.51%), and STK showed superior performance with Eclipse 2.0 (94.28%) and PC2 (99.48%). Similarly, NB performed best with MC2 (81.80%), LMT excelled on MW1 (99.00%), ALR showed the highest accuracy on PC1 (98.56%), BAG led on PC3 (94.07%), and MLR outperformed the others on PC5 (84.12%). These results highlight the dataset-specific effectiveness of the classifiers and underscore the importance of selecting suitable models for different datasets. Different datasets exhibited varying levels of classification complexity, as reflected in the average accuracy values of the classifiers shown in Figure 10. The highest average accuracies were observed for MC1 (98.92%), PC2 (98.71%), and PC1 (96.59%), indicating that these datasets are easier to classify with clearly defined patterns that classifiers can learn from effectively. Similarly, MW1 (95.76%) and PC4 (95.41%) demonstrated strong classification outcomes. In contrast, lower average accuracies in datasets such as KC1 (82.16%), PC5 (82.59%), and MC2 (78.83%) suggest challenges such as noise, class imbalance, and less distinctive features. These results underscore the significant influence of dataset characteristics on the model performance.

Table 4. Accuracy analysis of the proposed model

Dataset	Classifiers									
	NB	MLR	MLP	SVM	AB	BAG	ALR	STK	LMT	RF
CM1	88.11	91.42	87.05	95.59	93.55	94.00	90.62	94.25	91.99	93.35
KC1	82.11	82.41	81.10	82.32	81.49	83.30	82.11	80.77	82.51	83.46
KC3	84.19	83.16	82.13	87.28	87.80	84.19	87.28	86.77	84.71	86.77
MC1	97.12	98.59	99.51	98.98	99.09	98.79	99.09	98.89	99.48	99.37
MC2	81.80	75.27	81.16	81.21	79.32	81.29	77.51	75.16	75.22	80.33
MW1	88.51	95.69	93.74	95.91	95.50	96.69	98.77	96.91	99.00	96.88
PC1	94.14	95.68	97.42	97.31	95.92	96.17	98.56	96.65	96.77	97.27
PC2	95.64	98.92	98.47	99.32	98.85	99.23	98.99	99.48	99.03	99.18
PC3	89.18	92.45	92.11	93.59	93.95	94.07	93.39	92.81	92.33	93.30
PC4	92.54	95.85	96.51	95.98	96.15	95.97	96.08	92.61	95.96	96.45
PC5	83.34	84.12	81.45	81.40	82.74	83.45	82.36	80.75	82.88	83.44
Ecl.2.0	87.10	91.70	87.16	94.10	92.68	93.75	91.10	94.28	92.68	91.49
Ecl.2.1	89.57	90.57	87.90	92.53	95.57	93.49	90.86	94.84	93.05	92.96
Ecl.3.0	90.90	90.93	89.70	95.79	97.31	96.01	93.23	96.59	95.33	96.78

