The Bioscan AN INTERNATIONAL QUARTERLY JOURNAL OF LIFE SCIENCES 20(2): 180-193, 2025 www.thebioscan.com

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

[1] K. Yuvaraj, [2] Dr. R. Revathi

Research Scholar, Karpagam Academy of Higher Education, Coimbatore - 641021, India

Assistant Professor, Department of Computer Science, Karpagam Academy of Higher Education, Coimbatore - 641021, India

Email: revathilakshay@gmail.com

DOI: 10.63001/tbs.2025.v20.i02.pp180-193

KEYWORDS
Software fault prediction, Software defects, Class imbalance, SMOTE, Random Sampling, Edited k nearest Neighbour rule, Quadratic discriminant analysis.
Received on:

08-04-2025

Accepted on:

05-05-2025

Published on:

07-06-2025

ABSTRACT

Software fault prediction is a significant research intended for ascertaining the faults in the software modules by analysing its various parameters. It aims at ensuring maximum quality with minimum time, effort, cost and usage of testing resources for the underlying software. Alike 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, instance noise exist in the software defect datasets. This inconsequential data not only leads to inaccurate prediction results but also slows down the performance of the underlying prediction model. To overcome this issue, a data pre-processing model has been proposed that selects the vertical and horizontal dimensions optimally to ensure input data quality. To handle data imbalance in the horizontal dimensions, the hybrid sampling that makes use of both SMOTE for oversampling and random under-sampling is applied over the data. It also uses the edited k nearest neighbour rule for removing noises. On the other hand, significant attributes from the vertical dimensions of the dataset are identified by applying the quadratic discriminant analysis. The experimental analysis has been made to evaluate the performance of the proposed pre-processing model using various datasets. The obtained results exhibit the effective performance of the proposed model as it ensures the data quality in the pre-processed dataset. The comparative study ensures that the proposed model addresses the challenges and achieves robust performance to predict the faults in the software module with increased quality up to 2.6% to 5.2% of AuC values.

INTRODUCTION

In today's world, software evolved as the strongest medium of automated systems and is truly ruling our entire world. Owing to technological growth and e-business, software has become the most significant entity for both individuals as well as businesses (Kneuper, 2018). In the field of software development, identifying the faults in the software at an early stage is the most difficult part. Software fault prediction is a branch of study in software quality assurance which focuses on testing, code inspection and finding faulty modules.

As the size of the software is increasing rapidly, the probability of faults associated with it is also increasing substantially. In general, software faults can be either of the following including syntactic faults, semantic faults, service faults, communication faults and exceptions in the software modules which may be the cause of complexity exist in coding and lack of human awareness (Dhanajayan and Pillai, 2017). These faults are often not only leads to system failure and inferior quality but also affects the reliability of the software that highly affects the organization goodwill and create a monetary loss. As software development is carried out in the phases of the lifecycle, error in one phase completely affects the other phases in which detecting at the end of needs to enhance the entire process of the lifecycle phase and it involves a huge amount of money as well as manpower (Abubakar and Lawal, 2020). Thus, predicting the faults in the software earlier in the software development cycle has gained considerable attention over the last few years (Rathore and Kumar, 2019). Once the faulty modules are predicted by analysis, then it is quite easy to produce high quality and reliable software products.

In general, several statistical models, machine learning algorithms and software computing techniques are widely employed in predicting the faults in the software with the help of data from previously recorded software faults to enhance software quality (Abaei and Selamat, 2014). Classification models are widely used to classify faulty modules from normal ones. However, the models have limited capability in predicting the fault-prone software modules due to various challenges that exist in software fault prediction (Rathore and Kumar, 2019; Felix and Lee, 2020).

A primary concern in predicting faults from the software is the class imbalance problem as there is an unequal distribution of instances between the fault and non-fault classes that often leads to biased classification results (Menzies et al. 2010; Pandey and Tripathi, 2021). The second major concern is about choosing the suitable software metrics to be incorporated in the fault prediction model. A software metric can be clearly specified as the indicator or quantity based characteristics of the software that helps to evaluate its quality. The software metrics has a wide range of usage in analysing the software that includes analysing the quality, assessing the execution time and detecting defect during its development stage, evaluating the effectiveness, controlling executions (Ge et al., 2018). These metrics are broadly classified as size-based metrics using a line of code, quality-based metrics, the complexity of code, Halstead metrics, object-oriented metrics and so on. The usage of these metrics in the prediction model highly affects the classification results (Hall et al., 2012).

This paper focuses on the challenge that exists in software fault prediction in which it addresses the class imbalance problem along with the software metric selection. Only very few contributions that focus on class imbalance problem and metrics selection exists in the literature (Liu et al., 2016, Riaz et al., 2019). With this motivation, the paper aims at proposing a data pre-processing model that enhance the dataset by applying various algorithms in optimizing the horizontal dimensions and vertical dimensions to generate a high-quality dataset that paves the way for quality results. The horizontal data pre-processing focuses on handling imbalanced datasets by applying hybrid sampling with SMOTE oversampling and random under-sampling along with edited k nearest neighbour rule for removing noises. The vertical data pre- processing aims at selecting significant attributes from the entire feature set by utilizing the quadratic discriminant analysis. Extensive experimental analysis has been made to evaluate the performance of the proposed pre-processing model using various datasets and with various classifiers. The obtained results are also compared with various similar models in the study.

The organization of the paper is as follows. A detailed review related to the proposed study from the literature is presented in Section 2. Section 3 explains the proposed model with a suitable architecture framework along with the two phases of the work such as horizontal data pre-processing and vertical data pre-processing in the following subsections. Section 4 deals with the experimental setup and the dataset used along with the performance metrics used in the evaluation. Section 5 presents the detailed analysis of the results obtained in the experiments and compares the results with other existing models. Section 6 concludes the paper with the identified future directions.

1. Related Work

Software defect prediction is the most prominent field of research for predicting fault- prone software modules. Several pieces of research focus on applying machine learning classifiers to discriminate the software modules as a fault and non-fault (Annisa et al., 2020; Wójcicki and Dabrowski, 2018) besides applying other techniques such as clustering (Abaei and Selamat, 2014; Arshad et al., 2018), deep learning (Brumfield, 2020) for effective results. Usually, the classifiers are trained with the training set for predicting the faults (Alsaeedi and Khan, 2019). However, the fault class will obviously be a minority class having a minimum number of instances that results in the imbalanced dataset. With imbalanced class datasets, the prediction performance of the underlying model always gets compromised. Thus, the quality of the dataset is most important for better prediction results. Few of the researchers have contributed their research on instance reduction and feature selection specifically suitable for fault predictions.

