Journal of Data Acquisition and Processing
ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.7699787

# WABE: A MODIFIED VERSION OF ADA-BOOSTING APPROACH TO IMPROVE SOFTWARE QUALITY AND FOR EFFECTIVE SOFTWARE DEFECT PREDICTION

**K. Eswar Rao [1*], P. Ramkishor [2], P. Annan Naidu [3], S. Jayawardhana Rao [4] and T. Ravi Kumar [5]**

[1, 2, 3 ,4 ,5]*Department of Computer Science and Engineering, Aditya Institute of Technology and Management, Tekkali-532201, India.*

** Corresponding Author: eswarkoppala@gmail.com*

**Abstract**

*Boosting approaches have recently been developed in the area of software defect prediction (SDP) by combining various base classifiers. The use of boosting has sometimes proven to be more accurate than using single base classifier in some experiments. Massive research from the industry and experts has started working on this area, even though most of the predictive methods are still in their infancy and require further research.* To determine whether boosting models are superior to employing single classification models to produce high-quality software, more research is necessary. *A Weighted Adaptive Boosting Ensemble (WABE) approach has been proposed for software defect prediction with software quality. The misclassification costs are incorporated into the weight-update rule of the boosting method, which causes the proposed algorithms to increase the weights of the samples linked to misclassified defect-prone modules. The proposed model showed that a high area under the receiver operating curve (ROC-AUC) and the learning ratio of the new model look promising, and various performance metrics are compared with other state-of-the-art machine learning-based methods to prove its superiority. This research confirms that decision tree classifiers must be carefully chosen as estimators to accurately identify the defective parts for an effective quality product.*

**Keywords.** Ensemble Learning, Software defect, Software quality, Software Engineering, Weighted Adaptive Boosting.

## 1. Introduction

The enormous number of applications produced makes software quality a persistent issue that produces unfortunate results for business apps [1]. A constant issue throughout the whole software life cycle is how to increase software quality and decrease faults. Software defects have a massive impact on the quality of the product, as well as increasing expenditures for software maintenance. Most of the time, researchers are unable to provide a justification for any classification model's prediction, and further research is needed to help designers choose an explanation for a machine learning model that predicts a software defect [2]. The fact that no specific SDP technique outperforms the other methods on substantially dissimilar datasets was discovered, despite some differences in the research. In order to determine which projected model is the best for SDP, researchers used additional evaluation metrics [3]. With regard to modules from nine NASA programmes, we developed prediction models that took into account

various combinations of McCabe and Halstead traits (see Table 3). Researchers used twelve NASA datasets to compare the performances of several machine learning algorithms with respect to the metrics of accuracy, F-score, Precision, Recall, MCC, and ROC. In order to limit testing to the modules that are most likely to have defects, the technique of SDP can be utilized as a quality assurance activity to identify those modules. This method enables the development of more expensive but higher-quality software [4,5]. In order to address the problem of class imbalance, two classifiers—the asymmetric kernel partial least squares classifier (AKPLSC) and the asymmetric kernel principal component analysis classifier (AKPCAC)—are proposed in this study. Class imbalance may significantly reduce the performance of defect prediction. [6], The classification percentage Due to the size of these data sets and the software's sensitivity, using a hybrid technique that combines PCA, RF, NB, and SVM to provide the accurate predictions is essential for early detection [7]. The chief drawback of these statistical approaches was that they are established on the basis of the existent samples or patterns in the metrics of software and hence became unreliable for changing these patterns. To address these drawbacks, since, from the last few decades, ML (Machine Learning) approaches were already developed and being developed day-by-day. Existent approaches were passive in expressing the reliable outcomes of the software and it requires physical perpetuation support for adapting the emerged data sets in software industries. ML has the capability of cracking the conflict of physical perpetuation however there is a need for alterations in what manner the software companies of software industries are employed with SDPs.