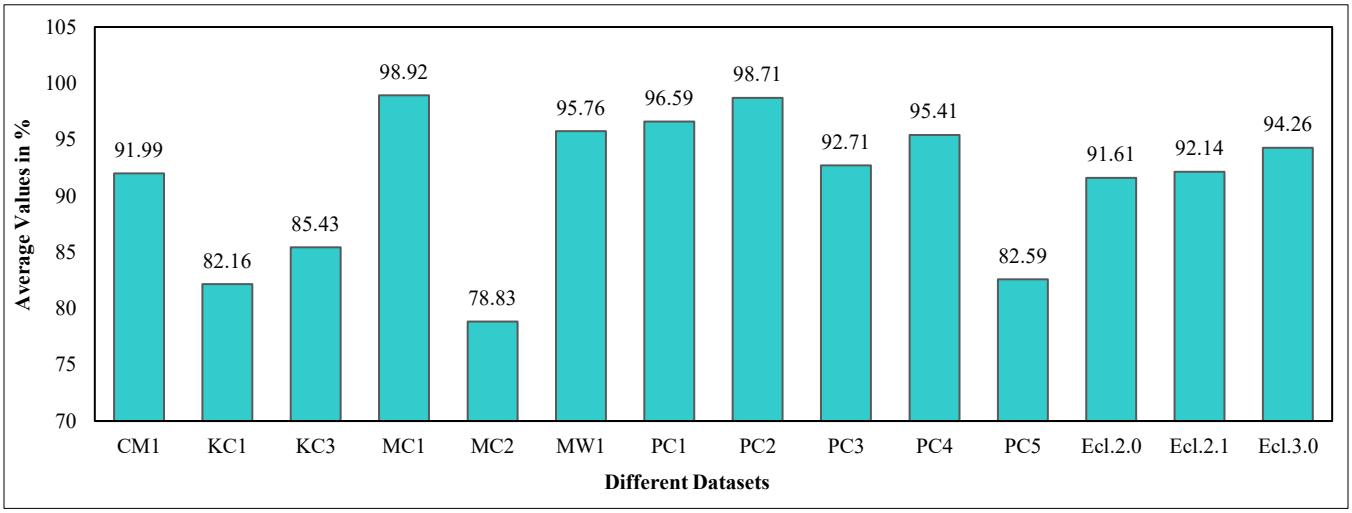

Fig. 10 Average ranking across different classifiers

The relative performance of the classifiers, as shown in Figure 11, is illustrated by their average accuracies across the datasets. The SVM (92.24%) and RF (92.22%) delivered the best overall performance, highlighting their strong generalizability across diverse software defect datasets. Similarly, BAG (92.17%) and AB (92.14%) demonstrated

high accuracy, further confirming the robustness of the ensemble methods. The ALR (91.42%), STK (91.48%), and LMT (91.49%) followed closely, indicating consistent reliability. In contrast, NB (88.87%) showed the lowest average accuracy, likely owing to its strong independence assumptions, while MLR (90.48%) and MLP (89.69%) offered moderate performance. These findings emphasize the effectiveness of kernel-based and ensemble techniques in addressing various software defect prediction challenges.

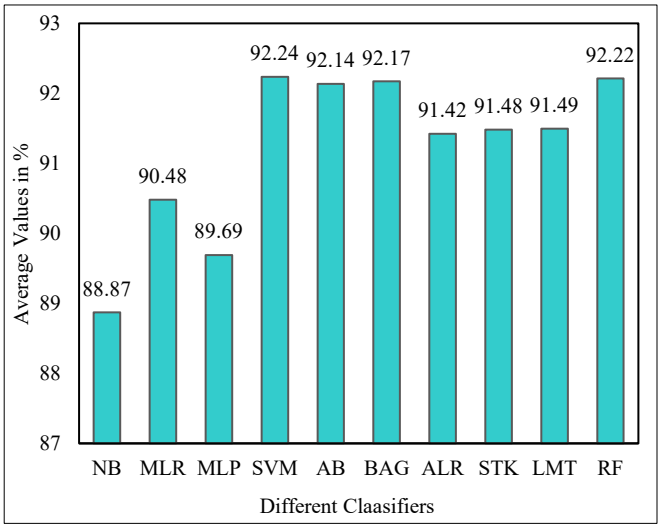

Fig. 11 Average ranking across different datasets

5.1. Comparison with Standard Models

The tests demonstrate that the proposed approach produces superior outcomes than those without preprocessing. However, several models exist to preprocess the data specifically used in software fault datasets. Thus, the performance evaluation results were contrasted with the standard models suggested in the field of research. The suggested HDP approach is analyzed using seven software fault prediction datasets, namely, KC1, KC3, MC2, MW1, PC1, PC2, and PC4, using the Naive Bayes classifier model. The AuC values obtained are presented in Table 5. Some instance reduction models used for these analyses are SMOTE, resampling, and Fisher Linear Discriminant Analysis (FLDA) [26]. The models with higher AuC values for each dataset are highlighted in bold.

Table 5. Performance comparison of HDP model using naïve bayes classifier

Datasets	SMOTE	Resample	FLDA	Proposed HDP
KC1	0.84	0.78	0.85	**0.862**
KC3	0.82	0.85	0.87	**0.881**
MC2	0.72	0.74	0.73	**0.781**
MW1	0.78	0.78	**0.89**	0.878
PC1	0.68	0.67	**0.91**	0.721
PC2	0.79	0.86	**0.91**	0.882
PC4	0.86	0.88	0.83	**0.896**

From the analysis, the proposed model offers better results for four out of seven datasets (KC1, KC3, MC2, PC4) than other existing models in the field of research. However, the model offers better results for the MW1 dataset, but the average performance for dataset PC2 with the minority class has minimal instances. For ease of comprehension, Figure 12 displays the results in Table 5 as a bar chart.

