Research Article

Advances in Machine Learning & Artificial Intelligence

Ensemble Machine Learning Model for Software Defect Prediction

Emmanuel Gbenga Dada[1*], David Opeoluwa Oyewola[2], Stephen Bassi Joseph[3] and Ali Baba Dauda[1]

[1]*Department of Mathematical Sciences, University of Maiduguri, Maiduguri, Nigeria*

[2]*Department of Mathematics and Computer Science, Federal University Kashere, Gombe, Nigeria*

[3]*Department of Computer Engineering, University of Maiduguri, Maiduguri, Nigeria*

[*]Corresponding author
Emmanuel Gbenga Dada, Department of Mathematical Sciences, University of Maiduguri, Maiduguri, Nigeria

Submitted: 16 July 2021; **Accepted**: 26 July 2021; **Published**: 30 July 2021

Citation: *Emmanuel Gbenga Dada, David Opeoluwa Oyewola, Stephen Bassi Joseph and Ali Baba Dauda (2021) Ensemble Machine Learning Model for Software Defect Prediction. Adv Mach Lear Art Inte, 2(1): 11-21.*

Abstract
Software defect prediction is a significant activity in every software firm. It helps in producing quality software by reliable defect prediction, defect elimination, and prediction of modules that are susceptible to defect. Several researchers have proposed different software prediction approaches in the past. However, these conventional software defect predictions are prone to low classification accuracy, time-consuming, and tasking. This paper aims to develop a novel multi-model ensemble machine-learning for software defect prediction. The ensemble technique can reduce inconsistency among training and test datasets and eliminate bias in the training and testing phase of the model, thereby overcoming the downsides that have characterized the existing techniques used for the prediction of a software defect. To address these shortcomings, this paper proposes a new ensemble machine-learning model for software defect prediction using k Nearest Neighbour (kNN), Generalized Linear Model with Elastic Net Regularization (GLMNet), and Linear Discriminant Analysis (LDA) with Random Forest as base learner. Experiments were conducted using the proposed model on CM1, JM1, KC3, and PC3 datasets from the NASA PROMISE repository using the RStudio simulation tool. The ensemble technique achieved 87.69% for CM1 dataset, 81.11% for JM1 dataset, 90.70% for PC3 dataset, and 94.74% for KC3 dataset. The performance of the proposed system was compared with that of other existing techniques in literature in terms of AUC. The ensemble technique achieved 87%, which is better than the other seven state-of-the-art techniques under consideration. On average, the proposed model achieved an overall prediction accuracy of 88.56% for all datasets used for experiments. The results demonstrated that the ensemble model succeeded in effectively predicting the defects in PROMISE datasets that are notorious for their noisy features and high dimensions. This shows that ensemble machine learning is promising and the future of software defect prediction.

Keywords: Ensemble learning, Machine learning, Defect prediction, Software defect, Software metrics

Introduction
Defective code is a threat to the development of better and high-quality software products [1]. Predicting defects in the software development process is very important in producing usable, dependable, efficient, robust, and maintainable software. Software defect prediction is the process of ascertaining the section of software that is likely to have flaws [2]. This field has attracted an enormous research interest with incredible audiences from software practitioners in a very little time. According to Zou and Hastie [3], Tricentis spent an approximated cost of 1.1 trillion USD on testing software in 2016. This shows how important software prediction is to software firms. Software quality assurance is an important aspect of the software development life cycle that decides the activities that will help guarantee the quality of the product [4].

For a long time, software firms have been using manual testing to detect flaws in software products. There is a need for about 27% human intervention for the entire design and implementation of software products using manual software testing [5]. Manual software testing has several drawbacks: it is time-consuming and inefficient for locating and correcting every error existing in the software [5]. Software firms widely adopt models for predicting flaws in software to tackle this problem effectively. These prediction models offer several advantages: the ability to test a software's ability to function under given environmental conditions over a period. They also estimate the effort needed to develop the software, identify risks in software, and assist in the rapid development of software products [6]. These models enhance user's acceptability of the software and decrease the development time and cost of the software at the initial phase of the software development lifecycle (SDLC) by reducing the risk involved.

Different scholars and researchers have suggested several conventional software defect prediction approaches. However, none of these proposals have demonstrated consistency in prediction accuracy [4]. These methods include statistical methods, machine learning methods, parametric models, and mixed algorithms [4]. Apparently, there is a need for 'the best prediction method for a given prediction problem or, conceivably, conclude that fault detection in software is impossible. The dynamic nature and complexity of software have motivated machine learning algorithms to be viable prediction models for accurate software quality prediction. Recently, machine learning has proved to be the most effective approach [6]. Many software developers have used machine learning (ML) methods to categorize software dataset into the defective and non-defective datasets as error prediction models. Researchers execute the classification process by supplying a machine learning classifier with a software dataset as input, whereas the user has foreknowledge of the actual class values. At this phase, the user divides the input dataset into training and testing sets. The ML classifier trains the training dataset and creates a trained model, which it uses for further processing. The classifier using the patterns existing in the input dataset creates the training model. In the subsequent phase, the test dataset is randomly supplied to the classifier and then compared with the trained model, which generates the result in the form of software defect prediction.

Before adopting machine learning models for software defect prediction, some of the techniques employed to solve this problem include requirement-based and object-oriented design [7, 8]. These techniques have demonstrated considerable efficiency in handling these problems. However, they have not succeeded in efficiently handling the issues of low prediction accuracy and software complexity. Conventional software defect prediction methods are characterised by low prediction accuracy and time wastage because of software design sophistication. Therefore, machine learning approaches such as Bagging and Boosting, Naïve Bayes, Support Vector Machine, J48, deep learning, and feature selection techniques were proposed to solve these problems [9-17]. These approaches have proved to be highly effective for predicting software defects compared to other aforementioned methods. Predicting the defects in software requires the extraction of features of a specific software product from software metrics. Machine learning classifier then analyzes these features.

After reviewing the relevant literature, it was discovered that the existing literature does not provide any work that uses ensemble technique to solve the problem of software detect prediction. Moreover, many of the existing work's prediction accuracy is rather low, while some did not use state-of-the-art metrics to evaluate their work performance. Furthermore, some authors did not compare their work with high-performing machine-learning models. Bearing in mind such inadequacy, this work's novelty centers on the use of ensemble machine-learning for software defect prediction. The summary of our contribution is as follows:

1. This paper proposed a multi-model ensemble machine-learning for software defect prediction. The ensemble method's diversity is leveraged. The Ensemble technique chooses the best model from a collection of options. Moreover, an ensemble can create lower inconsistency among training and test datasets. It also lessens bias in the training and testing phase of the model. The proposed model thereby overcomes the drawbacks associated with the existing techniques applied for the prediction of a software defect.
2. The technique employed in this paper makes use of both primary and auxiliary models. The primary model training is carried out using RF models. The grid search algorithm is used to optimize the hyper-parameters of the RF model. The auxiliary model uses the LDA algorithm for secondary learning to complete the ensemble of multiple primary models.
3. The multi-model-based ensemble uses kNN to make the final prediction, and the final prediction results are generated.
4. Average overall prediction accuracy of 88.56% for the proposed model was attained. The proposed model was evaluated using different performance metrics, and the results were compared with other algorithms
5. A recent review of state-of-the-art proposals for defect prediction in software is presented.