Day-by-day ML algorithms are being used for SDP. Benchmark algorithms such as Bagging, Support Vector Machines (SVM), Navie Bayes (NB), Artificial Neural Networks (ANN) and Decision Trees (DT) have been already employed to resolve this SDP problem [8,9] effectively with better accuracy. DTs (Decision Trees) have the capability of finding out the faults of software. The main advantage with DTs is that they were reliable to noisy environmental data and the omitted values can be indulged. However, there is a limitation that they were exposed to over-fitting. Another standard ML algorithm was SVM. They have also been used for solving SDP problems. The benefit with SVMs is that they are memory effective and they can accept KFs (Kernal Functions) as DFs (Decision Functions). However, this SVM was limited to produce reduced performance under the condition if the features to be considered were higher than the samples taken. BNNs (Bayesian Belief Networks) were also used to resolve SDP. The main advantage of BBN is that it can produce probabilistic predictions and training data is needed to produce these preceding probabilities. However, BBN's have the limitation that expertise of domain for generating the network is desirable and they are computationally exclusive. ANNs (Artificial Neural Networks) have the capability for predicting the SDs. The key benefits of ANNs are that they have the capacity to learn non-linear as well as composite functions. They can also be reliable to faults in training data. However, there are some limitations that the training process is very slow, confluent and they are prone to over-fitting. Considering this into account, in this study, we propose a Adaboost defect prediction model as a classifier that is trained to recognise software modules that are subject to defects. Constructing a defect dataset, implementing a prediction approach or a machine learning classifier to the defect dataset, and evaluating the performance of defect prediction models constitute defect prediction modeling.

The remaining of this paper organized as follows. In section 2 summarizes the literature review with the discussion on the early developed methods for SDP. Next we present and discuss the proposed model in section 3 with the complete working of procedure. Then section 4 presents the experimental set up along with the discussion on the obtained results. Finally section 5 concludes the research with future scope.

## 2. Review of The Literature

Several methods were anticipated for cracking the conflict of SDP. The mainly identified methodologies were ML approaches. ML approaches were employed expansively with commended achievement in SDP for finding out the vulnerable elements on the basis of chronological errors as well as necessary factors. Many studies have been already employed for predicting the software defects which evident the significance of ML techniques for SDP complications. The study in [10] proposed a novel way of approach for solving the SDP problem using SPCFNN. PD, as well as PF, were considered as evaluation measures. A comparison has been made with the benchmark methods such as LSTM, DSNN, DBN, RNN, etc. Higher performance was found over the compared ones to overcome the conflict of SDP. A modern method has developed for SDP and named the method as PMOFES [11]. Complexity and count rates were examined as performance metrics and a comparison was made with standard techniques such as NB, LR, k-NN. Based on the proposed experimentations, the authors have claimed that PMOFES yields better accuracy for resolving SDP. To overcome JT (Just-in-Time) SDP problem, to identify more defective software updates with less code inspection by using improved supervising models CBS+(classify before sorting) [12]. The effectiveness of CBS+ utilizing three distinct assessment settings, including time-wise cross-validation, 10-times 10-fold cross-validation, and cross-project validation. CBS+ detects 15% to 26% more incorrect modifications while retaining a similar amount of context switching and early false alarms. The authors proved that supervised techniques have the significance to overcome JT-SDP. In order to resolve the severe problem of SDP, systematic structure and semantic data is more significant to a program, in [13] proposed DP-ARNN takes vector input from ASTs for mapping and embedding than clearly finds the syntax and semantic features. According to the experimental findings, DP-ARNN, on average, outperforms state-of-the-art approaches in terms of F1-measure improvement and AUC improvement. Class imbalance problem would yield non-diverse synthetic instances, as well as numerous unneeded noise instances; to overcome this, [14] has developed the KMFOS model, which would distribute a new faulty instance in the space of a defective dataset. In addition, compared to other state-of-the-art class-imbalance approaches, such as balance bagging classifier, RUS boost classifier, Instance Hardness Threshold, and cost-sensitive methods, our method is more accurate and efficient. In comparison to other sampling and class-imbalance approaches and for SDP, KMFOS effectively produces superior Recall and bal values, which enhances the effectiveness of prediction models. Class imbalance problem and parameter selection are Typical problems of SDP; nevertheless, [15] has created a synchronous solution of hybrid multi-objective cuckoo search under-sampled software defect prediction model based on SVM (HMOCS-US-SVM). False positive rate (pf), probability of detection (pd), and G-mean are utilized to assess the efficacy of the proposed method (HMOCS), which is employed to simultaneously choose non-defective samples and improve the SVM's parameters. The suggested strategy is effective in resolving the problem of software defect prediction when compared to the output of eight