A two-stage data pre-processing approach was proposed to select the significant features and to reduce the instances of the fault datasets. It utilizes Symmetrical uncertainty along with threshold-based clustering for selecting features ensuring high relevancy and low redundancy and applies random sampling to make the class balanced (Chen et al., 2014; Liu et al., 2015). The obtained results are limited to 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 pre-processing results (Joon et al., 2020). Similarly, information gain based feature selection, SMOTE based resampling and iterative noise filter based on the fusion of classifiers for noise removal was proposed but the work lacks the proof for its efficiency comparison with other models (Yohannese et al., 2018).

A linear kernel support vector machine based recursive feature elimination (SVM-RFE) was proposed and evaluated using an SVM classifier to predict the faults in the software datasets. However, the selected features are more in numbers thus makes the classification process difficult (Xue et al., 2018). A recommendation system to select the fault prediction models using decision tree logic and fault characteristics were suggested (Rathore and Kumar, 2017). An improved regularized linear discriminant analysis was employed to select the significant features and was experimented with a few DNA microarray gene expression datasets (Sharma et al., 2014). A hybrid feature selection model using chi-squared, information gain and correlation was suggested (Jia, 2018). However, the model lacks experimental analysis.

An idea of using a partitioning filter with an iterative procedure was proposed to remove the instances that are identified as the noise was suggested (Khoshgoftaar and Rebours, 2007).

However, the results are still less significant for many datasets. An analysis has been made for sampling techniques in which random under-sampling offers better results (Pandey and Tripathi, 2021). An efficient dimensionality reduction model was proposed that utilizes Fisher linear discriminant analysis (FLDA) and it produces good results with resampling (Kalsoom et al., 2018). A survey on various sampling techniques along with their categories was made specifically for big data (Liu and Zhang, 2020).

Apart from fault predictions, the study has been carried out with other data pre- processing techniques recommended for various other prediction models. A self-weighted supervised discriminative feature selection (SSD-FS) method was proposed that utilizes sparsity-inducing regularization which has better results than many other sparse based feature selections. However, the performed experiments are limited to kNN and SVM classifiers (Zhang et al., 2017). Recently, the work was extended by introducing a redundancy matrix- based framework to minimize redundancy (Yu et al., 2021). An average weighted pattern score with attribute rank based feature selection was proposed (Sathya Bama and Saravanan, 2019) with the idea of using weights for different patterns (Sathya Bama et al., 2017), however, the model fails to apply the resampling process to balance the classes. A SMOTE-ENN algorithm for balancing the class ratio along with the CBoost, a cost-sensitive learning framework was suggested, but it is specifically experimented with bankruptcy detection (Le et al., 2019).

From the review of literature, it is clear that only a few of the methods focus on the issues of imbalanced classes and software metrics selection. This shows that there is an immediate need for a novel yet effective data pre-processing model that suits better for predicting faults from the software fault datasets. Thus, the paper suggests a data pre- processing model that is suitable for effective software fault prediction.

2. Proposed Pre-processing Model

The proposed optimized horizontal and vertical dimension selection model is intended to improve the data quality through a series of data pre-processing such as hybrid sampling and quadratic discriminant analysis specifically for predicting software faults. The 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 end with an inaccurate result. Thus, the model aims at applying sampling techniques such as SMOTE oversampling and random under-sampling to balance the instances in the binary class. It also applies the edited k nearest neighbour approach for removing the noises or outliers in the dataset instances. This phase completely deals with the instances which is 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 as well as 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 pre-processing model with horizontal and vertical dimension optimization approach is shown in Figure 1. Each phase on the proposed model is described in the below subsections.

2.1 Optimizing Horizontal Dimensions

The first step in the proposed data pre-processing model is optimizing the horizontal dimension. It includes hybrid sampling and removal of noise by processing the instances in the nderlying dataset. In the proposed model, hybrid sampling is applied to adjust the instances of the underlying imbalanced dataset to make a balanced dataset. The model applies the SMOTE oversampling and random under-sampling techniques to take the advantage of both the models thereby neutralizing its limitations.

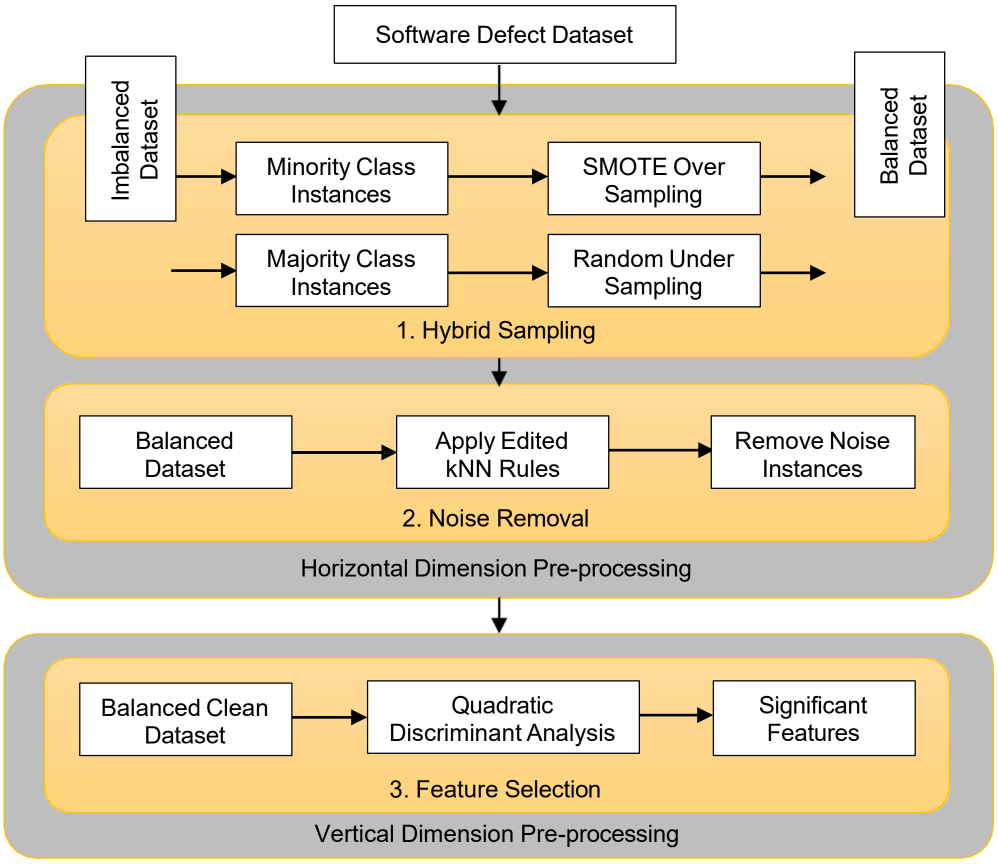

Figure 1. Overall Framework of the Proposed Data Pre-processing Model

Though the dataset is balanced, it may have some outliers which may lead to ineffective results. Thus the proposed model utilizes the edited kNN rules for effectively detecting the outliers to enhance the quality of data. The workflow for optimizing the number of instances for majority and minority class instances of the defect dataset is shown in Figure 2.