The rest of this paper's organization is as follows: Section 2 discusses the related works in the field of software defect prediction. Section 3 explained the proposed methodology used in this work. The presentation of the results and the discussion of the results are in section 4, and section 5 is the conclusion of the paper.

Related Works

This section presents a brief discussion on recent researches done in the field of software defect prediction. Section one explained that machine-learning algorithms had gained wide acceptance among data scientists and software engineers for software defect prediction. This is because of the efficacy of these algorithms in handling the problems of software defect prediction. Jayanthi and Florence applied metrics centered on neural network classifiers to predict software defects [5]. The proposed technique uses decreasing the dataset attribute using the principal component analysis (PCA) approach. The introduction of maximum-likelihood approximation further enhanced the performance of the system. It decreases the fault in PCA data rebuilding. The application of a neural network algorithm helps to predict the defect in the software and generated output. Simulation results demonstrated that their technique has great potential for software defect prediction and so can be adopted. Our approach is distinct from the one adopted by these authors because we used ensemble machine-learning, resulting in greater effectiveness in detecting defects in software.

Ghosh, Rena, and Kansal proposed a nonlinear manifold detection technique for software defect prediction [18]. Their method's objectives are to remove unwanted and inappropriate attributes of high-dimensional datasets by decreasing the dimension to achieve software with higher prediction accuracy and superior quality. The authors compared the performance of their method with other existing feature selection methods using different performance metrics. Results illustrated that the proposed model performance is satisfactory compared to other techniques. The drawback of this approach is that the prediction accuracy is low. We distinguished our work from this one because our proposed method has a higher prediction accuracy than theirs.

Majd et al., presented a technique used for software defect prediction [19]. Their method uses a deep-learning model on static code features. Their technique uses SLDeep to predict software defects. The base learners used were Long Short-Term Memory (LSTM) and random forest. The focus of their approach is on easing the workload of a software developer in locating the flaws in code. Their approach's strength lies in helping software developers build quality software with ease in minimum time. However, the prediction accuracy of 70.2% attained by their technique is still low. Our paper's approach addressed the shortcoming of this work by generating high prediction accuracy on the used datasets.

Qiao et al. applied deep learning to predict the number of flaws in software [20]. Their approach trains a deep learning model to predict the number of defects using publicly available datasets. Experimental results show that the proposed system performed better support vector regression (SVR), fuzzy support vector regression (FSVR), and decision tree regression (DTR). The authors demonstrated that the proposed system notably lowers the mean square error and squared correlation coefficient. The downside of their work is that the lack of performance comparison of the proposed method with other algorithms using more performance metrics to further verifies their system's effectiveness. Unlike their work that failed to compare their model's performance with other high performing machine-learning models, ours compared the performance of our proposed system with other state-of-the-art models.

Sun et al. developed a new algorithm for automatically endorsing appropriate sampling techniques for any data that have a new defect [21]. The approach works by initially rating current sampling approaches with past data that have flaws. It then extracts the data resemblance between past and new flawed data using meta-features. The system creates a recommendation network using a combination of information from ranked sampling approaches and data resemblance. Results demonstrated that the developed system is practicable and efficient. The system's drawback is that the performance is still low, and there is a need to improve on it. In addition, there is a need to use some popular performance metrics such as prediction accuracy, root mean square error, and others to further investigate the performance of the proposed system. Our work differs from theirs in that the proposed model was evaluated using several performance metrics.

Wei et al. applied a dimension reduction technique to predict software defects [22]. The authors used their proposed algorithm to reduce the dimension of defect data. Their proposed system used the SVM classifier as the base learner of the prediction model. Afterward, the system applied a grid search technique to improve the model's parameters before execution of ten-fold cross-validation. Inability to overcome conventional dimensionality reduction algorithms such as data loss is the downside of this technique. This method's strength is that it overcomes the shortcomings of the, which is triggered by inadequate features of data nonlinearity. Despite this advantage, their model's performance is still quite low with respect to prediction accuracy, precision, recall, and F-measure. Our proposed method addressed the downside of this work by proposing a model with high prediction accuracy.

Zhao et al. developed a cost-sensitive Siamese parallel fully connected neural networks technique to detect software systems' flaws [23]. The approach merges the strengths of Siamese networks with that of deep learning into an integrated technique. AdamW algorithm does the training and location of the optima weight for the model. Simulated results revealed that their approach outperformed DSNN, LSTM, DBN, and RNN. The difference between the technique adopted in their paper and ours is that our method is based on the ensemble of many machine-learning models. Moreover, the prediction accuracy of our proposed model is as good as theirs. The authors only used Mathew Correlation Coefficient (MCC) and Accuracy to evaluate their model. In contrast, our proposed model made use of Accuracy, MASE, MSE, and RMSE to evaluate our model.

Shao et al. proposed a correlation weighted class association rule mining (CWCAR) for detecting flaws in software [24]. The system uses a multi-weighted supports-based framework deal with inequality in class and uses a correlation-based heuristic method to allocate feature weight. Simulation results indicated that the proposed system is practicable and efficient for predicting flaws in software. The difference between this paper and ours is that these researchers evaluated their model's performance using Balance, MCC, and Geometrical mean (Gmean). They also compared CWCAR with predictors such as Classification Based on Associations (CBA), Naïve Bayes (NB), Random Forests (RF), Decision Tree (DT), and Partial Decision Trees (PART) while our work compared the performance of the ensemble model with RF, RPART, kNN, GLMNet, and LDA.

The related works discussed above give a brief account of artificial intelligence (AI) applications, machine learning algorithms, deep learning techniques, data mining, and feature selection method to software defect prediction. Literature has proved that machine-learning algorithms can produce high prediction accuracy for software defects [cite this statement]. Incorporating future extraction techniques into machine learning models decreases complexity and improves the general performance of the model.

Methodology

The previous section contains a concise discussion of different machine learning models for predicting defects in a software system. Many of the approaches discussed center on solving the challenge posed by imbalanced data of software faults. However, prediction accuracy and general performance continue to pose a daunting problem to software engineers and academicians. To overcome these challenges, we present a technique that combines ensemble machine learning for dimension reduction and feature reduction for predicting software defects. This section discusses different machine-learning algorithms and the proposed ensemble machine-learning model.