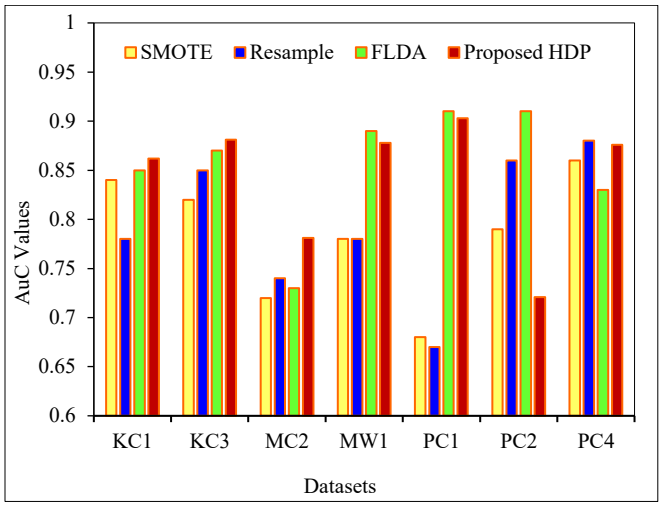

Fig. 12 Performance comparison of proposed HDP with existing model

As with the HFD, the proposed VFD model was also analyzed with five software fault datasets, namely CM1, KC3, MC1, MC2, and MW1, using the Random Forest classifier. The obtained AuC values for different datasets are compared with different feature selection algorithms, such as chi-square, information gain, Pearson correlation, and hybrid feature selection [24], and the values are listed in Table 6. Models with higher AuC values for each dataset are highlighted in bold.

Table 6. Performance comparison of VDP model using random forest classifier

Datasets	CM1	KC3	MC1	MC2	MW1	PC1
Chi-Squared	0.709	0.713	0.904	0.78	0.742	0.882
Information Gain	0.711	0.679	0.904	0.78	0.704	0.882
Pearson Correlation	0.719	0.695	**0.98**	0.738	0.704	0.882
Hybrid Feature Selection	0.726	0.725	0.907	0.779	0.73	0.88
Proposed VDP	**0.804**	**0.756**	0.973	**0.818**	**0.776**	**0.912**

From the analysis, the proposed model offers better results for five out of six datasets, including CM1, KC3, MC2, MW1, and PC1, than the other existing models under comparison. It acquired the first position in five trials. Although the model seems to have a second position with an AuC value of 0.973 for dataset MC1, the difference between the first and second positions was minimal. Thus, the model offers better AuC values for most of the datasets. The values presented in Table 6 are presented as a line graph in Figure 13 for easy understanding.

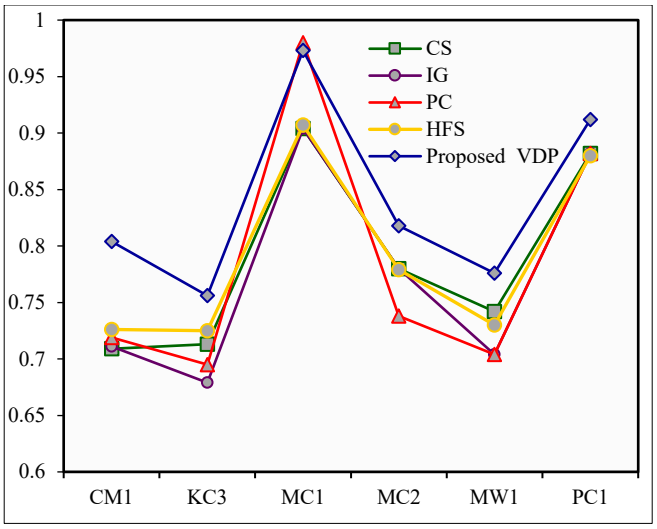

Fig. 13 Performance comparison of proposed VDP with existing models

5.2. Comparison with State-of-the-Art Models

To evaluate the performance of the overall model, both HDP and VDP were applied for dimension selections, and the pre-processed dataset was then evaluated using the Naive Bayes and C4.5 classifiers. The model has been experimented with 12 datasets, and the outcomes are examined in relation to those of other existing models that apply feature selection and, for instance, selection with the aim of improving the input data quality in the software fault detection field.