prediction models. In general, the approaches perform better when compared to the standard benchmark SDP models. The authors [16] proposed a novel SDP model called SDAEsTSE for addressing the class imbalance problem, and they showed how to extract the DPs from conventional software metrics using learning phase and two-stage (TSE) phase. They also showed the efficacy of DPs, TSE, and SDAEsTSE, and they obtained the highest score. According to F-measure, AUC, and MCC, her performance is assessed. Typically, the conventional classifiers' performance has a ceiling of around 80% recall. The performance of RF, NB, RPart, and SVM classifiers on various datasets is compared individually by the authors [17] who also predict and analyse the level of prediction uncertainty and carry out a sensitivity analysis. To sum up, not all competitors performed equally on predictions. They come to the conclusion from research that classifiers without majority voting-based decision-making are likely to perform at their highest level when predicting defects. Researchers have started to test semi-supervised approaches to identify software faults in order to address the existing challenges. As compared to supervised learning, semi-supervised learning may fully exploit unlabeled examples to predict defects and result in higher classification results with few labelled samples [18]. Furthermore, 80% of software testing problems are discovered in 20% of the code, indicating that the majority of software defects are focused in a small number of software modules. As a result, there is a substantial "class imbalance"[19] in software defect history data, which makes it difficult to learn from and makes it difficult to estimate results. All of these researches are aimed at demonstrating the AdaBoost algorithm's effectiveness, particularly in binary classification issues. We also aim to present the success of the suggested model versus best known and recent state-of-the-art methods after modifying and repeating experiments on a real dataset, namely the Software Defect prediction problem (SDP). Moreover, **Table 1** shows the several ML methodologies utilized for cracking out the SDP complications.

**Table 1** Other ML methods for decoding SDP problems

| ***S.No*** | ***Method used*** | ***Data Set Used*** | ***Problem Name*** | ***Compared method*** | ***Performance Metrics used for the study*** | ***Ref.*** |
|---|---|---|---|---|---|---|
| 1 | KPWE | PROMISE & NASA | SDP | ELM, BP, SVM | AUC<br>F-Measure | [20] |
| 2 | - | PROMISE | SD Estimation | MLP, RBF, SVM, Bagging, RF, NB, Multinomial NB | Accuracy<br>Precision<br>Recall<br>F-Measure | [21] |
| 3 | SVM, k-NN, RF, LiR | NASA, ANT | SDP | - | Accuracy, AUC, RMSE, RR | [22] |
| 4 | L-RNN | NASA | SFP | NB, ANN, LR, k-NN, C4.5 | AUC, ROC, TP,TN, FP, FN | [23] |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | SSA-BPNN | KC1,JM1, PC3, PC4 | SFP | k-NN, SVM, NB, LDA | AUC, Confusion Matrix, Sensitivity, Specificity, Accuracy | [24] |
| 6 | ConFS, CFS | PROMISE, NASA, AEEEM | SDP | LR, RF, NB, J48, LMT, SC, KM, PAM, FCM, NG | FA, RBM, AE, AUC, IQR, | [25] |
| 7 | GTB | JM1 | SDP | DT, KNN, LDA, LR, MLP,NB, QDA, RF, SGD | F1,ROC,AUC, TPR, FPR | [26] |
| 8 | DPDF | JM1, MC1, etc., | SDP | NB,LR, SVM,RF | gc-Forest, AUC Z- score | [27] |
| 9 | Random Forest, XGB | ivy, camel, jedit-4.1 | CPDP | - | PCA, CPDP ,WPDP Class imbalance | [28] |
| 10 | Voting | New xalan-2.5 New xalan-2.6 | SDP | DT,RF,AdB, GB,NB,KNN, ANN | Precision, Recall F1-score | [29] |

## 3. Methodology

This work aims to show the performance analysis of various machine learning techniques including proposed model for SDP on JM1 dataset considered from PROMISE repository [30]. The dataset has 10885 instances and 22 no. of attributes. It does not have any missing attributes. **Table 2** represents more details about the dataset with all attribute name and information. The proposed WABE method consists of two phases: i) Designing of weighted AdaBoost model to learn from past software defect profile data ii) Prediction of software defect by using trained WABE model.