Linear Discriminant Analysis (LDA)

The LDA proposed by Fisher is a leading and standard approach in discriminant analysis for solving classification problems. It is a classic example of a very famous single-label (multi-class) feature extraction method. The LDA achieves a good result when the population is the normal joint distribution, and various classes have homogenous covariance. LDA can locate an ideal prediction matrix M by exploiting inter-class scatter measures and reducing intra-class one simultaneously [25]. The LDA has been widely used as a supervised feature extraction method in single-label (multi-class) classification. LDA makes getting the group posterior Pr (M|X) for the M classification compulsory [26]. Assuming $f_i(x)$ is the class conditioned likelihood of X in class M=i. then let π_i be the likelihood of event before new data is collected in class i, with

$$\sum_{i=1}^{I} \pi_i = 1.$$

from

$$\Pr(M = i|X = x) = \frac{f_i(x)\pi_i}{\sum_{i=1}^{i} \pi_i f_i} \quad (1)$$

Assuming that:

$$f_i(x) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma_i|^{\frac{1}{2}}} \; EXP - \frac{1}{2}(x - \mu_i)^i \Sigma_i^{-1}(x - \mu_i) \quad (2)$$

In LDA, the classes possess mutual covariance matrix $\sum_i = \sum \forall i$ so, compare two classes j and i, the log-ratio is expressed as:

$$log \frac{\Pr(M=i|X=x)}{\Pr(M=i|X=x)} = log \frac{\pi_i}{\pi_j} - \frac{1}{2}(\mu_{i-}\mu_j)\Sigma^{-1}(\mu_{i-}\mu_j) + x^T \Sigma^{-1}(\mu_{i-}\mu_j) \quad (3)$$

Linear discriminant function is produced by this function

$$\delta_i(x) = x^T \Sigma^{-1} \mu_{i-} \frac{1}{2}\mu_i^T \Sigma^{-1}\mu_i + log\pi_i \quad (4)$$

The drawback of LDA is that it is exceptionally susceptible to outliers. It is impossible to interconnect dependent variable and multicollinearity (linear combination of other variables) variables totally. One more shortcoming of LDA is that occasionally the value of $\delta_i(x)$ is below 0, and at other times it is greater than 1, which is unjustifiable. The strengths LDA state in this section makes it a suitable algorithm for solving software prediction problem.

Recursive Partitioning and Regression Trees (RPART)

The RPART is an advanced implement many concepts explained in the Classification and Regression Trees (CART) book authored by Breiman et al. [27]. Recursive partitioning is a statistical technique that applies multivariable statistics to monitor more than one outcome variable concurrently [28]. Recursive partitioning helps create a decision tree that endeavours to accurately categorize the population's components by partitioning it into sub-groups using various bivalent variables that are autonomous. The reason for calling the process recursive is that it divides any sub-group successively on an unspecified number of occasions until the stopping criterion is satisfied and the partitioning process ends.

The first step to building a regression tree is first to use recursive partitioning to cultivate a big tree on the training dataset. The process only ends when each leaf node has less than the least number of observations. Recursive partitioning is a greedy and top-down algorithm. It reduces the Residual Sum of Squares (RSS). The disparity between the true and the measured value is determined by RSS. A linear regression environment also uses RSS [29].

Some of the strengths of RPART include its ability to estimate relationships between predictors and outcomes quickly, even when the relationship is sophisticated. It offers a straightforward and instinctive technique for classifying objects. It intends to recognize the interactions of two or more factors when their collective influence is more than the total influences observed when each factor alone is used. It is capable of detecting nonlinear relationships with the final stage of the process. It presents a simple way of building identical" ongoing process of assigning patients risk status. It forms part of previous probabilities and penalties, for instance, of wrongly assigning its variable to another class during the process of making choices. Lastly, their generated results are simple to understand. However, RPART has certain limitations as it does not work well for continuous variables, and it can result in data overfit. It can also omit other predictive factors throughout the subsequent phase of the process of picking the right candidate solution. This can result into the problem of "multiple testing." It is likely to intensify the setback of "over-training." Finally, it is likely not to represent the entire predictive adeptness of a continuous factor. The above advantages make RPART a good algorithm for solving software prediction problem.

k-Nearest Neighbour (kNN)

The kNN is a classic example of promising classification algorithms used for solving classification problem built on the concept of nearest learning examples in the feature space. The kNN algorithm is known as 'lazy learning' algorithm because it generalizes the data after a query is made. Every computation involved in the classification process takes place after the classification process. No definite learning or model creation carried out throughout the training stage, even though there is a need for a training dataset. The kNN uses the dataset to populate a sample of the search space with instances of a known class exclusively. This algorithm is classified as a 'lazy learning' algorithm for this reason [30]. The kNN algorithm being a lazy learner, does not have a training stage; even when it does, the training stage is within a relatively short time completed. However, the testing stage is expensive with respect to time and memory [31].

Some of the kNN algorithm's strengths are that it has proved to have the capacity to surmount many of the challenges confronting other existing algorithms. Secondly, only a few parameters (k and distance metric) need to be fine-tuned to get better classification accuracy. The effect of noise on the prediction decreases drastically by choosing large values of k. This, however, makes borderlines between the classes not to be so discrete. Thirdly, kNN surmounts the scalability problem that is prevalent among some existing machine learning algorithms, such as decision trees. This is because of its ability to handle training data that are too big to fit into memory effectively. Fourthly, the implementation is easy. This is because a simple Euclidean distance is used to determine the sameness between training subset data and the test subset data when the preceding information about function showing all the possible data values is nonexistent. These advantages, therefore, mean that kNN a beneficial algorithm for solving software prediction problems.

Generalized Linear Model with Elastic Net Regularization (GLMNet)

Generalized linear models (GLMs) are popular algorithms for solving regression and classification problems. This algorithm can easily find global optimum and knows when it reaches one. In addition, they are simple and economical to fit. GLMs are very easy to understand. This is due to their clearly outlined noise distributions and the absence of a direct relationship between an independent and a dependent variable at every point of a given set [32].

Friedman et al. proposed GLMNet. The algorithm is a package that suits a general linear model through penalized highest probability [33]. It provides a means to circumvent overfitting by penalizing high-valued regression coefficients for the regularization parameter lambda. Some of the strengths of the algorithm include high speed and the ability to use thinly scattered data in the input matrix efficiently [33]. The GLMNet algorithms episodically iterate through the directions, one at a time, reducing the objective function with regard to each coordinate direction at a time. It consecutively improves the objective function over each parameter with others stationary, and cycles are continually pending when it has a close to the local or global minimum. The algorithm also applies powerful rules to restrain the active set effectively. GLMNet has extremely effective updates and methods that leverage prior computation to dramatically reduce the time required to train a model (warm starts) and active-set convergence. GLMNet can use formal rules to compute how to do predicting quickly. It can effectively solve problems where most of the elements in the input-matrix are zero, in addition to solving coefficients that specified that the value must be between two given values [33, 34]. The combined strengths of GLMNet algorithm makes it an alternative solution for solving software defect prediction problem.