The existing algorithms used for the comparison are Rough-KNN Noise-Filter Easy Ensemble (RKEE) [12], Information Gain with Threshold-based Clustering (TC+IR) [18], noise filtering and imbalance distribution removal (NFIR) [19], information gain+ symmetric uncertainty+ random under-sampling (ISR) [11], Chi-Square+Symmetric uncertainty+ Random under-sampling (CSR) [11], learnable three-line hybrid feature fusion (LTHFFA) [44], and SHapley Additive exPlanations (SHAP) and local interpretable model-agnostic explanations (LIME) techniques [45].

Table 7 displays the AuC metric values for the proposed and the existing models. The' None' model shows that no preprocessing was applied to the datasets.

Table 7. Performance comparison of proposed model with existing models

Various Models		Different Datasets											
		CM1	**MC2**	**MW1**	**KC1**	**KC3**	**PC1**	**PC3**	**PC4**	**PC5**	**Ecl.2.0**	**Ecl.2.1**	**Ecl.3.0**
Naïve Bayes	None	0.748	0.71	0.741	0.718	0.808	0.756	0.772	0.84	0.869	0.796	0.746	0.762
	RKEE	0.776	0.688	0.809	0.809	0.784	0.809	0.803	0.839	**0.954**	0.823	0.766	0.780
	TC+IR	0.777	0.648	0.781	-	0.823	0.785	-	0.81	0.847	0.799	0.773	**0.784**
	NFIR	-	-	-	0.635	0.669	0.552	0.668	-	-	-	-	-
	ISR	0.767	0.685	0.778	0.81	0.786	0.767	**0.809**	0.841	0.951	0.83	0.766	0.78
	CSR	0.730	0.662	0.771	0.787	0.805	0.712	0.803	0.839	0.932	0.825	0.761	0.781
	Proposed	**0.793**	**0.726**	**0.821**	**0.871**	**0.843**	**0.812**	0.791	**0.887**	0.943	**0.843**	**0.779**	0.774
C4.5	None	0.506	0.562	0.493	0.536	0.605	0.650	0.589	0.752	0.472	0.664	0.586	0.637
	RKEE	0.674	0.589	0.652	0.715	0.765	0.8	0.741	**0.889**	**0.936**	0.786	0.745	0.755
	TC+IR	0.677	0.604	0.687	-	0.733	**0.775**	0.742	0.879	-	0.784	0.762	0.749
	Proposed	**0.682**	**0.623**	**0.729**	**0.731**	**0.798**	0.764	**0.793**	0.864	0.921	**0.808**	**0.781**	**0.783**
RF	NULL	0.583	0.579	0.518	0.546	0.583	0.606	0.633	0.692	0.672	0.604	0.604	0.693
	KPCA	0.597	0.585	0.645	0.620	0.565	0.647	0.664	0.641	0.618	0.628	0.592	0.538
	LLE	0.562	0.635	0.623	0.608	0.537	0.659	0.644	0.658	0.681	0.690	0.648	0.571
	BAVSSA	0.559	0.638	0.707	0.650	0.674	0.665	0.704	0.623	0.679	0.616	0.678	0.565
	SLLE	0.601	0.636	0.543	0.589	0.547	0.597	0.659	0.648	0.666	0.633	0.569	0.624
	Kt-SNE	0.536	0.513	0.579	0.551	0.549	0.545	0.538	0.582	0.616	0.586	0.567	0.555
	KS-LLE	0.531	0.623	0.592	0.641	0.507	0.511	0.518	0.630	0.658	0.593	0.531	0.545
	KSC	0.527	0.640	0.507	0.610	0.527	0.555	0.520	0.694	0.687	0.673	0.642	0.619
	LKB	0.654	0.683	0.708	0.710	0.602	0.670	0.667	0.687	0.712	0.672	0.678	0.644
	LTHFFA	0.674	0.705	0.665	**0.717**	0.685	0.680	0.694	**0.724**	**0.744**	0.673	0.601	0.615
	LIME &SHAP	**0.680**	0.706	**0.721**	0.560	0.710	0.670	0.680	0.692	0.672	0.604	0.604	0.693
	Proposed	0.676	**0.723**	0.719	0.689	**0.728**	**0.696**	**0.709**	0.712	0.719	**0.692**	**0.688**	**0.715**