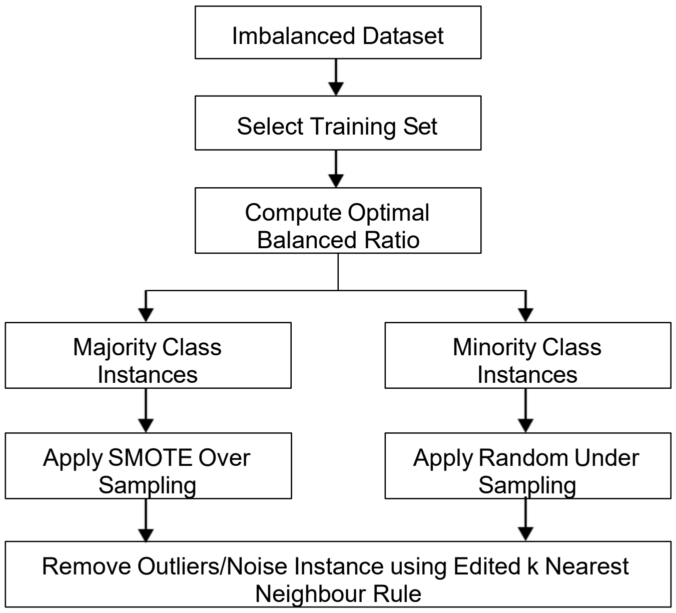
Imbalanced Dataset
Select Training Set
Compute Optimal Balanced Ratio
Majority Class Instances
Minority Class Instances
Apply SMOTE Over Sampling
Apply Random Under Sampling
Remove Outliers/Noise Instance using Edited k Nearest Neighbour Rule

Figure 2. Workflow for Optimizing Instances of the Defect Dataset

2.1.1 Class Balancing using Hybrid Sampling

Hybrid Sampling is prevalent since the imbalanced dataset always provides a biased prediction towards the majority class instances. Thus for effective results, the proposed model utilizes the Synthetic minority oversampling technique (SMOTE) algorithm for oversampling (Chawla et al., 2002) and random under-sampling for balancing the number of instances in the binary class. The simple illustration of Hybrid sampling is shown in Figure 3.

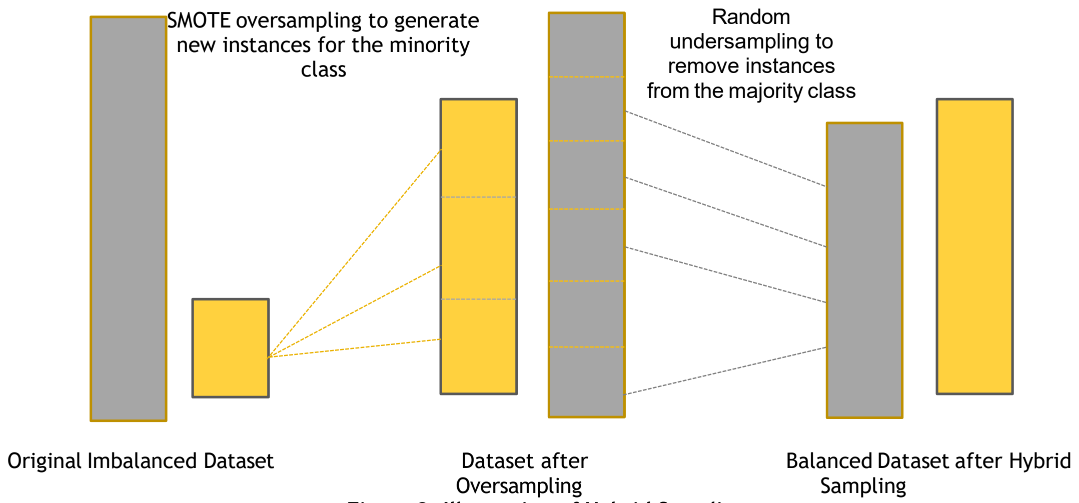
SMOTE oversampling to generate new instances for the minority class
Random undersampling to remove instances from the majority class
Original Imbalanced Dataset
Dataset after Oversampling
Balanced Dataset after Hybrid Sampling

Figure 3. Illustration of Hybrid Sampling

For hybrid sampling, the rate of oversampling and under-sampling has to be computed based on the training set. The rate of sampling is computed as half of the percentage of the difference between the majority and minority class instances. Consider the dataset containing n instances out of which n_1 instances belong to the majority class and n_2 is the number of instances in the minority class. Thus, the rate of sampling is computed as in Eq. (1)

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

Thus, in the proposed model, SMOTE increases the instances and random under- sampling reduces the instances by S_{rate}. On the other hand, the number of instances (nm) to be added and reduced can be given as $(n_1 - n_2)/2$ instances.

SMOTE Oversampling

The SMOTE algorithm generates a non-identical new instances for the minority class based on the similarity between the minority class instances. These added instances will make a class balanced dataset and thus avoids the issues related to class imbalance and overfitting issues. For every instance in the minority class, the k nearest neighbours are obtained. Then the distance between the feature vector of the instance with that of its N neighbours are computed. Then new instance is a synthetic instance that is generated by multiplying the distance with a random number that lies between 0 and 1 and is then added to the feature vector. This is shown in Eq. (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 class, $d(x_1, x_n)$ is the Euclidean distance between the instance x_1 and its neighbour x_n, rn is the random number between 0 and 1. In the proposed model, the value of N is computed as the ratio of the number of instances to be added and the number of instances 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

The random under-sampling is applied to reduces the number of instances in the majority class. It randomly selects the instances from the majority class to balance the dataset. This is an iterative procedure that is carried out until the specified distribution is reached. The random sampling method always offers better results than other under-sampling techniques (Pandey and Tripathi, 2021). Instead of randomly deleting the nm instances, the proposed model splits the entire set of samples into subsamples and then the deletion of instances is carried out. Thus, for each k sample, N samples are deleted randomly in which k is a random number and N can be computed as $(k * n_m)/n_1$. For example, if the number of instances to be deleted is 4000 with the total number of instances as 8000, and if the k value is selected as 10, then N is
5. This implies for every k (=10) instances, N (=5) instances are deleted randomly. The idea is to carry out the selection of instances for deletion to be made in a distributed manner based on the subsamples with k instances.

2.1.2 Noise Removal using Edited K Nearest Neighbour

Hybrid sampling balances the number of instances in the minority and majority classes in the dataset. However, the dataset may contain noises and thus, the noises in the datasets can be identified and removed using Edited K Nearest Neighbour (ENN) approach for effective prediction results. The simple illustration of noise removal using the ENN algorithm is presented in Figure 4.