Random Forest (RF)

Breiman and Cutler [35] first developed the RF. RF is a meta estimator that fits many classifying decision trees on several sub-samples of the dataset and employs averaging to enhance its predictive accuracy and regulate over-fitting [36]. It is a supervised learning algorithm. RF is simple, diversified, and can be implemented easily. It does not need a considerable amount of resources like time, processing power, or memory before producing optimal solutions to any problem. RF has proven to perform excellently in solving several real-world problems [36, 37]. It is a good example of ensemble machine learning and regression method suitable for finding solutions to classification and prediction jobs [38]. RF can produce optimal results most of the time, even without tuning any value of the used parameter during the learning process.

The benefits of using Random forests include minimized prediction error and improved f-scores compared to a number of other machines learning algorithms. Besides, its overall performance is better than that of Naïve Bayes and SVMs. Unlike SVM and Neural Networks, RF has a shorter training time. RF has a higher classification accuracy than many of the popular machine learning techniques. RFs can efficiently process unlabeled data, thereby making it a very suitable method for finding cohesion between data elements that are otherwise unlabeled and unclassified. RF is easy, and it uses the small number of parameters compared to the number of observations. RF allows the user to grow the trees they want very fast [38]. The RF discussed strengths make it a very suitable algorithm for proffering solution to software defect prediction problem.

Proposed Method

The ability to predict defects in software accurately is very critical in the software engineering discipline. Section 2 of this paper discussed the applications of machine learning techniques to software defect prediction. Conversely, these approaches attempted to solve issues related to software faults. Despite these efforts, attaining high prediction accuracy and improved performance of the entire system still poses an uphill task for software engineers and scientists. To solve this problem, this paper proposed an ensemble machine learning technique for predicting software defects. Stacking is a method to ensemble various learning algorithms, where a meta-level algorithm is trained to make a final prediction utilizing the outputs of based-level algorithms as features. It is a non-generative ensemble algorithm. In this research, three (3) machine-learning classifiers discussed in the previous sub-section (LDA, kNN, and GLMNet) were used as base learners, and Random Forest (RF) serves as the top layer of our proposed model. Because of software architecture's complexity, we combine all the optimal results produced by LDA, kNN, and GLMNet after they were trained using RF. The algorithms have two-layered architecture. All algorithms at the base layer, which include LDA, KNN, and GLMNet, were trained on the given training set, and after that, a combiner algorithm at the top layer such as RF is trained to utilize predictions of the first level as inputs. This technique helps in improving the accuracy of the learners. The combiner algorithm is named as Meta learner. Rather than the output from the base classifier, the probability predictions is passed as input to the Meta layer. Figure 1 depicts the structural design of the ensemble machine learning technique.

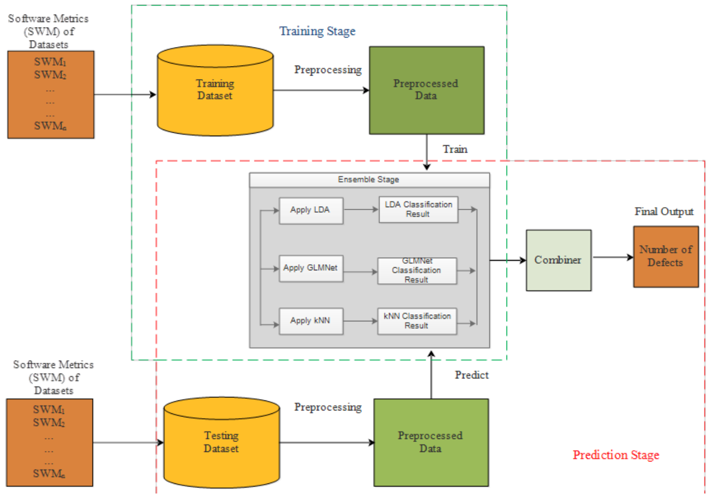


Figure 1: Architecture of proposed Ensemble Machine Learning Model

Presented in Figure 1 is the architecture of the proposed machine learning software defect prediction. The diagram shows that the proposed technique comprises of two steps: the training phase and the prediction phase using the trained model.

a. Preprocessing phase - Here, the NASA Metrics Data Program (MDP) for NASA PROMISE datasets containing software metrics was used for our experiments. The preprocessing operations such as data normalization and data transformation take place prior to training the ensemble machine-learning model. Preprocessing operation involves correcting missing values and outliers and selecting the input model's appropriate feature variables. Afterwards, the training of the proposed model to predict the flaws in software takes place.
b. Training phase - LDA, kNN, and GLMNet prediction models were trained, and each distinct prediction result and the actual result were accepted as a new training data set for secondary learning.
c. Ensemble phase – As represented in Figure 1, the set of models are trained in parallel, and their results are merged subsequently to produce the final output. Since the key

causes of error in learning models are a result of noise, bias, and variance. The ensemble phase is very important due to its ability to improve the prediction accuracy of learners. It consists of two-layered architecture, such as the base layer and top layer. The base layer consists of LDA, KNN, and GLMNet, while the top layer is RF. The base layer was trained on the given training set and after that, a combined algorithm at the top layer is trained utilizing the top layer's prediction. The output of all the models used for the ensemble learning is fused.

d. Prediction and Evaluation phase - In this phase, preprocessing operation is conducted to obtain the prediction stage's software metrics. The modeled data serves as input to the trained ensemble machine-learning model to predict the software's defects. The output is the aggregated classification result of all the machine-learning models (LDA, kNN, and GLMNet) used for the final prediction. Evaluation of prediction results happens at this stage also.

Depicted in Figure 2 is the flowchart of the ensemble model.

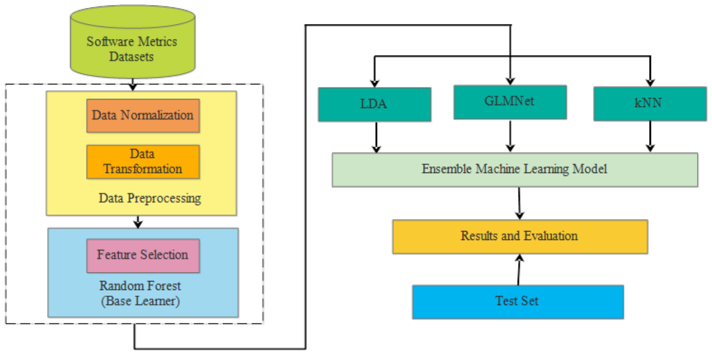


Figure 2: Flowchart of proposed Ensemble Machine Learning Model

This paper implements an ensemble model approach. The algorithm of the multi-model ensemble prediction model are as stated below:

Algorithm 1: Ensemble Prediction Model

Step 1: Divide the training dataset into train_train and train_valid using the technique of 5-fold cross-validation [37].

