Neural Computing and Applications
https://doi.org/10.1007/s00521-021-05811-3

ORIGINAL ARTICLE

Improved prediction of software defects using ensemble machine learning techniques

Sweta Mehta[1] · K. Sridhar Patnaik[1]

Received: 24 September 2020 / Accepted: 5 February 2021

Abstract
Software testing process is a crucial part in software development. Generally the errors made by developers get fixed at a later stage of the software development process. This increases the impact of the defect. To prevent this, defects need to be predicted during the initial days of the software development, which in turn helps in efficient utilization of the testing resources. Defect prediction process involves classification of software modules into defect prone and non-defect prone. This paper aims to reduce the impact of two major issues faced during defect prediction, i.e., data imbalance and high dimensionality of the defect datasets. In this research work, various software metrics are evaluated using feature selection techniques such as Recursive Feature Elimination (RFE), Correlation-based feature selection, Lasso, Ridge, ElasticNet and Boruta. Logistic Regression, Decision Trees, K-nearest neighbor, Support Vector Machines and Ensemble Learning are some of the algorithms in machine learning that have been used in combination with the feature extraction and feature selection techniques for classifying the modules in software as defect prone and non-defect prone. The proposed model uses combination of Partial Least Square (PLS) Regression and RFE for dimension reduction which is further combined with Synthetic Minority Oversampling Technique due to the imbalanced nature of the used datasets. It has been observed that XGBoost and Stacking Ensemble technique gave best results for all the datasets with defect prediction accuracy more than 0.9 as compared to algorithms used in the research work.

Keywords Defect prediction · Dimension reduction · Data imbalance · Machine learning algorithms · XGBoost · Stacking ensemble classifier

1 Introduction

The surge in usage of devices in our day-to-day lives has increased our dependency on various software systems. Any disruption in the working of software requiring high dependency can have serious consequences. To ensure error-free working of the systems, software must undergo a thorough testing process. In the process of developing software, testing process of the software plays a crucial role. However experienced a developer is, there is always a chance that there might be unforeseen defects. Software can also fail if defects get introduced during maintenance phases [1]. This makes software testing a very significant stage in the software development life cycle. Due to this reason, it is highly desirable to use the testing resources efficiently so that quality of the software is improved. Software Defect Prediction provides a mechanism for effective and efficient usage of testing resources by early prediction of modules in software as defect prone or non-defect prone. In recent research works, machine learning is being used extensively to create defect prediction models. Various techniques have been recommended to solve the task of software defect prediction. These techniques made predictions on the grounds of the historical defect data, the software metrics as well as the algorithm using which predictions are to be done. Classification, clustering and regression are the frequently used techniques for Machine learning algorithms such as Artificial Neural Network, K-Nearest Neighbor, Naive Bayes, Decision Tree and

✉ Sweta Mehta
mtit10008.18@bitmesra.ac.in

K. Sridhar Patnaik
kspatnaik@bitmesra.ac.in

[1] Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi 835315, India

Published online: 02 March 2021

ensemble techniques have been utilized in this domain. Although there are many other algorithms as well, but by using only the above-mentioned algorithms, complex structure of the data cannot be expressed with high accuracy. When these machine learning algorithms are combined with techniques to reduce data imbalance along with dimension reduction techniques, they provide better results. Software defects occurring at later stages of the software development life cycle increase the cost and complexity for fixing it [9]. Therefore, the aim of this research work is to design machine learning models that provide more accurate results in detecting if a software module is defect prone or not and help in finding the undiscovered defects. This can be achieved by extensive analysis of the software metrics and the features of the dataset. Useful features need to be filtered from the feature set to have clean data that can be analyzed properly. This research work focuses on combining feature extraction and feature selection method with an aim to get more accurate results.

2 Background

In this section, we discuss about the main topics involved in software defect prediction. This includes the defect metrics of a software, which forms the basis of software defect datasets. We also discuss about the dimension reduction process using feature extraction and feature selection along with its techniques. Data imbalance issue and various machine learning algorithms have also been covered in this section.

2.1 Software defect metrics

Software has different kind of features. These features indicate the characteristic of the software. Software metrics is used to measure these characteristics. Software performance and quality can be determined using these metrics. Maintainability and reliability of the software can be estimated using this. The capacity to evaluate the complexity of software and its design is an essential requirement for the creation of acceptable software quality standards. They are mainly of four types: Traditional, Object-Oriented, Hybrid and Miscellaneous Metrics [5, 24]. Traditional metrics mainly consists of metrics that measure complexity and functional size of the software. It includes size metrics, e.g., Lines of Code, metrics given by McCabe and Halstead. Properties of Object-Oriented (OO) software like cohesion, inheritance, coupling of Object-Oriented classes are measured using Object-Oriented metrics. Metrics defined by Chidamber and Kemerer [22] belong to this category. The most important OO metrics are Response for a Class, Coupling between Objects along with LOC as they improve efficiency of prediction when used with FS methods [9, 25–27].

2.2 Feature extraction

Feature Extraction is the process that transforms the original feature set to a more relevant and significant feature set which is much more manageable while data processing as compared to the initial dataset. This technique comes to use when the resources need to be minimized or reduced without loss of any useful information. It transforms the feature set to new set of features by ranking the initial feature set based on their significance in predicting the correct output. These features are then linearly combined to get the transformed feature set. Feature extraction is used to improve the capacity of feature expression. Some of the Feature Extraction Techniques are Independent component analysis, Principal Component Analysis, PLS Regression. etc.

PLS Regression Partial Least Squares regression is a feature extraction technique that combines the features from Principal Component Analysis (PCA) and Multiple Regression to give a more generalized output. This technique reduces the feature set to a smaller set of non-correlated features which is then further used to carry out least square regression instead of the initial dataset. PLS components are defined by PLS weights which are linear combinations of the features in the initial feature set. For this research work, PLS has been chosen for feature extraction as it gives better results when compared to other methods and also it does not retain the predictors that are unnecessary for prediction. PLS establishes relation between two sets of variables by reducing their dimension [19].