**Table 2.** JM1 dataset details

| Sl No | Attribute | Details | Attribute type |
|---|---|---|---|
| 1 | loc | McCabe's line count of code | Numeric |
| 2 | v(g) | McCabe "cyclomatic complexity" | Numeric |
| 3 | ev(g) | McCabe "essential complexity" | Numeric |
| 4 | iv(g) | McCabe "design complexity" | Numeric |
| 5 | n | Halstead total operators + operands | Numeric |
| 6 | v | Halstead "volume" | any (default) |
| 7 | l | Halstead "program length" | Numeric |
| 8 | d | Halstead "difficulty" | any (default) |
| 9 | i | Halstead "intelligence" | any (default) |
| 10 | e | Halstead "effort" | any (default) |
| 11 | b | Halstead | Numeric |
| 12 | t | Halstead's time estimator | any (default) |
| 13 | lOCode | Halstead's line count | Numeric |
| 14 | lOComment | Halstead's count of lines of comments | Numeric |
| 15 | lOBlank | Halstead's count of blank lines | Numeric |
| 16 | lOCodeAndComment | comments | Numeric |
| 17 | uniq_Op | unique operators | Numeric |
| 18 | uniq_Opnd | unique operands | Numeric |
| 19 | total_Op | total operators | Numeric |
| 20 | total_Opnd | total operands | Numeric |
| 21 | defects | reported defects | boolean (default) |

In this work, AdaBoost [31] has been used to boost the performance of decision trees for binary classification problems, i.e. software defect or not. Adaboost initially generates and allocates training and test subsets at random (Here the proposed model predicts the software defect from $N$ no. of decision trees constructed from weighted instances (software profile) in training data). It iteratively trains the model by picking the training set and gives higher weights to incorrectly classify observations, so that they will get a better probability of classification in the next iteration. Based on the decision tree classifier the algorithm assigns the weights to the classifier for each iteration. This entire procedure is repeated until there is no more improvement in prediction error or the appropriate number of estimators is formed. The prediction of software defects has been accomplished by calculating the weighted average of the forecast of the generated pool of decision trees. Figure 1 depicts a sample depiction of the proposed methods. As stated previously, the proposed algorithm utilizes the boosting strategy, and it is evident that a high learning rate is more probable. Gini, entropy, or both are utilised by a number of decision tree techniques. The Gini index and entropy are the measures that are used to determine how much information has been gained. Methods for decision trees employ this value to separate nodes. These two metrics represent the degree of noise and impurity at a node. A node with many classes, for instance, suggests impurity, whereas a node with only one

class exhibits purity. Furthermore, a decision tree is a graph-based solution to a problem that represents all feasible trial-by-decision solutions based on the provided conditions.

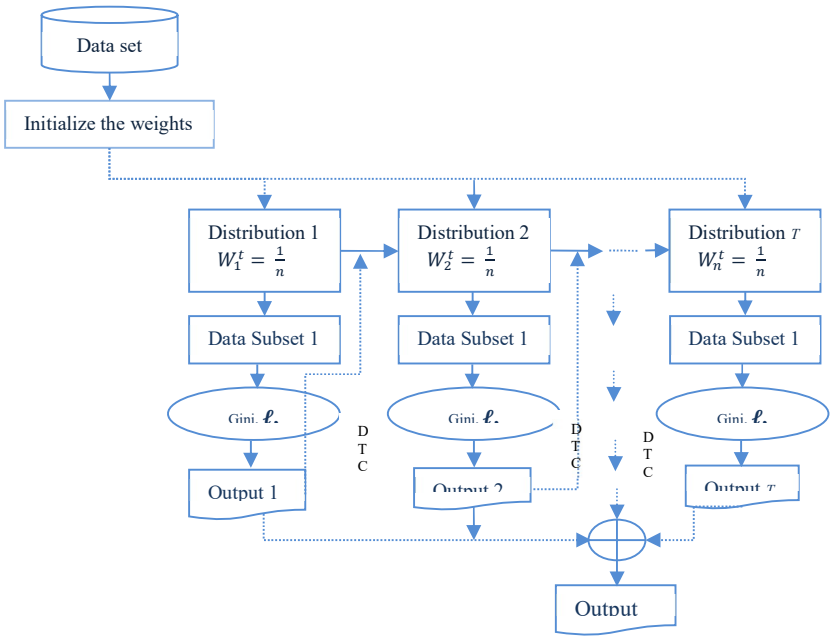


**Figure 1** Overview of the proposed WABE method.

In this proposed approach weights are updated with support of decision tree classifier followed by best split and employed as the estimator. Gini Index method is more popular and powerful in binary classification problems with short a period of time for execution and it's computationally very efficient in the state-of-the-art of machine learning methods. Here the "Gini purity" method was implemented by this classifier used calculate the purity of split calculated by following equation.