Step 2: The primary model, train five RF models with the train_train, use the grid search method to optimize the RF model's hyper-parameters and predict the train_valid and test data sets respectively to obtain the train_valid_pred and test_pred results. The five train_valid_preds produced by the 5-fold cross-validation are concatenated into rf_train_pred, and the five tests_preds are averaged to generate rf_test_pred.

Step 3: Repeat step 2 using LDA, GLMNet, and kNN to obtain lda_train_pred, lda_test_pred, glm_train_pred, glm_test_pred, knn_train_pred, and knn_test_pred produced by LDA, GLMNet, and kNN models.

Step 4: The auxiliary model, The LDA algorithm, was chosen for secondary learning to complete the ensemble of multiple primary models. The prediction values rf_train_pred, lda_train_pred, glm_train_pred, and knn_train_pred generated by each of the prediction models are used as input the corresponding actual value training_y is used as an output training kNN model.

Step 5: The rf_test_pred, lda_test_pred, glm_test_pred, and knn_test_pred produced by the primary model are employed as input features. Using the ensemble model based on kNN to predict them, the ultimate prediction results are attained.

Results and Discussion

This section presents the results of our experiments using the proposed model on PROMISE datasets. The work is implemented and analyzed using RStudio Version 1.2.5033 simulation tool.

Dataset Description

The Metrics Data Program (MDP) dataset used for the experiments was obtained from NASA PROMISE open-source datasets repository. Four datasets from the PROMISE repository named CM1, JM1, PC3, and KC3 with different attributes were used in this work. Table 1 contains different parameter descriptions of the datasets under consideration where several instances, number of defective modules, percentage defect, and number of metrics are represented

Table 1: PROMISE software defect prediction dataset details

Name of Dataset	Number of Instances	Defect	Defect (%)	Number of Metrics
CM1	344	42	12.21	37
JM1	9593	1759	18.34	21
PC3	1125	140	12.44	37
KC3	200	36	18.00	39

Table 2 contains the metrics and descriptions of the datasets used for our experiments. We evaluated the performance of our technique through these datasets and compared the performance with other machine learning techniques.

Table 2: Description of metrics in NASA PROMISE software defect datasets

Metrics	Description
LOC	Sum of line in the module
iv(g)	Design complexity of each module
ev(g)	Essential complexity of each module
N	Sum of operators and operands existing in the module
v(g)	Cyclomatic complexity of each module
D	Difficulties in each module
B	Effort approximation
L	Program size for each module
V	Volume of each module
I	Intelligence content
E	Error approximation
Locomment	Line of comments in each module
Loblank	Sum of blank lines in each module
uniq_op	Sum of unique operators
uniq_opnd	Sum of unique operand
T	Time determinist
Branchcount	Sum of branch in the software module
total_op	Sum of operators
total_opnd	Sum of operators
Locodeandcomment	Sum of line of code and comments
Defects	Details on whether there is existence of defect or not

Experimental Setup

Datasets used for many software projects are usually made available to the public so that researchers can make use of them for experimental and research purposes. One of the factors that can likely influence the prediction accuracy of a model is the ability of a particular dataset to perform an intended purpose. The design process that precedes the creation of a model and choices made influences a model's ability to predict defective parts. The potency of the predictive outcomes might not be the same at all times. This is because the performance evaluation measures used usually have diverse fundamental hypothetical basics. This explains why various models are likely to have a distinct score from various measurement standards. Some measurable qualities can change during a scientific experiment. These qualities are critical numerical quantities that must be considered in the overall strategy for experiments.

Performance Metrics

The proposed model was evaluated using the following performance metrics Mean Squared Error (MSE), Root Mean Square Error (RMSE), and Mean Absolute Scaled Error (MASE).

Root Mean Square Error (RMSE)

RMSE is the standard deviation of prediction errors in a test. The mathematical formula for RMSE is:

$$\sqrt{\frac{1}{n}\sum_{n=1}^{n}(y_t^n - \hat{y}_t^n)^2} \qquad (5)$$

Mean Squared Error (MSE)

MSE is the average squared difference between the estimated values and the actual value in the test. The mathematical formula for MSE is:

$$\frac{1}{n}\sum_{n=1}^{n}(y_t^n - \hat{y}_t^n)^2 \qquad (6)$$

Where N represents the number of data points, y_t is the value returned by the model, and ŷ is the actual value for data point *t*.

Mean Absolute Scaled Error (MASE)

MASE is a measure of the accuracy of predictions. MASE is a scale-free error metric that gives each error as a percentage in comparison to a standard mean error. MASE is defined as follows:

$$\frac{1}{n}\sum_{n=1}^{n}\frac{|y_t^n - \hat{y}_t^n|}{\frac{1}{n-m}\sum_{n=m+1}^{n}|y_t^n - y_{t-m}^n|} \qquad (7)$$

Where y_t^n is the target value, $\hat{y}_t^n$ is predicted values and m is the seasonal period of y_t^n

Simulation

In this section, we present the results of our simulations. The proposed model was used to classify and predict the number of software defects in the dataset. The dataset used for the work was divided into a training set (80%) and test set (20%), respectively. The dataset was trained using an RF classifier. Different test cases for each of the datasets were used to conduct the experiments.

Table 3: Performance evaluation of proposed ensemble model using CM1 dataset

Model	MSE	RMSE	MASE	Accuracy
RF	0.1384615	0.3721042	1.107692	0.8615
RPART	0.200000	0.4472136	0.762500	0.8000
kNN	0.1230769	0.3508232	0.4692308	0.8769
GLMNet	0.1230769	0.3508232	0.4692308	0.8769
LDA	0.1692308	0.4113767	0.6451923	0.8308
Ensemble	0.1230769	0.3508232	0.4692308	0.8769

Experimental results illustrated that the performance of our technique is as good as that of kNN and GLMNet. The three techniques produced an accuracy of 87.69% for the CM1 dataset. The RPART, with accuracy of 80.00% is the worst-performing model among all the methods compared. The three of them also have the lowest MSE, RMSE, and MASE. The ROC curve analysis for the dataset was presented to authenticate the proposed model's performance for CM1 dataset. This analysis is depicted in Figure 3. It represents the ROC curve for true positive rate and false-positive rate.