The other methods such as Kernel Principal Component Analysis (KPCA), Locally Linear Embedding (LLE), Binary Adaptive Variable Sparrow Search Algorithm (BAVSSA), Sparse Local Linear Embedding (SLLE), KPCA and t-Random Neighborhood Embedding (Kt-SNE), Enhanced Stream Shape Learning Method (KS-LLE), KPCA+SLLE with Correlation Analysis (KSC), Learnable Weight +Kernel Features selected by BAVSSA (LKB) are obtained from Tang et al. (2024). The 'None' model indicates no pre-processing is applied to the datasets. The results revealed that all existing models performed well in classifying fault-prone software modules. The proposed model outperforms the Naive Bayes

classifier in nine of 12 datasets. In comparison, the C4.5 classifier and RF classifier outperform it in nine and seven out of 12 datasets, with an increase in AuC values.

From the values presented in Table 7, when compared with datasets without preprocessing, all the existing models provide good results in classifying the fault-prone software modules. However, the proposed model won nine out of 12 datasets with the Naive Bayes classifier. Its losses for three datasets: PC3, PC5, and Eclipse 3.0. Thus, the average rate of the results for the existing models, such as RKEE, TC+IR, NFIR, ISR, CSR, and the proposed model is 80.3%, 78.3%, 63.1%, 79.8%, 78.4%, and 82.4%, respectively, with the Naive Bayes classifier. Similarly, with the C4.5 classifier, the proposed model wins for 9 datasets out of 12 datasets.

The average rates of the results for the RKEE, TC+IR, and proposed models were 75.4%, 73.9%, and 77.3%, respectively, with a C4.5 classifier. Thus, the increase in AuC values with respect to the other classifiers ranged from 2.6% to 5.2%. With extensive experimental and result analysis, it can be seen that the proposed horizontal and vertical dimension selection model offers the best results in many of the experiments compared to most of the existing models. However, there are some limitations to the proposed model. The model provided average results with the lowest number of instances in the minority classes, specifically when the defect samples in the dataset were below 5%. In addition, the model requires more time to select the features using a leave-one-out cross-validated error rate, particularly when there are significantly more features than instances in the datasets.

6. Discussion

The findings of this study show that the suggested VDP and HDP approaches provide notable performance gains over the existing approaches. The reason for this improvement is that the model was able to fill a significant research gap that has been mostly ignored in the literature. Most previous studies have addressed feature selection or class imbalance separately, but very few have offered a comprehensive framework that addresses both issues simultaneously.

Many strategies have been developed in the field of software fault prediction to address class imbalances. Some of these strategies include resampling techniques (SMOTE, resampling), whereas others have concentrated on feature selection techniques such as chi-square tests or Information Gain. The intricate interactions between these two critical data quality issues are typically overlooked by these solutions, which frequently concentrate on just one aspect of the issue: either correcting imbalance or improving features. This research gap is caused by the absence of a single framework that deals with both issues simultaneously. The suggested HDP and VDP techniques simultaneously address feature redundancy and class imbalance, bridging this gap. Unlike earlier research, this study provides a more comprehensive solution to these data quality problems by integrating HDP and VDP, offering a novel approach that simultaneously addresses feature redundancy and class imbalance. This work's unique combination of HDP and VDP offers a cohesive solution that simultaneously enhances feature quality and class balance. In contrast, previous approaches have proven useful in resolving either class imbalance or feature selection separately.

The existence of noise in the data and the irrelevant nature of some features, which significantly influence prediction accuracy, further complicate matters. These problems have been addressed independently or insufficiently in many current models, which can produce less consistent and dependable outcomes. Through a comprehensive approach to data preprocessing that addresses both vertical (feature-level) and horizontal (instance-level) dimensions in a coherent manner, the model balances the data and enhances the feature set, which in turn improves classification performance and stability. Earlier strategies, including conventional resampling and feature selection techniques, concentrated on discrete problems in the pipeline for data preprocessing. To improve prediction accuracy and stability across various datasets, this technique is innovative in simultaneously addressing feature redundancy and class imbalance.

Through efficient instance reduction, the HDP technique improves the class balance by guaranteeing that the most pertinent instances are retained even in unbalanced datasets. This contrasts current techniques, such as SMOTE and resampling, which occasionally create synthetic instances or eliminate valuable minority class instances, impairing model performance and causing noise. Without adding unnecessary bias, this approach guarantees that the data distribution remains representative of real-world situations.