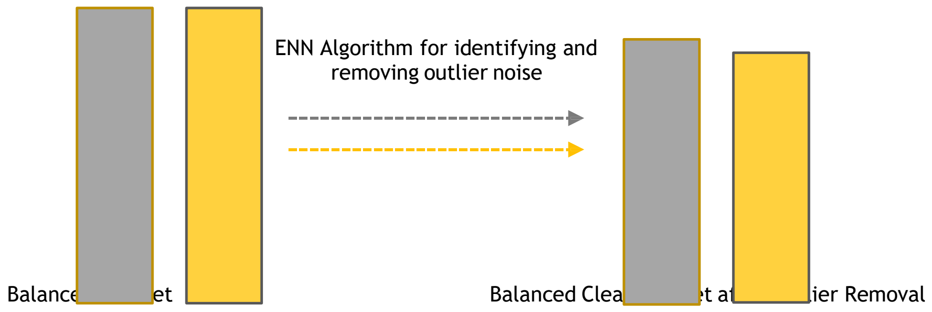
ENN Algorithm for identifying and removing outlier noise
Balanced Dataset
Balanced Clean Dataset after Outlier Removal

Figure 4. Illustration of Noise Removal using ENN Algorithm

The working principle of the ENN approach is that if the sample contains more neighbours from different classes, then it can be considered as an outlier and is removed. Thus, for each instance x, it identifies the k nearest neighbours, say k=10, then x can be considered as outlier and removed if and only if the number of neighbours from other classes is greater than the same class.

Thus, the output of the first phase after applying hybrid sampling and noise removal using ENN is the balanced clean dataset without more variation in the number of instances between the binary class. The algorithm for the proposed horizontal dimension optimization is given below.

```
Algorithm: Horizontal Dimension Optimization
Input: Imabalanced 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
    $S_{rate} = ( n_m/n)*100;$
    //Converting to sampling rate
    //SMOTE Oversampling
    $N = n_m/n2$ //SMOTE percentage Identify the suitable k value
    For each instance x in majority class do
        $N = n_m/n2$
        Find the k nearest neighbours from minority classes
        While $N \neq 0$ do
            Select a neighbour $x_n$
Compute $d(x, x_n) = |x- x_n|$ and select $r_n \epsilon(0,1)$ $x_{new} = x_1 + [d(x_1,x_n)*r_n]$
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
        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
```

2.2 Optimizing Vertical Dimensions

In the proposed model, the vertical dimensions represent the features in the software fault datasets. In order to detect the fault-prone software module, it is highly essential to select the best features that contribute to the classification accuracy. The proposed model utilizes the Quadratic Discriminant Analysis (QDA) for selecting significant features from the underlying datasets. It uses a wrapper approach in the feature selection process in which the best subset having a minimum error is considered as the significant feature set. For computing the error rate of the proposed model makes use of leave-one-out cross-validated error rate (Kamdjou, 2016).

2.2.1 Quadratic Discriminant Analysis

QDA is a reproductive supervised feature reduction model. Here the features are selected that increases the space between the classes (Hastie et al., 2008). 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, the class prediction k can be made using the Naïve Bayes algorithm that maximizes the posterior probability as given in Eq. (3).

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

However, the 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. Then the density ratio can be computed as in Eq. (4).

$$d'(x) = \frac{|\Sigma_1|^{-1/2}}{|\Sigma_2|^{-1/2}} exp\left[-\frac{1}{2}(x - \mu_1)^t\Sigma_1^{-1}(x - \mu_1) + \frac{1}{2}(x - \mu_2)^t\Sigma_2^{-1}(x - \mu_2)\right] \tag{4}$$

Here μ_1 and μ_2 is the mean vector of specific classes class1 and class2 respectively which can be computed by averaging the input variable of each specific classes and the variable
Σ_1 and Σ_2 specify the covariance matrix of specific classes and computed as the covariance of the variables of each specific class (Hrouda-Rasmussen). The expression $(x - \mu_1)^t\Sigma^{-1}(x - \mu_1)$ signifies the Mahalanobis distance between the instance x and the mean of the class (Scikit- learn).
By taking the natural logarithm on both sides in Eq. (2) results in the quadratic function as in Eq. (5).

$$\log(d'(x)) = \frac{1}{2}\log\left(\frac{\Sigma_1}{\Sigma_2}\right) - \frac{1}{2}\left[(x-\mu_1)^t\Sigma_1^{-1}(x-\mu_1) - (x-\mu_2)^t\Sigma_2^{-1}(x-\mu_2)\right] \quad (5)$$

Applying natural logarithm for the posterior probability given in Eq. (1) and substituting the values in Eq. (5) and solving the equation results in Eq. (6) in which cl_1 and cl_2 represents class 1 and class 2.

$$qda(x) = \begin{cases} x \in cl_1 & \text{if } \log(d'(x)) > \log\left[\frac{p_2 c(1|2)}{p_1 c(2|1)}\right] \\ x \in cl_2 & \text{if } \log(d'(x)) \le \log\left[\frac{p_2 c(1|2)}{p_1 c(2|1)}\right] \end{cases} \quad (6)$$

2.2.2 Feature Selection using QDA

In general, the feature selection approaches can be broadly classified into three types such as filter approach, wrapper approach and embedded approach. In the proposed model, the significant features are identified using the wrapper approach. The wrapper approach can either apply a forward selection or backward elimination. The forward selection starts with a null set and at each iteration, a feature is added to the subset that has the least error and stops when there is no change in the error. On the other hand, the backward elimination starts with a full set of features and at each iteration, the feature is identified and eliminated that has a maximum error and the iteration stops when there is not much change in the error rate.
The proposed model makes use of the forward elimination approach to identify the features having the highest biased information with respect to the class using QDA. At each iteration, the discrimination function using QDA is computed for the set of features returned from the forward selection search. Here, each feature subset is evaluated using the QDA and the subset having the minimum error rate is considered to be the best subset that is to be selected.
The proposed model utilizes the leave-one-out cross-validated error rate offered by Kamdjou (2016) as it produces best results than other error rates. The model leaves each instance from the given dataset and applies the QDA discrimination function to the remaining instances. With the evaluation of the remaining instances, the class value is predicted for the instance for the one left. This is carried out for all the instances in each group. Then the error rate can be computed as in Eq. (7).

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

Here, n_{1m} and n_{2m} represents the number of misclassified instances in class 1 and class 2 respectively and n_1 and n_2 specify the number of instances in class 1 and class 2 respectively.
The algorithm for selecting the significant features from the given dataset with the wrapper approach based forward selection search method using quadratic discriminant analysis classifier and leave-one-out-cross validation error rate estimation is given below.