$$Gini = 1 - \sum_{i=1}^{n} P^2(c_i)$$

The probability of class $c_i$ in that node is given by $P^2(c_i)$. Because the Gini index indicates that the characteristics are split in two, we may compute a weighted sum of the impurity for each partition individually. The classifier of weighted adaboost model is given below.

$$Cls_i = AdaBoostClassifier(DTclassifier(criteria =' gini', maxdepth = 5), learningrate = 1.0)$$

In the above function, there is a split strategy called "best." It is the default value. Another critical parameter is the "maximum depth of the tree." A larger maximum depth value results in overfitting, whereas a smaller one results in underfitting. As noted in the model specification, this value is set to "5". Another parameter used in this model is learning rate which is denoted as $\ell$, which is indicated by and represents the model's speed of learning. A weight of adaboost is set to $0.1 \leq \ell \leq 1.0$ in this study because this range was shown to be more appropriate during the tests. Another factor influencing accuracy is the number of estimators. The number of estimators has to increase the WABME and find the accuracy, the default value is 50, which indicates when boosting should be stopped. For weighted adaboost, the default value is used.

Algorithm 1 presents the working schema of the proposed model and its step-by-step execution.

**Algorithm 1:** Pseudo code of Weighted AdaBoost classifier

**Input:**

Let $X = \{X_1, X_2, X_3, \dots, X_n\}$ be dataset of software defect profiles,

Where, Training Dataset $X_i = \{X_{i,1}, X_{i,2}, X_{i,3}, \dots, X_{i,m}, st_i\}, X_i \in X, st_i \in \{'DEFECT', 'Normal'\}$, represents $i^{th}$software profile.

Here $st_i$ denotes software type, which can be either $'\boldsymbol{DEFECT}'$ or $'\boldsymbol{Normal}'$.

**Learning rate** $\boldsymbol{\ell}$

Learning turns $\boldsymbol{T}$ times

**Begin:**

**1.** Initialize the weights of each $X_i \in X$ : $W_i^t = \frac{1}{n}$

**2. for** $t \leftarrow 0$ *to* $T$ **do**

**3.** Add decision tree sequentially $DT^i(X)$ by using splitting along features by using information gain computation.

Find out the prediction of software defect from trained model

$$DT^i(X) : st = DT^i(\mathrm{X})$$

Select the model with least weighted prediction error:

$$e^t = Error\ \langle W^t \left[1_{st_i' \neq st_i}\right]_{i=1}^{n} \rangle$$

Compute the weight of $t^{th}$ model:

$$\delta^t = \frac{1}{2} \times \ln \langle \frac{1-e^t}{e^t} \rangle$$

Obtain DT Classifier (w.r.t gini, learning rate, other parameters)

Update the weight of each software defect profile $X_i$

$$W_{X_i'}^{i+1} = \frac{W^t\left(X_{i,1}, X_{i,2}, X_{i,3}, \dots, X_{i,m}, st_i\right) e^{(-\delta^t \times st_i \times DT^t(\mathrm{X_i}))}}{\theta},$$

// In the Boosting approach, update the distribution using the DT classifier.

Here is the normalization factor such that $\sum_{i=1}^{n} W_{\mathrm{i}}^{\mathrm{t}} = 1$ .

**If** ( $e^t - e^{t+1} < \lambda$ , here is the $\lambda$ threshold) then Break

**Else** Continue

**4.** Return the final prediction:

$$\Psi_{AdaBost}(X) = \sigma\left(\sum_{i=1}^{T} \delta^t DT^i(X)\right)$$

$$\Psi_{AdaBost}\ (X)$$

**5. Test the dataset due to**

**6. end for**

The average of the generated model outputs is used as the final forecast in test cases. If the prediction is positive than DP= 1, otherwise NDP = 0 is the expected class label

**7. END**

---

---

**Generate the output**

Solution to the test dataset and the success rate

## 4. Result and Discussion

In this section, the experimental design and result analysis are broken down in great depth before being presented and discussed.

### 4.1. Configuration of the Simulation Environment, System, and Parameters

The suggested approach has been implemented on a machine with Intel(R) Core(TM) i7-6700 processor at 3.40 GHz, 4.00 GB RAM, and Windows 10 64-bit operating system specifications. The simulation environment is comprised of Python Anaconda and Spyder IDE. The parameters of the classifiers (Table 3) are set by picking appropriate values through trial and error.