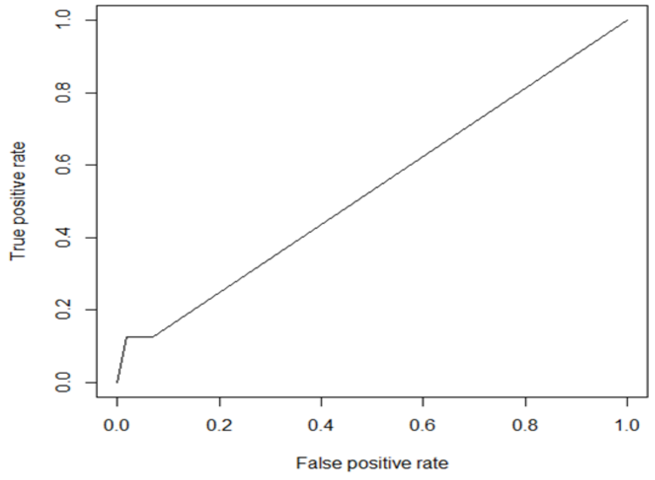
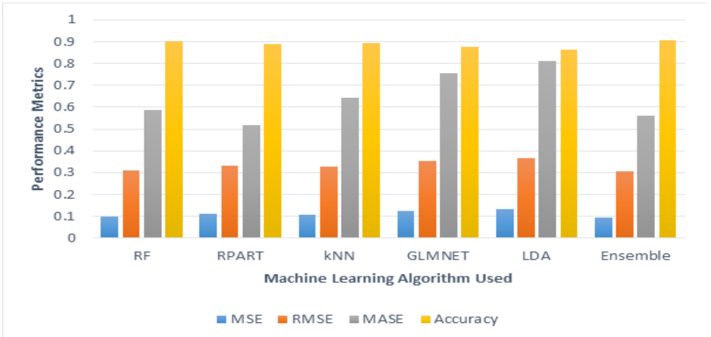


Figure 3: ROC curve analysis for CM1 dataset

A bar chart, which compares the proposed model with other existing models, is presented in Figure 4.



Test Case 2: JM1 dataset

The JM1 dataset has 9535 instances, 1759 defects, while the percentage of defective modules is 18.35%. Experiments were performed on the JM1 dataset under the same simulation environment as that of CM1. The comparative study was carried out as depicted in table 4 and in figures 5 and 6.

Table 4: Performance evaluation using JM1 dataset

Algorithm	MSE	RMSE	MASE	Accuracy
RF	0.1947301	0.4412823	0.6105476	0.8053
RPART	0.2037275	0.4513618	0.6387578	0.7963
KNN	0.2107969	0.4591263	0.6609229	0.7892
GLMNet	0.1960154	0.4427363	0.6145777	0.8040
LDA	0.1960154	0.4427363	0.6145777	0.8040
Ensemble	0.188946	0.4346792	0.5924126	0.8111

Simulation results indicated that the Ensemble technique's performance is superior to that of the other techniques under consideration. The Ensemble method produced an accuracy of 81.11% for JM1 dataset. The kNN algorithm produced the worst performance among all the methods compared with the accuracy of 78.92%, MSE value of 0.2107969, RMSE (0.4591263), and MASE (0.6609229). The ROC curve for the true positive rate and the false positive rate is presented in figure 5.

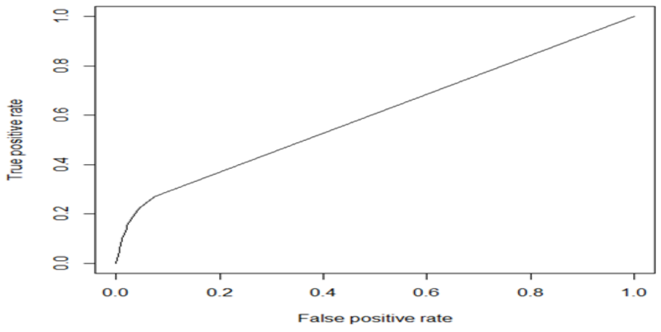


Figure 5: ROC curve analysis for JM1 dataset

Figure 6 depicts a comparison of the proposed model with other models.

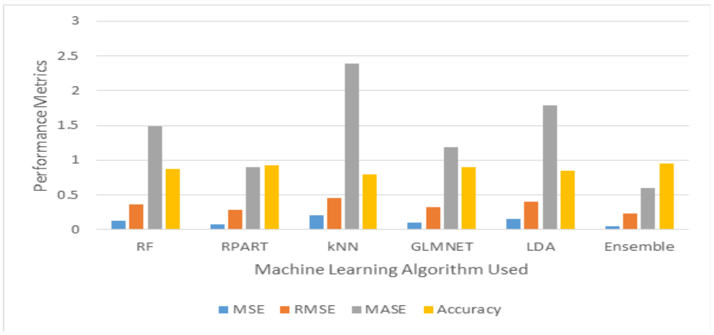


Figure 6: Performance Comparison of Models for JM1 dataset

Test Case 3: PC3 dataset

The PC3 dataset was used for our experiment in this case. The dataset contains 1125 instances, 140 defects, and a percentage defect of 37%. The simulation was carried out using the PC3 dataset under the same experimental condition. Table 5 and Figures 7 and 8 illustrate the novel model's performance evaluation against other techniques.

Table 5: Performance evaluation using PC3 dataset

Algorithm	MSE	RMSE	MASE	Accuracy
RF	0.09767442	0.3125291	0.5888372	0.9023
RPART	0.1116279	0.3341076	0.5193124	0.8884
KNN	0.1069767	0.327073	0.6449169	0.8930
GLMNet	0.1255814	0.3543747	0.7570764	0.8744
LDA	0.1348837	0.3672652	0.8131561	0.8651
Ensemble	0.09302326	0.3049971	0.5607973	0.9070

For the PC3 dataset, simulation results indicated that the Ensemble technique outperformed other methods under consideration. The Ensemble method generated an accuracy of 90.70% for the PC3 dataset. RF closely follows this with an accuracy of 90.23%. The LDA algorithm produced the worst performance among all the methods compared with an accuracy of 86.51%. The ROC curve for the true positive rate and the false positive rate is presented in Figure 7.

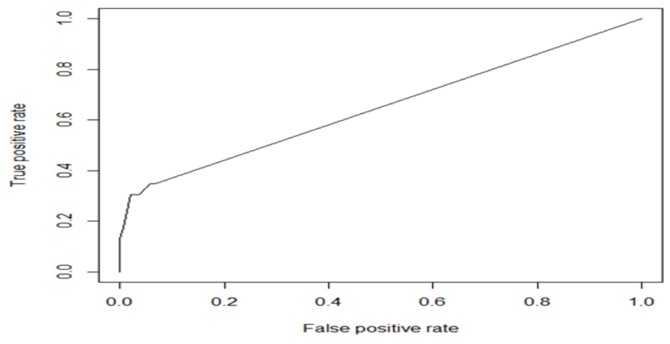


Figure 7: ROC curve analysis for PC3 dataset

Depicted in figure 8 is a comparison of the proposed model with other models.