Algorithm: Vertical Dimension Optimization Input: Dataset D with n features
Output: Significant Feature Subset

```
Begin QDA_feature_selection() Fs={ }
While (variation exist 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 as
            in Eq. (6) Estimate the error
            loo_cv_error as in Eq. (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
```

3. Experimental Analysis

This section presents the experimental evaluation and various result analyses made by simulating the experiments on the proposed model to prove its effectiveness. The experiment was performed on the system with an Intel Core i3 processor with a speed of 1.70GHz and 4GB RAM running a 64-bit Windows Operating System. The proposed pre-processing model is implemented using Python and software tools such as Weka and Orange Tools are used to analyse the accuracy of various models statistically.

3.1 Dataset Used

To evaluate the performance level of the proposed model, a few experiments and analyses of obtained results are carried out using various real-time datasets collected from software projects including NASA datasets [Promise] and Bug dataset from Eclipse project [Zimmermann, 2007]. For our analysis, 11 datasets (CM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5) from NASA projects (Shepperd et al., 2013) and 3 datasets (Eclipse 2.0, Eclipse 2.1, Eclipse 3.0) from eclipse projects are used in our experiments. In software defence datasets, the features represent some sort of software metrics that helps to classify the error- prone module.
Various software metrics used in the NASA datasets are lines of code (LOC), complexity metrics and Halstead metric. The complexity measure is proposed by McCabe (1976) in which the increase in the complexity of a path increases the fault possibility. It includes various metrics such as essential, cyclomatic, design and line of code. Halstead (1977) takes the readability of the code as a metric in which hard to read the code indicates the higher possibility of faults which can be divided as the base, derived and line of code metrics. The attributes having continuous values are discretized for easy processing and data management. The Eclipse dataset has a set of metrics that includes code complexity metrics, syntax tree based metrics as well as abstract syntax tree based measures. However, the model pre-processes the datasets by removing non-numeric features thereby utilizing only the numeric features and transforming the datasets into binary class by updating various defect classes as defects. Also, in any datasets, the attributes with a single value will not give any information and so, it is also removed (Liu et al., 2015). The details about the datasets used for the study is given in Table 1. The table presents the number of features, number of instances and the percentage of defect instances in the datasets.

Table 1. Description of 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

3.2 Evaluation Metrics

The performance of any model can be evaluated only by means of quantifying its performance using evaluation metrics. It specifies the quality of the underlying model. Several evaluation metrics are available in the literature for various applications (Sathya Bama et al., 2015). The most frequently used evaluation metrics in many of the applications pertaining to different field of research are accuracy, precision, and error rate. More specifically, some of the most common metrics used in the field of fault detection are utilized in this study.

Classification Accuracy: Classification accuracy is the primary evaluation metric that computes the percentage of correctly classified instances among the total number of instances. Most effectively, it is well effective for balanced datasets containing an equal number of instances in all the classes. Higher values in accuracy indicate the best performance.

AUC: Area under the receiver operating characteristic (ROC) curve eventually applies trapezoid rule to estimate the performance by plotting the true positive and false-positive rates. It evaluates the distance between the target variables. The value of AuC always lies between 0 and 1 representing the worst and best performance of the model respectively.

Precision: Precision represents the rate of positive predictions that are really positive. It can be measured as the ratio of true positive instances to positive instances. The increase in the values indicates an increase in performance.

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

F-measure: It is a metric used to quantify the test accuracy and can be computed as the harmonic mean of precision and recall values.

Root Mean Square Error (RMSE): It is the quantifiable metric that computes the standard deviation of the errors in prediction. It is computed as the square root of the mean of the square of all the errors.

Mean Absolute Error (MAE): It evaluates the closeness between the predictions and the actual outcomes. In general, the lower the errors, the higher the performance of the prediction model. Thus, low values of RMSE and MAE indicates better performance.

4. Results and Discussion

In order to analyse the performance of the model, the proposed horizontal data pre- processing (HDP) and vertical data pre-processing (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 are Naïve Bayes (NB), Multinomial Logistic Regression (MLR), Multiplayer Perceptron (MLP), Support Vector Machine (SVM), AdaBoost (ADB), Bagging (BAG), Additive Logistic Regression (ALR), Stacking (STK), Logistic Model Tree (LMT), Random Forest (RF). The classification accuracy of the classifiers after HDP, VDP and without data pre-processing (WDP) for the classifiers using 14 datasets are presented in Table 2.

Table 2. Classification Accuracy of Different Classifiers using 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 analysing the obtained results, the proposed vertical data pre-processing and horizontal data pre-processing provide better results individually in terms of classification accuracy than without any data pre-processing. In general, the overall accuracy of the classifiers with different datasets without applying any instance reduction or feature selection is 84.73% whereas that of with only HDP and with only VDP are 87.26% and 87.74%. Thus, the upsurge in the accuracy for the proposed HDP and VDP is about 3% and 4% respectively. More specifically, the maximum increase in the accuracy can be seen for the dataset MC2 with HDP as 7.24% and VDP as 8.79% respectively.

On the other hand, the minimum increase in the accuracy with HDP and VDP can be seen through the dataset PC2 as 0.95% and MC1 as 1.21% respectively. This minimum variation is due to the higher range of imbalance in the datasets. More precisely, the datasets PC2 and MC1 is having a very minimum amount of defect instances as 2.1% and 2.3% and thus resampling the instances in the majority and minority classes may not create much difference. The classification accuracy obtained for various classifiers and datasets is presented as a distribution in Figure 5.