2.3 Feature selection

Feature selection refers to the process of selecting features from a large set of features given in the dataset. It is one of the key research areas in data analysis involving high-dimensional data. The process of feature selection plays a very important role in developing highly efficient machine learning models. Relevant features are selected from a larger set of features depending on the predetermined measures such as separability of class or classification performance. The main advantage is that it helps to increase accuracy of prediction by reducing overfitting problems. This also helps in faster training process of the machine learning model as the complexity is minimized and makes its interpretability easy. The available feature selection techniques can be grouped into three categories: filter, wrapper and embedded methods.

2.3.1 Filter methods

In filter methods of feature selection the process of selecting features does not depend on the algorithm being used for classification. In this research work, we have used Pearson's Correlation, as used dataset has continuous values for the features and the output values are categorical. Pearson's Correlation between two variables is the linear dependence measure between them. It focuses to find feature subsets that are highly correlated with the output class and has very low values of inter correlation (Fig. 1).

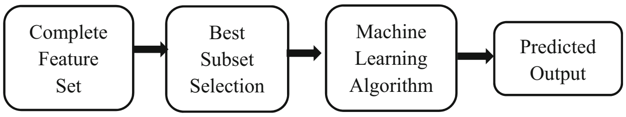

Fig. 1 Feature selection process using filter method

2.3.2 Wrapper methods

In wrapper methods, subsets of features are selected from the complete feature set which is further used to train the model. Based on the analysis of result of the previous model, the features that need to be removed or added from the feature subset are determined (Fig. 2).

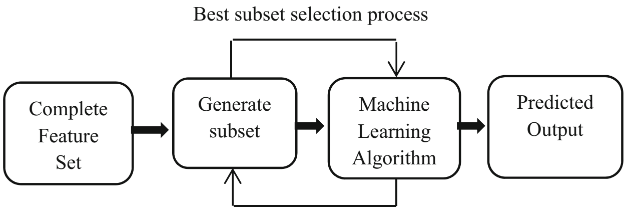

Fig. 2 Feature selection process using wrapper method

Recursive feature elimination RFE represents the wrapper model of feature selection and makes use of greedy algorithm in its implementation. The complete feature set is the starting point for RFE algorithm. The feature set is evaluated using classifier's accuracy. On completion of each iteration, the features in the feature set are ranked and the most irrelevant attribute or rather the least relevant attribute is eliminated. This process goes on till the relevant features are left in the feature set.

Boruta It is a package that provides feature selection by creating shadow features. Shadow features refer to the shuffled copy of the features created by adding randomness into the dataset. This extended dataset is then used to train a Random Forest Classifier and then evaluates the importance of each feature. For each iteration, it verifies if a feature has greater importance as compared to the best of its respective shadow features and then eliminates the least important feature. When the maximum number of specified iterations for random forest is reached or if all the features are either rejected or accepted, then the algorithm terminates.

2.3.3 Embedded methods

It forms a hybrid of the filter methods and wrapper methods discussed above.

Lasso Least Absolute Shrinkage and Selection Operator (LASSO) regularization refers to implementation of Linear Regression with L1 regularization. L1 regularization technique adds the below-mentioned penalty to the loss function. L1 penalty aims to reduce the sum of absolute differences denotes by 'S.' The difference between target value and estimated value is summed up.

$$S = \sum_{i=1}^{n} |y_i - f(x_i)| \tag{1}$$

Here 'S' is the sum of absolute difference, 'y_i' refers to the target value, while '$f(x_i)$' is the estimated value.

This penalty forces lasso to assign zero value to the coefficients of weak features due to which the resultant solution is sparse. Usage of this penalty helps lasso in continuous shrinkage of features and selection of features. It is a technique to perform feature selection and regularization by reducing to minimum the sum of coefficients [17]. If the dataset has n features, then Lasso hypothesis is:

$$h_\Theta(f) = \Theta_0 + \Theta_1 f_1 + \Theta_2 f_2 + \cdots + \Theta_n f_n \tag{2}$$

where feature set $F = \{f_1, f_{2\ldots}, f_n\}$ and Θ is the learnable parameter. Values to Θ are assigned after the training process. Higher Θ value results in greater influence on the result. A certain value of Θ is selected as the threshold value. The features for which Θ value is greater than the threshold belong to the feature subset, while the rest of the features are discarded. This new feature subset is then further used for training the model, and the above steps are repeated to find the optimal feature set which gives the best accuracy of prediction. Using Lasso can give good prediction accuracy as reducing the feature set by removing redundant or irrelevant features reduces the variance without any significant gain in bias. This also results is minimization of overfitting conditions.

Ridge Ridge regression is an extended version of linear regression that makes use of a modified loss function that helps in reduction of complexity in model designing. In order to achieve this target $L2$ penalty is added to achieve regularization. $L2$ is also known as least square error (LSE) as it helps to reduce the square of differences between predicted and actual value. This added penalty is the square of value of the coefficients. $L2$ penalty added to the loss function in Ridge regression is mentioned below:

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{3}$$

Here 'S' is the sum of absolute difference, '$f(x_i)$' is the estimated value, and 'y_i' is the target value. $L2$ provides a single stable solution as compared to $L1$ which can produce multiple solutions.

ElasticNet ElasticNet regression model is a merged version of lasso and ridge regression discussed above. It shrinks coefficients value just as in case of ridge regression and assigns zero as coefficients value to some features just like lasso regression. It makes use of both L1 and L2 penalties while training the model. ElasticNet overcomes the regularization disadvantages present in lasso and ridge regression. In case of presence of very high correlation values between group of features of the input feature set, lasso picks out just one feature from such group of features and eliminates the rest of the features. In order to prevent this elastic net incorporates a quadratic expression in its penalty function. During execution of elastic net, the coefficients of ridge regression are determined initially, followed by shrinking the values of coefficients of the features using lasso regression. It is mainly used in scenarios where the features are independent but form highly correlated groups.

2.4 Data imbalance

The defect dataset used in this research work has a severe class imbalance issue as generally few of the modules are defect prone in a software. This imbalance results in bias in the defect prediction results. Imbalanced class distribution can be handled by mainly two techniques, i.e., oversampling which includes SMOTE and undersampling which includes Near Miss Algorithm. Oversampling technique balances the class distribution by increasing the count of minority class samples by randomly replicating the minority class samples, while the undersampling technique balances the dataset by randomly removing the data samples belonging to the majority class.