Similarly, the VDP technique enhances feature selection by considering feature dependencies and their impact on the model performance. Instead of choosing features based on their worth, as with traditional feature selection approaches, this approach considers classifier input to select features that help the model perform better. This is where VDP shines, compared to more conventional approaches, such as Pearson Correlation or Information Gain, which are not particularly good at capturing these interactions. Using HDP and VDP together creates a stronger and more transferable model, particularly when working with complicated and noisy data. Better overall performance was achieved by methodically handling class imbalance and feature redundancy using this integrated strategy. This is supported by improved accuracy, recall, precision, and AUC metrics across many datasets.

In addition to overcoming the limitations of the existing techniques, the combined HDP and VDP approach offers a flexible solution for managing feature redundancy and class imbalance in complex datasets. This method distinguishes

itself from previous state-of-the-art models by effectively handling moderate to severe class imbalances and noisy data.

Moreover, the proposed model differentiates itself from current methods, such as RKEE, NFIR, and LTHFFA, by addressing class imbalance and feature selection simultaneously within a single data preprocessing framework. There is a notable gap in software failure prediction research owing to the absence of an integrated preprocessing methodology that handles both class imbalance and feature selection. Although current methodologies have provided valuable solutions to one part of the problem, these issues frequently coexist and influence one another in real-world applications. Closing this gap is essential to finding a workable solution.

Because no prior work has offered an integrated approach addressing class imbalance and feature selection within a single framework, the comparison with established methods such as RKEE, NFIR, and LTHFFA emphasizes the originality of the suggested methodology. Therefore, the proposed model provides a more workable answer for software failure prediction in the real world.

It was also shown that earlier models were inaccurate and that software fault prediction models can be made more accurate, stable, and robust by considering both the vertical and horizontal aspects of data preprocessing. The proposed model outperformed state-of-the-art methods in various performance parameters according to the testing. This is particularly true in datasets with noisy attributes and moderate-to-severe class imbalance.

7. Conclusion

This study presents a preprocessing model that optimizes horizontal and vertical dimension selection, specifically for software detection datasets employed in fault prediction. The proposed model has two main phases. The first phase is the HDP handling data imbalance by processing the instances using SMOTE for oversampling and random under-sampling, as well as the edited k-nearest neighbour rule for noise removal. In the second phase, VDP selects significant attributes using quadratic discriminant analysis. The performance of the proposed preprocessing approach was assessed experimentally using a variety of datasets and metrics. The obtained results demonstrate the effective performance of the anticipated model, with a mean accuracy of 87.26% and 87.74% for the HDP and VDP models, respectively. The average rates of error with RMSE and MAE for HDP were 26.61% and 14.14%, respectively, and those of VDP were 23.39% and 13.34%, respectively. The comparative study ensures that the model achieves robust performance in predicting faults in the software module with an increase in the rate of AuC values from 2.6% to 5.2%. Although the model offers better results for most experiments, the results are insufficient when the number of instances in the minority class is minimal. Thus, future work will focus on offering a better solution with 100% accuracy and implementing the model in a real-time environment for further analysis. In addition, future research could explore integrating HDP and VDP with other advanced methods, such as deep learning models, to further enhance the prediction accuracy. Further investigation into how data imbalance impacts the performance of HDP and VDP could provide valuable insights into the scalability of these techniques.

References

[1] Kateryna Alekseieva et al., "State Business Support Programs in Wartime Conditions," *Economic Affairs*, vol. 68, no. 1s, pp. 231-242, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[2] Florian Tambon et al., "Bugs in Large Language Models Generated Code: An Empirical Study," *Empirical Software Engineering*, vol. 30, no. 3, pp. 1-48, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[3] Musa Murtala Abubakar, and Bashiru Lawal, "Exploring the Potential Failure Modes in the Software Development Process," *International Journal of Science for Global Sustainability*, vol. 6, no. 3, pp. 94-104, 2020. [Google Scholar] [Publisher Link]