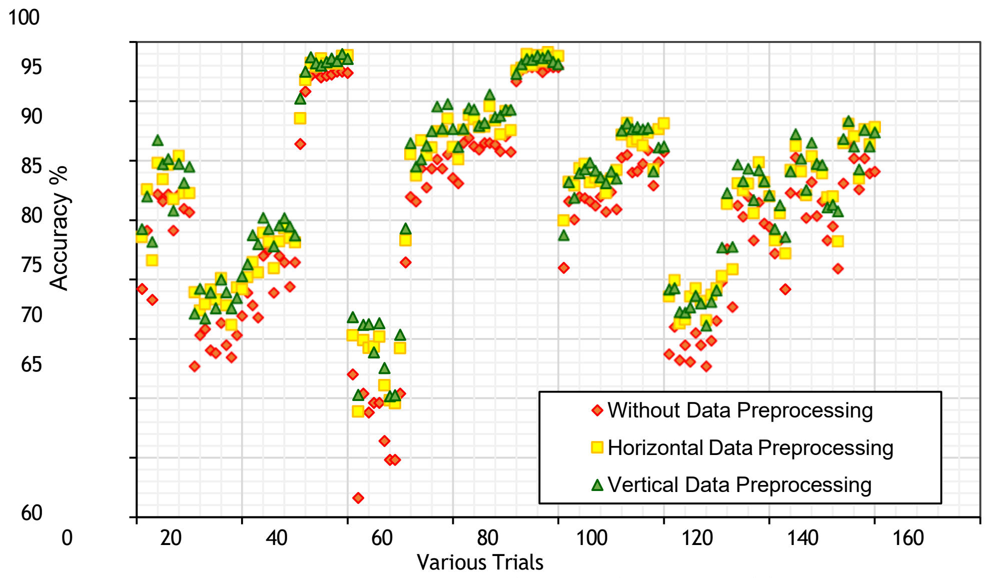

Figure 5. Accuracy Distribution with Various Models

The performance of the proposed HDP and VDP are also analysed individually using various other metrics such as precision, recall, f-measure, AuC, RMSE and MAE by applying the 11 classifiers used in the study. The average obtained values for HDP, VDP and without any data pre-processing (WDP) with the above metrics for 14 software fault datasets are presented in Table 3. From the obtained results, the proposed HDP has an average value of 80.7% precision, 85.11% of recall, 82.75% of f-measure, 73.75% of AuC, 26.61% of RMSE and 14.14% of MAE and on the other hand, the proposed VDP has an average of 80.8% precision, 85.81% of recall, 8316% of f-measure, 75.44% of AuC, 23.39% of RMSE and 13.34% of MAE respectively.

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 increase in values of rate of precision, recall, f-measure and AuC for the proposed HDP are 7.37%, 3.35%, 5.45% and 9.53% respectively than without data pre-processing. Typically, the decrease in the values of error rates such as RMSE and MAE for HDP are highly remarkable with 28.40% and 34.72% down. Correspondingly, the hike in the rate of precision, recall, f-measure and AuC for the proposed VDP are appreciable with 7.52%, 4.19%, 5.98% and 12.30% respectively than with no data pre-processing. Additionally, the decreases in the error rates such as RMSE and MAE for VDP are notable with the decrease of about 37.08% and 38.44% separately. The average of the values presented in Table 3 is presented as a graph in Figure 6.

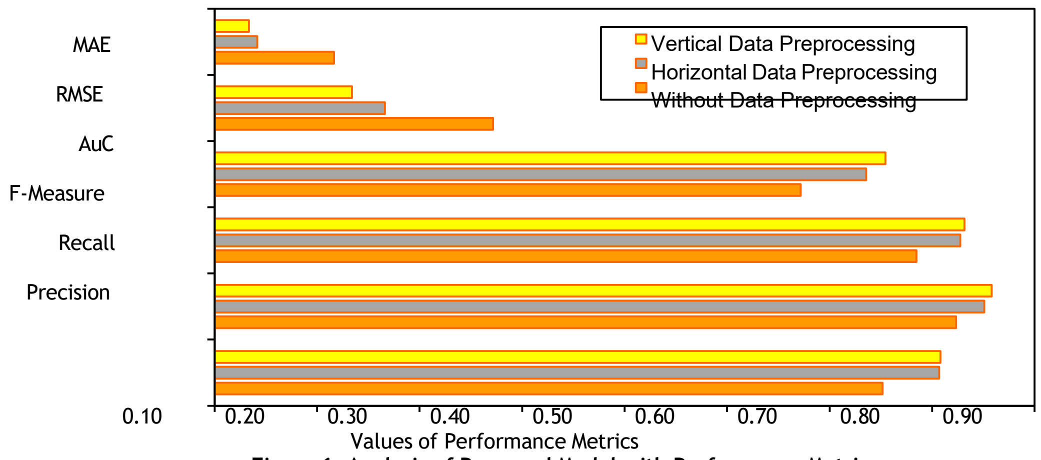

Figure 6. Analysis of Proposed Model with Performance Metrics

The above experiments show that the proposed model offers better results than the results obtained without pre-processing. However, there exist several models to pre-process the data specifically used in software fault datasets. Thus, the performance evaluation has been made and the results are compared with the existing models suggested in the field of study. The proposed HDP model is analysed with 7 software fault prediction datasets such as KC1, KC3, MC2, MW1, PC1, PC2 and PC4 by using the Naïve Bayes classifier model. The obtained AuC values are listed in Table 4. Some of the instance reduction models used for the analyses are SMOTE, resampleing, Fisher Linear Discriminant Analysis (FLDA) (Kalsoom et al., 2018). The models having higher AuC values for each dataset is highlighted with bold text.

Table 4. 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 4 out of 7 datasets (KC1, KC3, MC2, PC4) than the other existing models under the field of research. However, the model offers better results for the MW1 dataset but average performance for the dataset PC2 with the minority class having very minimum instances. The values presented in Table 4 is presented as a bar graph in Figure 7 for easy understanding.

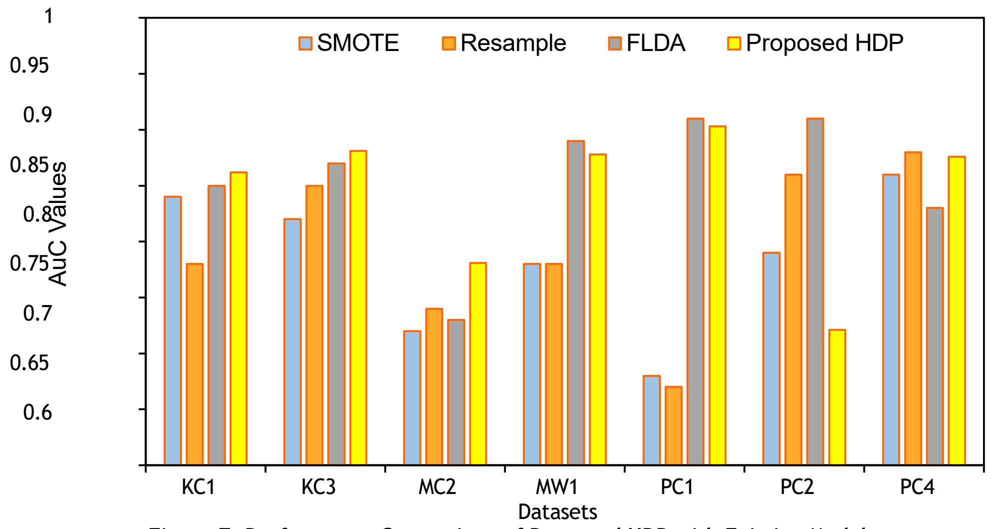

Figure 7. Performance Comparison of Proposed HDP with Existing Model

As with HFD, the proposed VFD model is also analysed with 5 software fault datasets such as CM1, KC3, MC1, MC2 and MW1 using Random Forest classifier. The obtained AuC values for different datasets are compared with different feature selection algorithms such as Chi-Squared, Information Gain, Pearson Correlation, Hybrid Feature Selection (Jia et al., 2018) and the values are listed in Table 5. The models having higher AuC values for each dataset is highlighted with the bold case.