2.5 Machine learning algorithms

In this research work of predicting software defects at an early stage of the developing the software, the research of Malhotra et al. [8, 28] was taken into consideration for selecting the machine learning algorithms. Among multiple techniques mentioned in this research such as Bayesian Learners, Ensemble Learners, SVM and Artificial Neural Networks [13], we have considered the following techniques: Artificial Neural Network, SVM, Decision Tree, K-Nearest Neighbor, Logistic Regression, Bagging, Random Forest, Extra Trees, AdaBoost, Gradient Boosting, Stacking and XGBoost [16].

Artificial neural network ANN consists of a set of connected nodes also known as artificial neurons that form the input, output and hidden layers in the network [13]. Each and every connection in the neural network has a weight which is used to compute the output. In this research work, multi-layer perceptron is used. It has one input layer, one output layer and at least one or more hidden layers.

Support vector machine The aim of this technique is to find decision planes using which decision boundaries can be designated. It places a boundary between the instances of the two specified classes.

Decision trees It is a technique to create a model for classification by training it using decision rules generated using the training data. It is a highly efficient machine learning technique that provides multiple possible outcomes and then chooses the best model.

K-nearest neighbor (KNN) It is an algorithm which classifies data based on measure of similarity such as distance functions. In order to predict the output class of a new data, this algorithm tries to find 'k' number of similar data from the training dataset. When using KNN for classification tasks, the value of k should be odd to ensure that the majority class is identified distinctly.

Logistic regression It solves classification problems by using independent variables to get the probabilistic value as predicted output for a binary dependent variable. Dataset in which there is one or more independent variables can be examined using this statistical technique.

Ensemble learning Ensemble method is an algorithm that combines the output of multiple models, which in this case are different classifier models, into one model in order to enhance predictive capability of the model. A major drawback of decision trees is that there are high chances that the tree might overfit. A solution to this problem is using ensemble techniques such as bagging and boosting methods. Bagging makes use of multiple base classifiers for initial prediction results and then aggregates those results for the final prediction, while Boosting tries to convert multiple weak learners into strong classifiers by sequential training of the weak learners.

- *Random forest* Random Forest makes use of numerous decision trees to create a model along with randomly selected training data as well as feature subset. This machine learning algorithm was introduced by Breiman [23]. It is an advancement to tree bagging. During execution random forest tries to obtain the optimal feature set at each decision point. This optimal feature set is obtained randomly from the candidate features of size '*s*.' Many researches conducted in this field have shown that results of random forest are better than other tree bases ensembles and bagging. If the value of 's' is much less than total number of features '*k*,' then randomization degree is very high in random forest.
- *Extra trees* Extra Trees belong to the category of ensemble machine learning methods. It follows the top-down approach to build unpruned decision trees ensemble. Its implementation uses Random Forest and Bagging. This technique is alike Random Forest as Extra Trees are its more randomized version. This randomization is achieved by selecting random splits for the decision trees instead of best splits as in case of Random Forest. When extra trees are used for classification, predictions from different decision trees are selected by using majority voting. But, if it is used for regression tasks, then the final prediction value is the average of all the predictions done by the decision trees.
- *AdaBoost* Adaptive Boosting (AdaBoost) is an ensemble learning technique. It implements series of weak classifiers to form a single strong classifier. It uses decision trees as the base model. Each tree is trained to overcome the weakness of the previously trained decision tree. The tree being trained currently focuses on the misclassified samples that are boosted using weights.
- *Bagging* Bagging combines the output of several models to get better output of the model than a single one. In this research work, Decision Tree has been used as the base model.
- *XGBoost* XGBoost refers to e**X**treme **G**radient **B**oosting which implements gradient boosted trees algorithm. **G**radient **B**oosting algorithm is a supervised machine learning technique which tries to predict the output by combining the predictions of multiple weaker or low performing models. This machine learning technique makes use of ensemble of decision trees using the gradient boosting framework. While training a machine learning model using boosting, the error generated from the previous model is minimized and the influence of better performing models is enhanced. Gradient Boosting helps to minimize the errors of the models better than Boosting. Gradient Boosting uses Gradient Descent algorithm to reduce the errors in sequential models. Further enhancement to this tree-based algorithms is added by the use of XGBoost which uses the benefits of Gradient Boosting algorithm to provide facility to avoid bias or overfitting conditions, tree pruning and parallel processing.
- *Stacking* Stacked Generalization is an advanced ensemble technique in which a brand new model is trained to consolidate the predictions from two or more models that are already trained on the dataset. The models at the base level generally are dissimilar learning algorithms. Thus, stacking ensemble models are of heterogeneous type. The test dataset is then used for prediction using this meta-classifier [14] (Fig. 3).

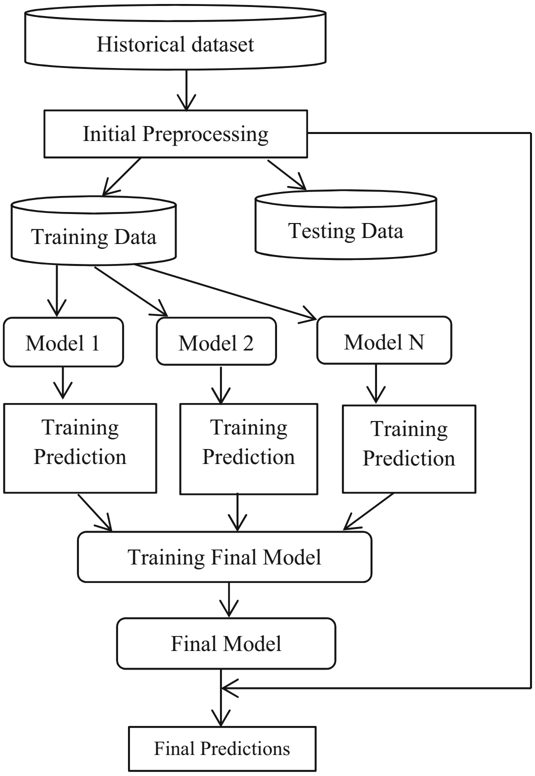

Fig. 3 Flowchart diagram of Stacking Ensemble

3 Literature review

Research in the area of developing defect prediction models using machine learning algorithms has seen a significant rise in past studies. Researchers' have been making use of various Machine Learning Algorithms for classifying the software modules as defect prone or not defect prone. Yalçıner et al. [2] used seven classification algorithms for their performance evaluation in defect prediction in software and identify the best performing algorithm by using NASA datasets that are publically available on PROMISE repository [3]. The research indicated that performance of Random Forests (RF) and Bagging belonging to ensemble learner category produced best results. Two major issues that researchers have faced while using software defect datasets are data imbalance and higher dimensionality of data. Researchers have implemented multiple techniques of feature selection from the dataset.