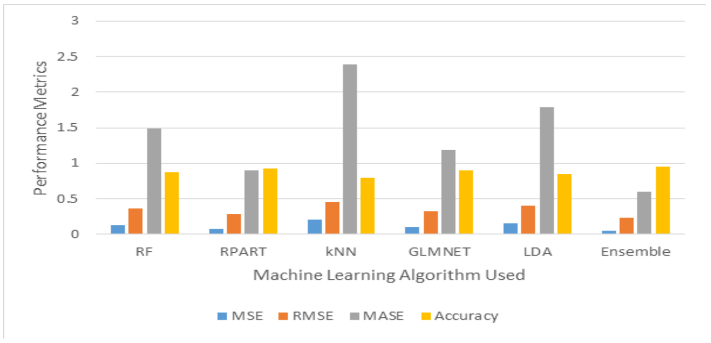


Figure 8: Performance Comparison of Models for PC3 dataset

Test Case 4: KC3 dataset

The KC3 dataset was used for our experiment this time around. There are 200 instances in the dataset having 18.00% defect. The experiment was conducted under the same condition as the previous tests. Simulation results of our experiments are represented in table 6 and Figures 9 and 10.

Table 6: Performance evaluation using KC3 dataset

Algorithm	MSE	RMSE	MASE	Accuracy
RF	0.1315789	0.3627381	1.491228	0.8684
RPART	0.07894737	0.2809757	0.8947368	0.9211
KNN	0.2105263	0.4588315	2.385965	0.7895
GLMNet	0.1052632	0.3244428	1.192982	0.8947
LDA	0.1578947	0.3973597	1.789474	0.8421
Ensemble	0.05263158	0.2294157	0.5964912	0.9474

For the KC3 dataset, results indicate that the novel Ensemble technique outperformed other approaches considered in this paper. The Ensemble method produced an accuracy of 94.74% for the KC3 dataset. This is a significant increase compared to that of other techniques compared. This is followed by RPART that has an accuracy of 92.11%. The worst performance was produced by the kNN algorithm with an accuracy of 78.95%. The ROC curve for the true positive rate and the false positive rate is depicted in Figure 9.

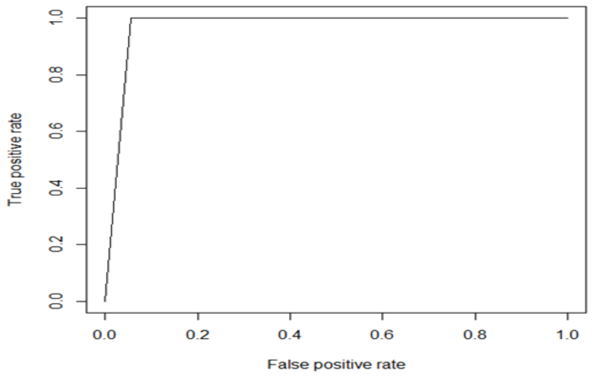


Figure 9: ROC curve analysis for KC3 dataset

Presented in Figure 10 is a comparison of the proposed Ensemble model with other models.

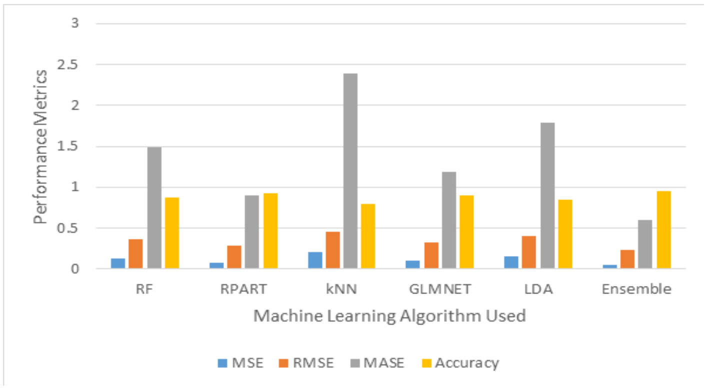


Figure 10: Performance Comparison of Models for KC3 dataset

The overall accuracy of the models used in this work is presented in table 7.

Table 7: Average Overall Prediction Accuracy of the Models on each Dataset

Data Set	RF	RPART	KNN	GLM Net	LDA	ENSE MBLE
CM1	0.8615	0.8000	0.8769	0.8769	0.8308	0.8769
JM1	0.8053	0.7963	0.7892	0.8040	0.8041	0.8111
PC3	0.9023	0.8884	0.8930	0.8744	0.8651	0.9070
KC3	0.8684	0.9211	0.7895	0.8947	0.8421	0.9474
Overall Accuracy	0.8594	0.8515	0.8372	0.8625	0.8355	0.8856

Our findings show that the Ensemble approach increases defect prediction accuracy using fewer attributes. This is unlike the other existing techniques. The proposed model gives equal and, in some cases, better prediction accuracy even when very few attributes are used. The performance of the proposed model was tested on CM1, JM1, PC3, and KC3 datasets. Comparative analysis of Area Under Curve (AUC) of the proposed model with previous work is presented in table 8 [39-44].

Table 8: Comparative analysis AUC of the proposed model with other work

Algorithm	PC3	JM1	CM1	KC3
NN [5]	0.89	0.81	0.79	0.79
KNN [38]	0.77	0.69	0.72	0.70
NB [38]	0.81	0.69	0.75	0.76
SVM [39]	0.77	0.72	0.75	0.76
L-SVM [40]	0.84	0.73	0.75	0.76
LS-SVM [41]	0.83	0.74	0.77	0.77
LDA [42]	0.82	0.73	0.77	0.78
Ensemble	0.95	0.89	0.84	0.87

Conclusion

A novel multi-model Ensemble machine-learning model has been proposed in this paper. The results show that the model can handle PROMISE datasets that are known for their noisy attributes and high dimensions effectively. The proposed model's performance was compared with that of seven other machine-learning models in terms of Area under the curve (AUC). The Ensemble model's experimental results show promising results in predicting defects in a software system for the four datasets used. The advantage of the proposed ensemble model apart from high classification accuracy, is that it decreases the model's time cost in parameter optimization. Future research work will be focused on benchmarking this proposed model with ensemble deep learning. Finally, it is expected that this paper has contributed and impacted more knowledge in the field of software defect prediction. It is hoped that this will aid software developers to detect flaws in software systems easily and with higher accuracy. This has the ability to enhance the development of a high-quality software package. According to the generated results, some thought-provoking observations can be deduced from this work which are listed below:

1. The most pragmatic upshot of this study was that the Ensemble machine-learning technique has better prediction accuracy for software defect prediction.
2. Elimination of the occurrence of high intercorrelations among two or more independent variables in a multiple regression model enhances the machine-learning model's performance.
3. Feature extraction has significantly enhanced the prediction accuracy of the performance of the prediction model.
4. The MSE of the proposed model is significantly reduced compared to other models because the Ensemble technique effectively handles error in the learning model caused by noise, bias, and variance.