Table 5. Performance Comparison of VDP Model using Random Forest Classifier

Datasets	Chi-Squared	Information Gain	Pearson Correlation	Hybrid Feature Selection	Proposed VDP
CM1	0.709	0.711	0.719	0.726	**0.804**
KC3	0.713	0.679	0.695	0.725	**0.756**
MC1	0.904	0.904	**0.980**	0.907	0.973
MC2	0.78	0.78	0.738	0.779	**0.818**
MW1	0.742	0.704	0.704	0.73	**0.776**
PC1	0.882	0.882	0.882	0.880	**0.912**

From the analysis, the proposed model offers better results for 5 out of 6 datasets including CM1, KC3, MC2, MW1 and PC1 than other existing models under comparison. It acquires the first position for 5 trials. Though the model seems to have a second position with the AuC value as 0.973 for the dataset MC1, the difference between the first and second position is very minimum. Thus, the model offers better AuC values for most of the datasets. The values presented in Table 5 is presented as a line graph in Figure 8 for easy understanding.

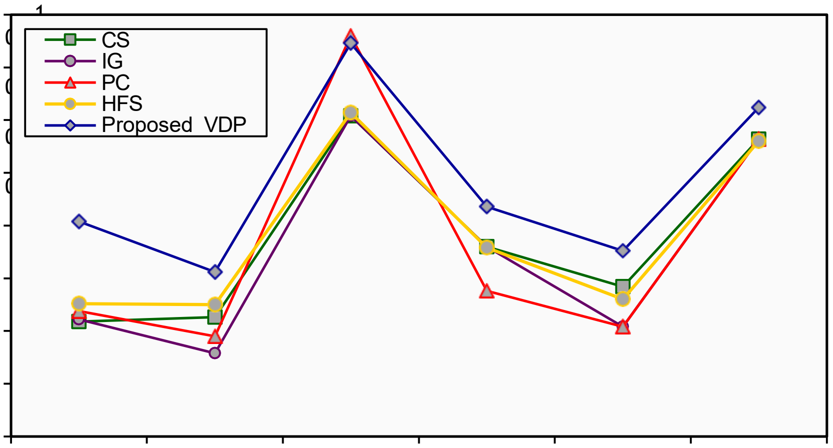

Figure 8. Performance Comparison of Proposed VDP with Existing Models

To be more particular in evaluating the performance of the overall model, both HDP and VDP are applied for dimension selections in which the pre-processed dataset is then evaluated using the Naïve Bayes and C4.5 classifiers. The model has been experimented with 12 datasets and the obtained results are compared with various exiting models that apply feature selection as well as for instance selection with the aim of improving the input data quality in the field of software fault prediction. The existing algorithms that are used for the comparison are Rough-KNN Noise-Filter Easy Ensemble (RKEE) (Riaz et al., 2018), Information Gain with Threshold-based Clustering (TC+IR) (Chen et al., 2014), Noise filtering and imbalance distribution removal (NFIR) (Joon et al., 2020), Information gain+Symmetric uncertainty+ Random under-sampling (ISR) (Liu et al., 2015) and Chi-Square+Symmetric uncertainty+ Random under-sampling (CSR) (Liu et al., 2015). The values for the AuC metrics for the proposed and the existing models are presented in Table 6. The model named 'None' represents that no pre-processing is applied over the datasets.

Table 6. 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	**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**

From the values presented in Table 6, when compared with datasets without pre- processing, all the existing models provide good results in classifying the fault-prone software modules. However, with the Naïve Bayes classifier, the proposed model wins 9 datasets out of
12. Its losses for 3 datasets such as PC3, PC5 and Eclipse 3.0. Thus, the average rate of the results for the exiting models such as RKEE, TC+IR, NFIR, ISR, CSR and proposed model are 80.3%, 78.3%, 63.1%, 79.8%, 78.4% and 82.4% respectively with Naïve Bayes classifier. Similarly, with the C4.5 classifier, the proposed model wins for 9 datasets out of 12. The average rate of the results for the models RKEE, TC+IR, and proposed model are 75.4%, 73.9% and 77.3% respectively with a C4.5 classifier. Thus, the rate of increase in the AuC values with respect to other classifiers ranges from 2.6% to 5.2%.
With the 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 than most of the existing models under comparison. However, there exist some limitations in the proposed model. The model provides average results with the lowest number of instances in the minority classes specifically when the defect samples in the dataset is below 5%. Also, the model takes more time to select the features using a leave-one-out cross-validated error rate, particularly when the number of features is very high than the number of instances in the dataset.

CONCLUSION

This paper presents the data pre-processing model that optimizes the horizontal and vertical dimension selection specifically for software fault datasets used in fault prediction. The proposed model has two main phases in which the first phase is the HDP handles data imbalance by processing the instances using SMOTE for oversampling and random under-sampling as well as edited k nearest neighbour rule for noise removal. The second phase, VDP focuses on selecting significant attributes using quadratic discriminant analysis. The experimental analysis has been made to evaluate the performance of the proposed pre- processing model using various datasets and metrics. The obtained results exhibit the effective performance of the proposed model with an average accuracy of 87.26% and 87.74% for HDP and VDP models respectively. The average rate of error with RMSE and MAE for HDP are 26.61% and 14.14% and that of VDP are 23.39% and 13.34% respectively. The comparative study ensures that the model achieves robust performance in predicting the faults in the software module with the increase in the rate of AuC values from 2.6% to 5.2%. Though the model offers better results for most of the experiments, the results are not good enough when the number of instances in the minority class is very minimum. Thus, the future work focuses on offering a better solution that leads to 100% accuracy and implementing the model in a real- time environment for further analysis.