A new technique of feature selection using support vector machine was proposed by Shenvi [4]. Technique of feature selection based on filter was proposed by Caglayan et al. [5]. Their study used datasets from multiple software to create model based on rank. Since software has very few defect-prone modules, software defect datasets are highly skewed toward not defect-prone modules. In recent years, the issue of imbalanced class due to skewed dataset has been researched a lot. To balance the dataset one might use oversampling technique SMOTE [29] or undersampling technique. Bennin et al. [7] investigated the significance and impact that data resampling has on defect prediction models.

Yang et al. [10] conducted experiments to predict defects in multiple software projects such as Mozilla, Bugzilla, JDT, Columba, PostgreSQL. They used the historical data related to changes made to the code of these software. They developed a software defect prediction model in two phases. The first phase was model building, while the second was prediction phase. Code changes were used to build the model, and any random changes to the software were made to identify if it lead to any software defects.

Wang et al. [6] intended to predict software bugs without diminishing the model performance. Naive Bayes and Random Forest were the two machine learning algorithms used in their research work. Datasets available on publically accessible PROMISE repository were used. Their research helped to understand that data imbalance has a negative impact on the performance of the model which in turn reduces the chances of developing an efficient defect prediction model.

Malhotra [8] analyzed defect prediction for many datasets and pointed out the most frequently used software defect metrics to compute the accuracy of the prediction models. According to her research work, the most used metrics were accuracy, recall, precision, AUC measures, etc. Another category was defined named as miscellaneous category for less frequently used metrics. The research done by Malhotra [8] also examined various machine learning models for software defect prediction.

Palaste et al. [18] have presented a technique that helps to minimize the possibility of false positives. Their study indicates that prediction of software modules as defect-prone and non-defect prone earlier than testing is cost efficient. This paper also throws light upon the data mining techniques that can be used for software defect prediction.

Regardless of whether the noise in defect datasets has been lessened, there is a huge dependency of the defect prediction models on the technique being used for classification. Surely, Song et al. [11] propose that various datasets should utilize various techniques that are available for classification as different techniques may identify varied software modules as defect prone or not defect prone. Panichella et al. [20] likewise inferred that utilizing diverse classification methods can help diversify the process of identification of new distinctive defective modules. Ghotra et al. [21] likewise affirm that the selection of classification algorithm strategies hugely affects the execution of defect prediction models for both restrictive and open-source frameworks.

4 Research methodology

In this research work, two approaches for software defect prediction are proposed: Stacking and XGBoost with combination of Partial Least Square (PLS) Regression and Recursive Feature Elimination (RFE). The model developed is then refined using Synthetic Minority Over-sampling Technique (SMOTE) which helps to overcome the imbalance dataset problem. This model combines the algorithm used for classification with cross-validation, PLS Regression and Recursive Feature Elimination. Initially the dimension was reduced by extracting the features using PLS Regression. After which the features that reduce the redundancy and increase the relevance was selected. This action was performed by Recursive Feature Elimination with cross-validation. XGBoost, i.e., extreme gradient boosting, is an ensemble learning technique that implements gradient boosting tree algorithm which is highly efficient and flexible [15, 16]. Boosting is a category of algorithms which combines multiple weak learners to form a strong learner. Gradient tree boosting is implemented by XGBoost in parallel making the algorithm run faster and also increases its precision. The sections mentioned below describe in detail the research procedures, various prerequisites and machine learning techniques used in this research work.

4.1 Datasets

Publically available datasets such as NASA Data Sets, PROMISE Repository Data Sets were used in this research [1, 3]. The two categories of dataset are mentioned below:

4.1.1 Procedural datasets

- *CM1* CM1 is a project of NASA spacecraft instrument. The programming language in which it is written is ‘C.’ Total number of instances is 498 with 22 attributes. In this 9.7% of the modules are defective.
- *PC1* It has data from function codes written in ‘C’ of a flight software project developed for satellites orbiting the earth. Total number of instances is 1109 with 22 attributes. In this 6.9% of the modules are defective.

4.1.2 Object-oriented datasets

- *KC1* These project data are from a system implemented for storage management required to receive and process ground data written using C++ programming language. Total number of instances is 2109 with 22 attributes. In this 15.4% of the modules are defective.
- *KC2* It is a software system designed for data processing as an extension of KC1 using only some libraries of KC1 that belonged to third-party software. Total number of instances is 522 with 22 attributes. In this 6.3% of the modules are defective.

4.2 Data preprocessing

4.2.1 Missing value processing

Depending on the characteristics of the dataset used in this research, the missing values were filled with the mean value of the feature. For the features having numeric values, average of all the values of that feature is used to fill missing values. In case of a non-numeric feature, the feature value having the highest frequency is used to fill the missing values.

4.2.2 Data normalization

Data Normalization is a crucial part of machine learning model development. In a dataset, the values of the features are in different orders of magnitude because of which some important features might get ignored which might result is a less efficient model. Z-Score normalization is used for this research work as it is efficient in handling outliers. The features of dataset used have exhaustive and large differences between their ranges, especially for attributes related to halstead metrics (halstead_content,halstead_difficulty, halstead_effort, halstead_length, halstead_prog_time, halstead_volume). It is required to bring the values of these attributes on the same scale for which we tried multiple normalization techniques. As we are using PLS regression for feature extraction which combines the features from Principal Component Analysis (PCA) and Multiple Regression, we are interested in the components that maximize the variance therefore among all the normalization techniques we chose z-score normalization.