[4] Santosh S. Rathore, and Sandeep Kumar, "A Study on Software Fault Prediction Techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255-327, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] Bahman Arasteh et al., "Sahand: a Software Fault-Prediction Method using Autoencoder Neural Network and K-Means Algorithm," *Journal of Electronic Testing*, vol. 40, no. 2, pp. 229-243, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] Ebubeogu Amarachukwu Felix, and Sai Peck Lee, "Predicting the Number of Defects in a New Software Version," *PloS One*, vol. 15, no. 3, pp. 1-30, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Zhiqiang Li, Jingwen Niu, and Xiao-Yuan Jing, "Software Defect Prediction: Future Directions and Challenges," *Automated Software Engineering*, vol. 31, no. 1, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[8] Sushant Kumar Pandey, and Anil Kumar Tripathi, "An Empirical Study Toward Dealing with Noise and Class Imbalance Issues in Software Defect Prediction," *Soft Computing*, vol. 25, pp. 13465-13492, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[9] Jianxin Ge, Jiaomin Liu, and Wenyuan Liu, "Comparative Study on Defect Prediction Algorithms of Supervised Learning Software Based on Imbalanced Classification Data Sets," *2018 19th IEEE/ACIS International Conference on Software*

Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Busan, Korea (South), pp. 399-406, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[10] Li Sheng Kong et al., "A Systematic Review on Software Reliability Prediction via Swarm Intelligence Algorithms," *Journal of King Saud University-Computer and Information Sciences*, vol. 36, no. 7, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] Wangshu Liu et al., "Empirical Studies of a Two-Stage Data Preprocessing Approach for Software Fault Prediction," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 38-53, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[12] Saman Riaz, Ali Arshad, and Licheng Jiao, "Rough Noise-Filtered Easy Ensemble for Software Fault Prediction," *IEEE Access*, vol. 6, pp. 46886-46899, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[13] Riski Annisa, Didi Rosiyadi, and Dwiza Riana, "Improved Point Center Algorithm for K-Means Clustering to Increase Software Defect Prediction," *International Journal of Advances in Intelligent Informatics*, vol. 6, no. 3, pp. 328-339, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[14] Bartlomiej Wójcicki, and Robert Dabrowski, "Applying Machine Learning to Software Fault Prediction," *e-Informatica Software Engineering Journal*, vol. 12, no. 1, pp. 1-18, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[15] Jianming Zhan et al., "A Fuzzy C-Means Clustering-Based Hybrid Multivariate Time Series Prediction Framework with Feature Selection," *IEEE Transactions on Fuzzy Systems*, vol. 32, no. 8, pp. 4270-4284, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[16] Hemant Kumar, and Vipin Saxena, "Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study," *Journal of Software Engineering and Applications*, vol. 17, no. 4, pp. 155-171, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[17] Abdullah Alsaeedi, and Mohammad Zubair Khan, "Software Defect Prediction using Supervised Machine Learning and Ensemble Techniques: A Comparative Study," *Journal of Software Engineering and Applications*, vol. 12, no. 5, pp. 85-100, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[18] Jiaqiang Chen et al., "A Two-Stage Data Preprocessing Approach for Software Fault Prediction," *2014 Eighth International Conference on Software Security and Reliability (SERE)*, San Francisco, CA, USA, pp. 20-29, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[19] Ankush Joon, Rajesh Kumar Tyagi, and Krishan Kumar, "Noise Filtering and Imbalance Class Distribution Removal for Optimizing Software Fault Prediction using Best Software Metrics Suite," *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, Coimbatore, India, pp. 1381-1389, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[20] Chubato Wondaferaw Yohannese, Tianrui Li, and Kamal Bashir, "A Three-Stage Based Ensemble Learning for Improved Software Fault Prediction: An Empirical Comparative Study," *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, pp. 1229-1247, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[21] Yangtao Xue et al., "Nonlinear Feature Selection using Gaussian Kernel SVM-RFE for Fault Diagnosis," *Applied Intelligence*, vol. 48, no. 10, pp. 3306-3331, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[22] Santosh S. Rathore, and Sandeep Kumar, "A Decision Tree Logic Based Recommendation System to Select Software Fault Prediction Techniques," *Computing*, vol. 99, no. 3, pp. 255-285, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[23] Kaijie Xue, Jin Yang, and Fang Yao, "Optimal Linear Discriminant Analysis for High-Dimensional Functional Data," *Journal of the American Statistical Association*, vol. 119, no. 546, pp. 1055-1064, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[24] Lina Jia, "A Hybrid Feature Selection Method for Software Defect Prediction," *IOP Conference Series: Materials Science and Engineering*, vol. 394, no. 3, pp. 1-10, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[25] Wahaj Alkaberi, and Fatmah Assiri, "Predicting the Number of Software Faults using Deep Learning," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13222-13231, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[26] Anum Kalsoom et al., "A Dimensionality Reduction-Based Efficient Software Fault Prediction using Fisher Linear Discriminant Analysis (FLDA)," *The Journal of Supercomputing*, vol. 74, no. 9, pp. 4568-4602, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[27] Zhicheng Liu, and Aoqian Zhang, "Sampling for Big Data Profiling: A Survey," *IEEE Access*, vol. 8, pp. 72713-72726, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[28] Rui Zhang, Feiping Nie, and Xuelong Li, "Self-Weighted Supervised Discriminative Feature Selection," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3913-3918, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[29] Haihong Yu, Liangliang Zhang, and Zhanshan Li, "Self-Weighted Supervised Discriminative Feature Selection via Redundancy Minimization," *IEEE Access*, vol. 9, pp. 36968-36975, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[30] S. Sathya Bama, and A. Saravanan, "Efficient Classification using Average Weighted Pattern Score with Attribute Rank based Feature Selection," *International Journal of Intelligent Systems and Applications*, vol. 10, no. 7, pp. 29-42, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[31] S. Sathya Bama, M.S. Irfan Ahmed, and A. Saravanan, "Average Weight based Pattern Frequency for Performing Outlier Mining in Web Documents," *International Journal of Emerging Technology and Advanced Engineering*, vol. 7, no. 9, pp. 702-709, 2017. [Publisher Link]