REFERENCES

- Kneuper, R., 2018. Software processes and lifecycle models. Cham: Springer.
- Dhanajayan, R.C.G. and Pillai, S.A., 2017. SLMBC: spiral life cycle model-based Bayesian classification technique for efficient software fault prediction and classification. Soft Computing, 21(2), pp.403-415.
- Abubakar, M.M. and Lawal, B., 2020. Exploring the Potential Failure Modes in the Software Development Process. International Journal of Science for Global Sustainability, 6(3).
- Rathore, S.S. and Kumar, S., 2019. A study on software fault prediction techniques. Artificial Intelligence Review, 51(2), pp.255-327.
- Abaei G., Selamat A. (2014) Software Fault Prediction Based on Improved Fuzzy Clustering. In: Omatu S., Bersini H., Corchado J., Rodríguez S., Pawlewski P., Bucciarelli E. (eds) Distributed Computing and Artificial Intelligence, 11th International Conference. Advances in Intelligent Systems and Computing, vol 290. Springer, Cham.
- Felix, E.A. and Lee, S.P., 2020. Predicting the number of defects in a new software version. PloS one, 15(3), p.e0229131.
- Menzies T, Milton Z, Burak T, Cukic B, Jiang Y, Bener et al (2010) Defect prediction from static code features: current results, limitations, new approaches. Autom Softw Eng 17(4):375–407
- Pandey, S.K. and Tripathi, A.K., 2021. An empirical study toward dealing with noise and class imbalance issues in software defect prediction. Soft Computing, pp.1-28.
- Ge, J., Liu, J. and Liu, W. (2018) 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, 27-29 June 2018, Busan, 399-406.
- Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic review of fault prediction performance in software engineering. IEEE Trans Softw Eng 38(6):1276– 1304.
- W. Liu, S. Liu, Q. Gu, J. Chen, X. Chen and D. Chen, "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.
- RIAZ, S., ARSHAD, A., JIAO, A. L., Rough Noise-Filtered Easy Ensemble for Software Fault Prediction, IEEE Access, Vol. 6, pp. 46886-46899, 2018.
- Annisa, R., Rosiyadi, D. and Riana, D., 2020. Improved point center algorithm for K- Means clustering to increase software defect prediction. International Journal of Advances in Intelligent Informatics, 6(3), pp.328-339.
- Wójcicki, B. and Dabrowski, R., 2018. Applying machine learning to software fault prediction. e-Informatica Software Engineering Journal, 12(1).
- Arshad, A., Riaz, S., Jiao, L. and Murthy, A., 2018. Semi-supervised deep fuzzy c-mean clustering for software fault prediction. IEEE Access, 6, pp.25675-25685.
- Brumfield, M., 2020. A Deep Learning approach to predict software bugs using micro patterns and software metrics (Doctoral dissertation, Mississippi State University).
- Alsaeedi, A. and Khan, M.Z., 2019. Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. Journal of Software Engineering and Applications, 12(5), pp.85-100.
- Chen, J., Liu, S., Liu, W., Chen, X., Gu, Q. and Chen, D., 2014, June. A two-stage data preprocessing approach for software fault prediction. In 2014 Eighth International Conference on Software Security and Reliability (SERE) (pp. 20-29). IEEE.
- Joon, A., Tyagi, R.K. and Kumar, K., 2020, June. Noise Filtering and Imbalance Class Distribution Removal for Optimizing Software Fault Prediction using Best Software Metrics Suite. In 2020 5th International Conference on Communication and Electronics Systems (ICCES) (pp. 1381-1389). IEEE.
- Yohannese, C.W., Li, T. and Bashir, K., 2018. A three-stage based ensemble learning for improved software fault prediction: an empirical comparative study. International Journal of Computational Intelligence Systems, 11(1), pp.1229-1247.
- Xue, Y., Zhang, L., Wang, B., Zhang, Z. and Li, F., 2018. Nonlinear feature selection using Gaussian kernel SVM-RFE for fault diagnosis. Applied Intelligence, 48(10), pp.3306-3331.
- Rathore, S.S. and Kumar, S., 2017. A decision tree logic based recommendation system to select software fault prediction techniques. Computing, 99(3), pp.255-285.
- Sharma, A., Paliwal, K.K., Imoto, S. and Miyano, S., 2014. A feature selection method using improved regularized linear discriminant analysis. Machine vision and applications, 25(3), pp.775-786.
- Jia, L., 2018, July. A hybrid feature selection method for software defect prediction. In IOP Conference Series: Materials Science and Engineering (Vol. 394, No. 3, p. 032035). IOP Publishing.
- Khoshgoftaar, T.M. and Rebours, P., 2007. Improving software quality prediction by noise filtering techniques. Journal of Computer Science and Technology, 22(3), pp.387- 396.
- Liu, Z. and Zhang, A., 2020. A survey on sampling and profiling over big data (technical report). arXiv preprint arXiv:2005.05079.
- Kalsoom, A., Maqsood, M., Ghazanfar, M.A., Aadil, F. and Rho, S., 2018. A dimensionality reduction-based efficient software fault prediction using Fisher linear discriminant analysis (FLDA). The Journal of Supercomputing, 74(9), pp.4568-4602.
- Zhang, R., Nie, F. and Li, X., 2017. Self-weighted supervised discriminative feature selection. IEEE transactions on neural networks and learning systems, 29(8), pp.3913- 3918.
- Yu, H., Zhang, L. and Li, Z., 2021. Self-Weighted Supervised Discriminative Feature Selection via Redundancy Minimization. IEEE Access, 9, pp.36968-36975.
- Sathya Bama, S. and Saravanan, A., 2019. Efficient Classification using Average Weighted Pattern Score

with Attribute Rank based Feature Selection. International Journal of Intelligent Systems and Applications, 10(7), p.29.

- Sathya Bama, S., Irfan Ahmed, M.S., Saravanan. A., 2017. Average Weight based Pattern frequency for Performing Outlier Mining in Web Documents. International Journal of Emerging Technology and Advanced Engineering, 7(9), pp. 702-709.
- Le, T., Vo, M.T., Vo, B., Lee, M.Y. and Baik, S.W., 2019. A hybrid approach using oversampling technique and cost-sensitive learning for bankruptcy prediction. Complexity, 2019.
- Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, pp.321-357.
- Kamdjou, H. D. T., Classification and Variable Selection Using Linear and Quadratic Discriminant Analysis, Bachelor Thesis, University of Duisburg-Essen, 2016.
- Hastie T., Tibshirani R., Friedman J., “The Elements of Statistical Learning”, Section 4.3, p.106-119, 2008.
- Hrouda-Rasmussen, S., Quadratic Discriminant Analysis, A deep introduction to Quadratic Discriminant Analysis (QDA) with theory and Python implementation (Online).
  Accessed on 25.09.2021. Available at: https://towardsdatascience.com/quadratic-discriminant-analysis-ae55d8a8148a
- Scikit-learn, Linear and Quadratic Discriminant Analysis, (online). Accessed on 25.09.2021. Available at: https://scikit-learn.org/stable/modules/lda_qda.html
- PROMISE Software Engineering Repository. Accessed: Mar. 10, 2018. [Online].
  Available: http://promise.site.uottawa.ca/SERepository
- Zimmermann, T., Premraj, R. and Zeller, A., 2007, May. Predicting defects for eclipse. In Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007) (pp. 9-9). IEEE.
- Shepperd, M.; Song, Q.; Sun, Z.; Mair, C. Data Quality: Some Comments on the NASA Software Defect Datasets. IEEE Trans. Softw. Eng. 2013, 39, 1208–1215.
- McCabe, T.J., 1976. A complexity measure. IEEE Transactions on Software Engineering, (4), pp.308-320.
- Halstead, M.H., 1977. Elements of Software Science (Operating and programming systems series). Elsevier Science Inc.
- Sathya Bama, S., Ahmed, M.I. and Saravanan, A., 2015. A survey on performance evaluation measures for information retrieval systems. International Research Journal of Engineering and Technology, 2(2), pp.1015-1020.