4.3 Implementation

The research work was done in the following stages for each of the datasets. In the first stage, the best performing feature selection techniques are determined. The second stage deals with development of defect prediction models using the best performing feature selection technique.

Initially data pre-processing is done by handling the missing values in the dataset. Further these data are normalized using z-score normalization technique. The normalized dataset undergoes the process of dimension reduction by applying PLS regression feature extraction technique in combination with multiple Feature Selection techniques such as RFE, Lasso, Ridge, ElasticNet, Boruta and Correlation-based feature selection. SMOTE is then applied to overcome skewness of the data. These data are then used to create defect prediction models using the machine learning algorithms mentioned earlier. Based on the performance analysis of the models, the model giving better prediction accuracy is selected for each dataset (Fig. 4).

4.4 Evaluation criteria

Model evaluation plays a key role in the process of model development. It helps to identify the best model that correctly depicts our data and identifies its performance in future. Evaluation of the performance of techniques used for this research was done by using confusion matrix (Fig. 5). Values of the confusion matrix are listed in Tables 1, 2. There are various criteria to evaluate the models performance such as accuracy, precision, recall, specificity, F-Measure, G-Mean.

Accuracy Accuracy of model is the percentage between correctly predicted samples and the entire sample. It indicates how often the developed model is able to correctly predict the output.

Precision It measures the ratio of true positives that were identified correctly.

Recall Recall is also known as true-positive rate or sensitivity. It helps to identify the ratio of true positives compared to all the true positives.

F-measure It helps to measure the harmonic mean of recall and precision evaluation metrics. Its value ranges from 0 to 1.

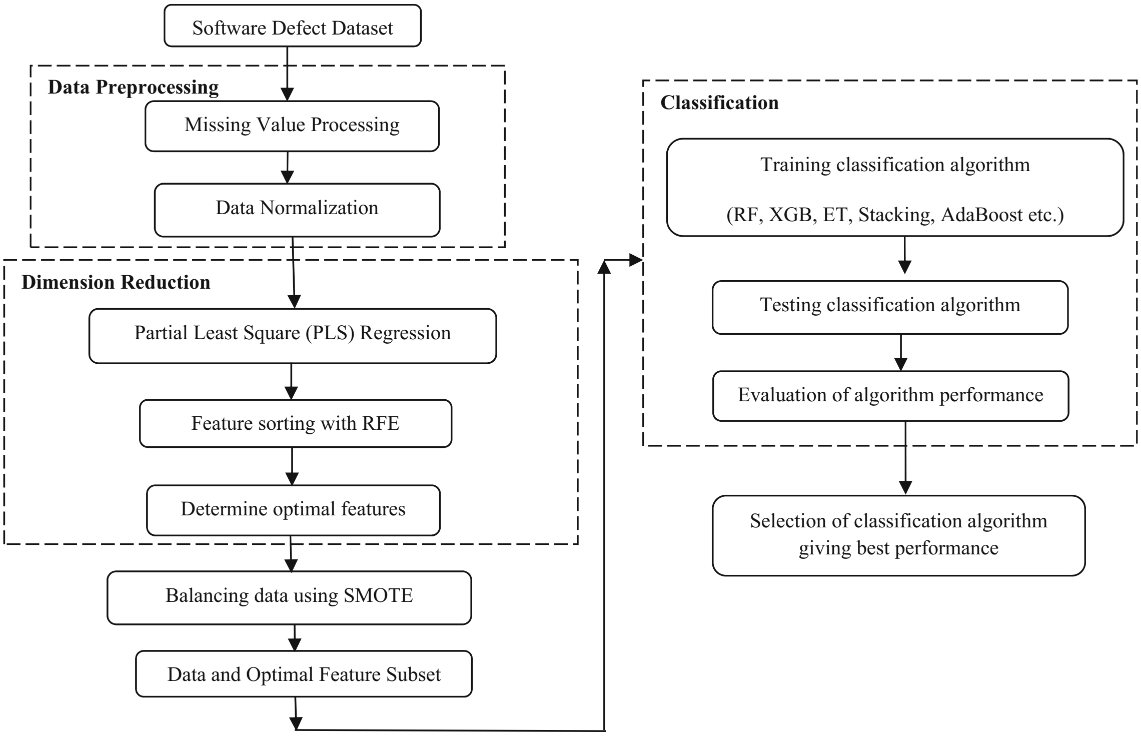
Software Defect Dataset
Data Preprocessing
Missing Value Processing
Data Normalization
Dimension Reduction
Partial Least Square (PLS) Regression
Feature sorting with RFE
Determine optimal features
Balancing data using SMOTE
Data and Optimal Feature Subset
Classification
Training classification algorithm
(RF, XGB, ET, Stacking, AdaBoost etc.)
Testing classification algorithm
Evaluation of algorithm performance
Selection of classification algorithm giving best performance

Fig. 4 Defect prediction model construction process

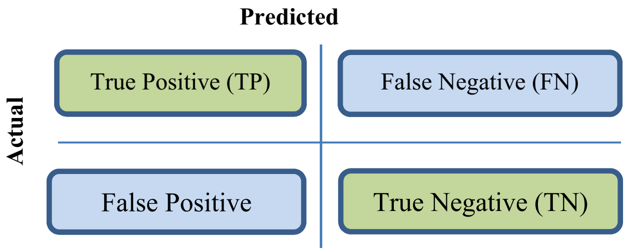

Fig. 5 Representation of confusion matrix

5 Experimental results

5.1 Logistic regression results

The table depicts the accuracy of defect prediction when various feature selection techniques are applied to the dataset. The accuracy values in bold indicate the maximum accuracy achieved for a particular dataset. Based on analysis of the table it did not give any conclusive result that which Feature Selection technique works best for all the datasets when using Logistic Regression.

5.2 Decision tree results

In case of Decision Tree algorithm Recursive feature Elimination (RFE) performed the best when compared to other FS algorithms. The feature set for CM1, KC1 and KC2 reduced by 47%, while for PC1 the feature set was reduced by 28.5%.