References

1. Li L, Lessmann S, Baesens B (2019) Evaluating software defect prediction performance: an updated benchmarking study, ArXiv preprint arXiv: 1901.01726.
2. Kitchenham B, Charters S (2007) Guidelines for Performing Systematic Literature Review in Software Engineering, Technical Report. EBSE-2007-001, Keele University and Durham University: Staffordshire, UK.
3. Zou H, Hastie T, Tibshirani R (2006) Sparse principal component analysis, Journal of computational and graphical statistics 15: 265-286.
4. Shepperd M, Song Q, Sun Z, Mair C (2013) Data quality: Some comments on the nasa software defect datasets. IEEE Transactions on Software Engineering 39: 1208-1215.
5. Jayanthi R, Florence L (2019) Software defect prediction techniques using metrics based on neural network classifier. Cluster Computing 22: 77-88.
6. Chauhan NS, Saxena A (2013) A green software development life cycle for cloud computing. IT Professional 15: 28-34.
7. Sandhu PS, Goel R, Brar AS, Kaur J, Anand S (2010) A model for early prediction of faults in software systems. In 2010 the 2nd International Conference on Computer and Automation Engineering (ICCAE), IEEE 4: 281-285.
8. El Emam K, Melo W, Machado JC (2001) The prediction of faulty classes using object-oriented design metrics. Journal of systems and software 56: 63-75.

9. Kuncheva LI, Skurichina M, Duin RP (2002) An experimental study on diversity for bagging and boosting with linear classifiers. Information fusion 3: 245-258.
10. Aljamaan HI, Elish MO (2009) An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software. In 2009 IEEE Symposium on Computational Intelligence and Data Mining, IEEE 2009: 187-194.
11. Okutan A, Yıldız OT (2014) Software defect prediction using Bayesian networks. Empirical Software Engineering 19: 154-181.
12. Shan C, Chen B, Hu C, Xue J, Li N (2014) Software defect prediction model based on LLE and SVM. In: Proceedings of the Communications Security Conference (CSC '14) 2014: 1-5.
13. Koru AG, Liu H (2005) Building effective defect-prediction models in practice. IEEE software 22: 23-29.
14. Fu W, Menzies T (2017) Easy over hard: A case study on deep learning. In Proceedings of the 2017 11th joint meeting on foundations of software engineering 2017: 49-60.
15. Menzies T, Majumder S, Balaji N, Brey K, Fu W (2018) 500+ Times Faster than Deep Learning:(A Case Study Exploring Faster Methods for Text Mining StackOverflow). In2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR) 2018: 554-563 IEEE.
16. Hoque N, Singh M, Bhattacharyya DK (2018) EFS-MI: An Ensemble feature selection method for classification. Complex & Intelligent Systems 4: 105-118.
17. Jiarpakdee J, Tantithamthavorn C, Treude C (2018) Autospearman: Automatically mitigating correlated metrics for interpreting defect models. arXiv preprint arXiv:1806.09791.
18. Ghosh S, Rana A, Kansal V (2018) A nonlinear manifold detection-based model for software defect prediction. Procedia computer science 132: 581-594.
19. Majd A, Vahidi-Asl M, Khalilian A, Poorsarvi-Tehrani P, Haghighi H (2020) SLDeep: Statement-level software defect prediction using deep-learning model on static code features. Expert Systems with Applications. 147: 113156.
20. Qiao L, Li X, Umer Q, Guo P (2020) Deep learning-based software defect prediction. Neurocomputing 385: 100-110.
21. Sun Z, Zhang J, Sun H, Zhu X (2020) Collaborative filtering-based recommendation of sampling methods for software defect prediction. Applied Soft Computing 90: 106163.
22. Wei H, Hu C, Chen S, Xue Y, Zhang Q (2019) Establishing a software defect prediction model via effective dimension reduction. Information Sciences 477: 399-409.
23. Zhao L, Shang Z, Zhao L, Zhang T, Tang YY (2019) Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. Neurocomputing. 352: 64-74.
24. Shao Y, Liu B, Wang S, Li G (2020) Software defect prediction based on correlation weighted class association rule mining. Knowledge-Based Systems. 105742.
25. Duda RO, Hart PE, Stork DG (2001) Pattern classification 2nd ed. John Willey & Sons Inc.
26. Discriminant Analysis (2021) STAT 508 Applied Data Mining and Statistical Learning. PennState Eberly College of Science Available at https://online.stat.psu.edu/stat508/lesson/9/9.2
27. Breiman L, Friedman J, Olshen R, Stone C (2017) Classification and regression trees. Wadsworth Int. Group 37: 237-251.
28. Breiman L, Friedman J, Stone CJ, Olshen RA (1984) Classification and regression trees. CRC press 1984.
29. James L (2018) Decision Trees in R. Retrieved on May 15th 2020 from https://www.datacamp.com/community/tutorials/decision-trees-R
30. Saini I, Singh D, Khosla A (2013) QRS detection using K-Nearest Neighbor algorithm (KNN) and evaluation on standard ECG databases. Journal of advanced research 4: 331-344.
31. Thirumuruganathan S (2010) A detailed introduction to K-nearest neighbor (KNN) algorithm.
32. Jas M, Achakulvisut T, Idrizović A, Acuna D, Antalek M, et al. (2020) Pyglmnet: Python implementation of elastic-net regularized generalized linear models. Journal of Open-Source Software 5: 47. https://doi.org/10.21105/joss.01959
33. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.
34. Benjamin AS, Fernandes HL, Tomlinson T, Ramkumar P, VerSteeg C, et al. (2017) Modern machine learning outperforms GLMs at predicting spikes. BioRxiv 2017: 111450.
35. Dada EG, Joseph SB (2018) Random Forests Machine Learning Technique for Email Spam Filtering. Faculty's Seminar Series, University of Maiduguri, Nigeria 9: 29-36.
36. Dada EG, Bassi JS, Chiroma H, Abdulhamid SM, Adetunmbi AO, et al. (2019) Machine learning for email spam filtering: review, approaches and open research problems. Heliyon 5: 01802.
37. Breiman L, Cutler A (2007) Random forests-classification description, Department of Statistics Homepage. http://www.stat.berkeley.edu/~breiman/RandomForests/cchome.htm.
38. Donges N (2020) A complete guide to the random forest algorithm. Retrieved from https://builtin.com/data-science/random-forest-algorithm.
39. Yongdong F (2013) A review of cross validation methods in model selection [D]. Shanxi University.
40. Andersson C (2007) A replicated empirical study of a selection method for software reliability growth models. Empir. Softw. Eng 12:161-182.
41. Andersson C, Runeson P (2007) A replicated quantitative analysis of fault distributions in complex software systems. IEEETrans. Softw. Eng 33: 273-286.
42. Mangasarian OL, Musicant DR (2001) Lagrangian support vector machines. J. Mach. Learn. Res 1: 161-177.
43. Suykens JAK, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Process. Lett 9: 293-300.
44. Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEETrans. Softw. En 34: 485-496.