[32] Tuong Le, "A Hybrid Approach using Oversampling Technique and Cost-Sensitive Learning for Bankruptcy Prediction," *Complexity*, vol. 2019, no. 1, pp. 1-12, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[33] Nitesh V. Chawla et al., "SMOTE: Synthetic Minority Over-Sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002. [CrossRef] [Google Scholar] [Publisher Link]

[34] Herve Donald Teguim Kamdjou, Classification and Variable Selection Using Linear and Quadratic Discriminant Analysis, Bachelor Thesis, University of Duisburg-Essen, 2016. [Online]. Available: https://www.researchgate.net/publication/351664471_Classification_and_Variable_Selection_Using_Linear_and_Quadratic_Discriminant_Analysis

[35] Trevor Hastie, Jerome Friedman, and Robert Tibshirani, *The Elements of Statistical Learning, Data Mining, Inference, and Prediction*, 2nd ed., Springer, New York, pp. 106-119, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[36] Stefan Hrouda-Rasmussen, Quadratic Discriminant Analysis, A Deep Introduction to Quadratic Discriminant Analysis (QDA) with Theory and Python Implementation, Towards Data Science, 2021. [Online]. Available: https://towardsdatascience.com/quadratic-discriminant-analysis-ae55d8a8148a/

[37] Scikit-Learn, Linear and Quadratic Discriminant Analysis, 2025. [Online]. Available: https://scikit-learn.org/stable/modules/lda_qda.html

[38] PROMISE Software Engineering Repository, 2018. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[39] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller, "Predicting Defects for Eclipse," *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, Minneapolis, MN, USA, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[40] Martin Shepperd et al., "Data Quality: Some Comments on the NASA Software Defect Datasets," *IEEE Transactions on Software Engineering*, vol. 39, pp. 1208-1215, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[41] Thomas J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, 1976. [CrossRef] [Google Scholar] [Publisher Link]

[42] Maurice H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*, Elsevier Science Inc, United States, 1977. [Google Scholar] [Publisher Link]

[43] S. Sathya Bama, M.S. Irfan Ahmed, and A. Saravanan, "A Survey on Performance Evaluation Measures for information Retrieval Systems," *International Research Journal of Engineering and Technology*, vol. 2, no. 2, pp. 1015-1020, 2015. [Google Scholar] [Publisher Link]

[44] Yu Tang et al., "A Software Defect Prediction Method based on Learnable Three-Line Hybrid Feature Fusion," *Expert Systems with Applications*, vol. 239, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[45] Susmita Haldar, and Luiz Fernando Capretz, "Interpretable Software Defect Prediction from Project Effort and Static Code Metrics," *Computers*, vol. 13, no. 2, pp. 1-23, 2024. [CrossRef] [Google Scholar] [Publisher Link]