5.3 Support vector machine results

In this research, Linear Kernel SVM was used along with regularization parameter [12]. Analysis of SVM results indicates that Lasso was a better feature selector for majority of the datasets by reducing the feature set by 38% for CM1, 23% for KC1 and 52% for PC1.

Table 1 List of values of confusion matrix

Values	Meaning
True positive (TP)	Output predicted by model and output in test data is true
True negative (TN)	Output predicted by model and output in test data is false
False positive (FP)	Output predicted by model is true while output in test data is false
False negative (FN)	Output predicted by model is false while output in test data is true

Table 2 List of evaluation metrics

Evaluation metrics	Values
Accuracy	$\frac{TN+TP}{TP+FP+TN+FN}$
Recall	$\frac{TP}{FN+TP}$
Precision	$\frac{TP}{FP+TP}$
F-measure	$2 * \frac{Precision*Recall}{Precision+Recall}$

5.4 Random forest results

Random Forest provided results that are better than Decision Tree for all the datasets using RFE.

5.5 Extra trees results

Software defect prediction model using Lasso feature selection technique performed better for datasets PC1 and KC1. Datasets CM1 and KC2 perform better with correlation-based feature selection.

5.6 AdaBoost results

Lasso gives better prediction accuracy for KC1, while the rest of the datasets perform well with RFE technique.

5.7 Gradient Boosting results

Gradient Boosting provided better results for all the datasets using RFE.

5.8 Stacking results

The result of Stacking algorithm with RFE is better when compared to the other feature selection techniques used in this experiment. RFE reduces the feature set by 28% for each of the dataset.

5.9 XGBoost result

It provided similar performance for CM1 when XGBoost is used with Lasso Elastic Net and RFE. Dataset PC1 provided good results for Elastic Net and RFE. Correlation-based feature selection gave good result for KC2. RFE results for KC2 are similar to correlation-based feature selection.

5.9.1 Result analysis of the models

The main purpose of this research is to assess the performance of multiple machine learning algorithms when used with various feature selection and feature extraction techniques using the performance measures mentioned above. In the first phase of the research feature selection techniques were tested with different supervised machine learning algorithms for all the datasets in order to identify the best performing feature selection technique (Tables 3, 4, 5, 6, 7, 8, 9, 10, 11). Among Lasso, ElasticNet, Ridge, Boruta, RFE and correlation-based feature selection, RFE gave better performance for most of the datasets. XGBoost, Stacking, Random Forest, Extra Trees and AdaBoost algorithms showed good performance when used with RFE.

It was observed that the performance of a Multi-Layer Perceptron model for procedural datasets (PC1 and CM1),

Table 3 Results of logistic regression

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.84	**0.91**	0.89	0.90	0.89	0.91
PC1	0.815	0.909	0.932	**0.946**	0.941	0.802
KC1	0.744	0.848	0.839	**0.858**	0.855	0.838
KC2	0.819	0.80	**0.857**	0.80	**0.857**	0.847

Bold indicates significant results

Table 4 Results of decision tree

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.86	0.81	0.85	0.81	0.75	**0.89**
PC1	0.90	0.919	0.914	0.896	0.901	**0.932**
KC1	0.801	0.829	0.825	0.815	0.806	**0.843**
KC2	0.828	0.762	0.838	0.705	0.80	**0.867**

Bold indicates significant results

Table 5 Results of SVM

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.80	**0.9**	0.90	0.85	0.87	0.84
PC1	0.815	**0.932**	0.932	0.932	0.905	0.802
KC1	0.735	**0.853**	0.844	0.846	0.8341	0.842
KC2	0.828	0.80	0.838	0.752	0.80	**0.847**

Bold indicates significant results

Table 6 Results of Random Forest

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.85	0.90	0.90	0.89	0.89	**0.90**
PC1	0.949	0.941	0.936	0.950	0.932	**0.95**
KC1	0.841	0.865	0.846	0.858	0.834	**0.879**
KC2	0.857	0.81	0.867	0.80	0.819	**0.91**

Bold indicates significant results

Table 7 Results of ExtraTrees

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	**0.93**	0.91	0.89	0.87	0.89	0.91
PC1	0.90	**0.941**	0.937	0.937	0.928	0.932
KC1	0.832	**0.872**	0.827	0.855	0.834	0.841
KC2	**0.866**	0.790	0.857	0.771	0.819	0.819

Bold indicates significant results

Table 8 Results of AdaBoost

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.91	0.85	0.88	0.85	0.85	**0.946**
PC1	0.891	0.923	0.932	0.932	0.946	**0.96**
KC1	0.829	**0.860**	0.827	0.851	0.846	0.851
KC2	0.838	0.81	0.848	0.819	0.80	**0.873**

Bold indicates significant results

Table 9 Results of Gradient Boosting

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.91	0.89	0.91	0.89	0.85	**0.946**
PC1	0.932	0.934	0.937	0.941	0.941	**0.945**
KC1	0.810	0.855	0.851	0.867	0.844	**0.896**
KC2	0.829	0.819	0.857	0.790	0.819	**0.892**

Bold indicates significant results

Table 10 Results of Stacking

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.91	0.90	0.90	0.90	0.90	**0.90**
PC1	0.928	0.941	0.936	0.932	0.941	**0.946**
KC1	0.819	0.846	0.846	**0.863**	0.829	0.839
KC2	0.83	0.809	0.809	0.790	0.809	**0.834**

Bold indicates significant results

Table 11 Results of XGBoost

Dataset	Correlation	Lasso	Ridge	Elastic Net	Boruta	RFE
CM1	0.94	**0.944**	0.917	**0.944**	0.888	**0.944**
PC1	0.941	0.944	0.944	**0.96**	0.954	**0.96**
KC1	0.806	0.912	0.908	0.887	0.896	**0.941**
KC2	**0.943**	0.879	0.88	0.898	0.8975	0.933

Bold indicates significant results

accuracy of the model after carrying out feature selection based on correlation was better in comparison with other MLP models, while MLP model with SMOTE and dropout implementation performed better. Bagging implementation of decision tree gives better result for correlation-based feature selection approach. Bagged decision tree performs better than individual decision tree model.

The proposed method combines PLS Regression with RFE, since RFE showed good performance among the feature selection techniques, for reducing the dimension of the input data. Five models were created using PLS, RFE, SMOTE and one of the five best performing algorithms, i.e., XGBoost, Stacking, Random Forest, Extra Trees and AdaBoost. It has been observed that the defect prediction accuracy was higher for Stacking model and XGBoost model. The summary of results obtained for all the machine learning algorithms used in this research work is described in Table 12.

Based on the work of Yalçıner et al. [2], the accuracies of best performing machine learning techniques are listed in Table 14. In their experiment the best accuracy of defect prediction was achieved by using Bagging for PC1 and CM1, Random Forest for KC1 and MLP for KC2 dataset. In their work of defect prediction the issue of data imbalance has not been addressed. By addressing the issue of data imbalance and high dimensionality better accuracy of defect prediction is achieved for majority of the machine learning algorithms. Based on the analysis of data in Tables 12 and 13, it is concluded that XGBoost and Stacking-based ensemble technique used in this research gives better results when compared to the techniques used in previous research papers. This comparison is depicted in Fig. 6 for all the datasets used in this research (14).

6 Conclusion and future work

The requirement of defect prediction techniques of higher accuracy and efficiency for software systems will always be a field of interest for researchers. This research work helps in prediction of defects before the actual testing process and hence reduces the time, which in turn reduces

Table 12 Result summary of machine learning algorithms

Project	Metrics	MLP	LR	DT	KNN	SVM	RF	ET	Bagging	AdaBoost	Gradient Boosting	XGB	Stacking
PC1	Accuracy	0.923	0.932	0.918	0.9324	0.9399	0.949	0.945	0.936	**0.901**	**0.945**	**0.968**	**0.968**
	Precision	0.93	0.94	0.95	0.94	0.94	0.96	0.96	0.95	0.91	0.946	**0.971**	**0.96**
	Recall	0.99	1.0	0.96	1.0	1.00	0.99	0.99	0.98	0.95	0.96	**0.98**	**0.97**
	F-Measure	0.96	0.96	0.96	0.96	0.97	0.97	0.97	0.97	0.92	0.94	**0.974**	**0.959**
CM1	Accuracy	0.93	0.91	0.86	0.92	0.93	0.91	0.92	0.92	**0.93**	**0.946**	**0.95**	**0.957**
	Precision	0.95	0.94	0.93	0.93	0.93	0.94	0.94	0.94	0.93	0.93	**0.962**	**0.951**
	Recall	0.98	0.97	0.91	0.99	1	0.97	0.98	0.98	0.96	0.95	**0.97**	**0.96**
	F-Measure	0.96	0.95	0.92	0.96	0.96	0.95	0.96	0.96	0.94	0.937	**0.964**	**0.95**
KC1	Accuracy	0.853	0.838	0.794	0.831	0.845	0.858	0.855	0.822	**0.8767**	**0.896**	**0.96**	**0.96**
	Precision	0.86	0.86	0.87	0.86	0.85	0.86	0.88	0.86	0.86	0.896	**0.962**	**0.96**
	Recall	0.98	0.97	0.89	0.95	0.99	0.97	0.95	0.94	0.94	0.92	**0.97**	**0.98**
	F-Measure	0.92	0.91	0.88	0.90	0.91	0.91	0.92	0.90	0.90	0.91	**0.964**	**0.965**
KC2	Accuracy	0.866	0.828	0.819	0.838	0.8571	0.913	0.838	0.857	**0.9047**	**0.892**	**0.952**	**0.948**
	Precision	0.88	0.86	0.87	0.87	0.85	0.88	0.85	0.87	0.91	0.87	**0.946**	**0.94**
	Recall	0.96	0.94	0.90	0.94	1.0	0.98	0.96	0.96	0.95	0.90	**0.96**	**0.95**
	F-Measure	0.92	0.90	0.89	0.90	0.92	0.93	0.90	0.91	0.92	0.89	**0.94**	**0.942**

Bold indicates significant results

Table 13 Results of the selected models

Dataset	Random Forest	Stacking	XGBoost	AdaBoost
CM1	**0.95**	**0.95**	**0.95**	0.93
PC1	0.9054	**0.967**	**0.9684**	0.9009
KC1	0.8578	**0.9549**	**0.9597**	0.8767
KC2	0.9133	**0.9433**	**0.9523**	0.9047

Bold indicates significant results

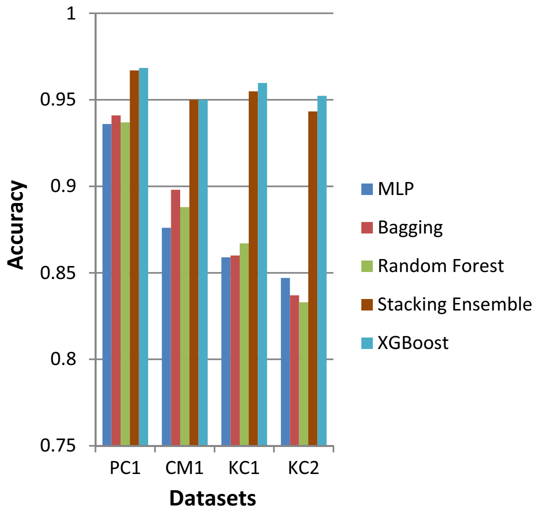

Fig. 6 Comparison of performance between best techniques in Yalçıner et al.'s [2] experiment and best performing techniques of this research work, i.e., XGBoost and Stacking-based ensemble

Table 14 Best performing techniques as per Yalçıner et al. [2] experiment

Dataset	MLP	Bagging	Random Forest
PC1	0.936	**0.941**	0.937
CM1	0.876	**0.898**	0.888
KC1	0.859	0.86	**0.867**
KC2	**0.847**	0.837	0.833

Bold indicates significant results

the cost of software project. The software defect datasets are generally high dimensional and imbalanced. The NASA dataset also faces the same issue resulting in low accuracy in prediction of software defects. Therefore, in order to improve the prediction accuracy a hybrid model is proposed. This model uses Partial Least Square (PLS) regression for feature extraction process, Recursive Feature Elimination (RFE) for selection of relevant feature subset, correlation-based FS and Synthetic Minority Oversampling Technique (SMOTE) for getting a balanced dataset. This refined dataset helps to improve the results of multiple machine learning algorithms. Stacking ensemble learning method and XGBoost provided best results among the techniques used in this research. Further research could be planned to create many other hybrid models so as to develop models which would predict the defects of software systems with more accuracy and minimum errors.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Hauer F, Pretschner A, Schmitt M, Grötsch M (2017) Industrial evaluation of search-based test generation techniques for control systems. In: The 28th international symposium on software reliability engineering (ISSRE)
2. Yalçıner B, Özdeş M (2019) Software defect estimation using machine learning algorithms. In: 4th international conference on computer science and engineering (UBMK), Samsun, Turkey, pp 487–491. https://doi.org/10.1109/UBMK.2019.8907149
3. Shirabad JS, Menzies TJ (2005) The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Ottawa
4. Shenvi AA (2009) Defect prevention with orthogonal defect classification. In: Proceeding ISEC '09 proceedings of the 2nd India software engineering conference
5. Caglayan B, Tosun A et al (2010) Usage of multiple prediction models based on defect categories. In: Proceeding PROMISE '10 proceedings of the 6th international conference on predictive models in software engineering
6. Wang S, Yao X (2013) Using class imbalance learning for software defect prediction. IEEE Trans Reliab 62(2):434–443
7. Bennin KE, Keung J, Monden A, Phannachitta P, Mensah S (2017) The significant effects of data sampling on software defect prioritization and classification. In: Proceedings of the 11th ACM/IEEE international symposium on empirical software engineering and measurement, IEEE Press, pp 364–373
8. Malhotra R (2015) A systematic review of machine learning techniques for software defect prediction. Appl Soft Comput J 27:504–518
9. Reddivari S, Raman J (2019) Software quality prediction: an investigation based on machine learning. In: IEEE 20th International conference on information reuse and integration for data science (IRI), Los Angeles, CA, USA, pp 115–122. https://doi.org/10.1109/IRI.2019.00030
10. Yang X, Lo D, Xia X, Zhang Y, Sun J (2015) Deep learning for just-in-time defect prediction. In: IEEE international conference on software quality, reliability and security, Vancouver, BC, pp 17–26. https://doi.org/10.1109/QRS.2015.14
11. Song Q, Guo Y, Shepperd M (2019) A comprehensive investigation of the role of imbalanced learning for software defect prediction. IEEE Trans Software Eng 45(12):1253–1269. https://doi.org/10.1109/TSE.2018.2836442
12. Arora I, Saha A (2018) Software defect prediction: a comparison between artificial neural network and support vector machine. Advanced computing and communication technologies. Springer, Singapore, pp 51–61
13. Immaculate SD, Begam MF and Floramary M (2019) Software bug prediction using supervised machine learning algorithms. In: International conference on data science and communication (IconDSC), Bangalore, India, pp 1–7, https://doi.org/10.1109/IconDSC.2019.8816965
14. Awad MA, ElNainay MY, Abougabal MS (2017) Predicting bug severity using customized weighted majority voting algorithms. In: Japan-Africa conference on electronics, communications and computers (JAC-ECC), Alexandria, pp 170–175
15. Nielsen D (2016) Tree boosting with XGBoost—why does XGBoost Win "Every" machine learning competition? Norwegian University of Science and Technology, Trondheim
16. Chen T, Guestrin C (2016) XGBoost: a scalable tree boosting system. In: Proceedings of the ACM SIGKDD, international conference on knowledge discovery and data mining (ACM), San Francisco, CA, USA, pp 785–794
17. Muthukrishnan R, Rohini R (2016) LASSO: a feature selection technique in predictive modeling for machine learning. In: IEEE international conference on advances in computer applications (ICACA), Coimbatore, pp18–20. https://doi.org/10.1109/ICACA.2016.7887916
18. Palaste VG, Nandedkar VS (2015) A Survey on software defect prediction using data mining techniques. Int J Innov Res Comput Commun Eng 3(11):10–94
19. Guo G, Mu G (2013) Joint estimation of age, gender and ethnicity: CCA vs. PLS. In: Proceedings of 10th IEEE international conference and workshops on automatic face and gesture recognition (FG), Shanghai, pp 1–6. https://doi.org/10.1109/FG.2013.6553737
20. Panichella A, Oliveto R, Lucia AD (2014) Cross-project defect prediction models: L'union fait la force. In: Proceedings of the international conference on software maintenance, reengineering and reverse engineering (CSMR/WCRE), pp 164–173
21. Ghotra B, McIntosh S, Hassan AE (2015) Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings of the international conference on software engineering (ICSE), pp 789–800
22. Chidamber SR, Kemerer CF (1994) A metrics suite for object-oriented design. IEEE Trans Softw Eng 20(6):476–493
23. Breiman L (2001) Random forests. Mach Learn 45(1):5–32. https://doi.org/10.1023/A:1010933404324
24. Meiliana, Karim S, Warnars HLHS, Gaol FL, Abdurachman E, Soewito B (2017) Software metrics for defect prediction using machine learning approaches: a literature review with PROMISE repository dataset. In: IEEE international conference on cybernetics and computational intelligence, Phuket, pp 19–23
25. Chhillar SR, Gahlot S (2017) An evolution of software metrics: a review. ICAIP 2017:139–143
26. Hariprasad T, Vidhyagaran G, Seenu K, Thirumalai C (2017) Software complexity analysis using halstead metrics. In: International conference on trends in electronics and informatics (ICEI), Tirunelveli, pp 1109–1113. https://doi.org/10.1109/ICOEI.2017.8300883
27. Abreu, Fernando B (1995) Design metrics for OO software system. ECOOP'95, Quantitative Methods Workshop
28. Wang F, Ai J, Zou Z (2019) A cluster-based hybrid feature selection method for defect prediction. In IEEE 19th international conference on software quality, reliability and security (QRS), Sofia, Bulgaria, pp 1–9. https://doi.org/10.1109/QRS.2019.00014
29. Nitesh V Chawla et al. (2002) SMOTE: synthetic minority over-sampling technique. In: Journal of artificial intelligence research, pp 321–357