**Table 3: Variable Configuration**

| Classifiers | Parameter Setup |
|---|---|
| RandomForestClassifier | n_estimators : 100; Random state : 3 |
| KNeighborsClassifier | n_neighbors : 3 |
| LDA Classifier | random_state = 1 |
| GaussianNB | Priors: 'None' |
| SGDClassifier | loss= 'modified_huber'; shuffle = True; random_state = 101 |
| LogisticRegression | random_state = 9 |
| DecisionTreeClassifier | random_state = 108 |
| MLP Classifier | random_state = 1 |
| QDA Classifier | random_state = 1 |
| Proposed Model | Training and Testing Spit: 70% - 30% |

### 4.2. Statistical Competitiveness of Models

In this research work, we will compare the prediction performance of individual classifiers in a relatively recent and notable study in the literature. Then we compare all of the methods described in [26] from which we import SGD, RF, NB, LR, KNN, DT, LDA, MLP, and QDA are the nine well-known algorithms discussed for performance evolution with the proposed tree based model [32,33]. As a result, the same methods are employed and compared here together with the same dataset. Various performance metrics such as accuracy, TP, FP, TN, FN, TPR, FPR, precision, TNR, F1 score, and ROC-AUC has been computed. The prediction performance of RF and DT was comparable to that of separate classifiers, with only slight variations in accuracy and AUC ratings. The parameters are then adjusted, and a new approach WABE is developed, then we compare all of the methods described in **Table 4** and show the effectiveness of the proposed method with a difference (3.3%) in accuracy tree based methods.

The proposed weighted model for software defect prediction has been analyzed and compared with respect to variations in error rate with no. of estimators. Confusion matrix has been generated from the prediction of the proposed model for both version, i.e. real and discrete, and presented in **Figure 2.a** and **2.b** respectively.

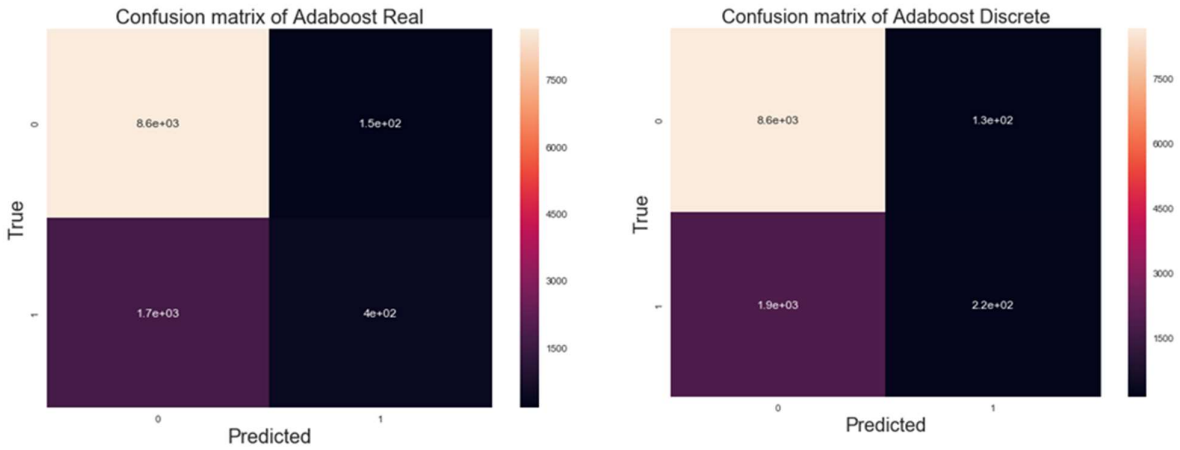


**Figure 2.** Confusion Matrix of a) WABE Real Classifier, b) WABE Discrete Classifier

**Table 4.** Comparison of Performance Metrics of all the models & proposed model

| Models | Performance Metrics | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Accuracy | TP | FP | TN | FN | TPR | FPR | Precision | TNR | F1 | ROC-AUC |
| SGD | 80.18 | 97 | 148 | 8631 | 2009 | 0.0460 | 0.0168 | 0.3959 | 0.98 | 0.0825 | 0.51 |
| RF | 92.30 | 1406 | 138 | 8641 | 700 | 0.6676 | 0.0157 | 0.9106 | 0.98 | 0.7704 | 0.82 |
| NB | 80.47 | 352 | 371 | 8408 | 1754 | 0.1671 | 0.0422 | 0.4868 | 0.95 | 0.2488 | 0.56 |
| LR | 80.89 | 181 | 155 | 8624 | 1925 | 0.0859 | 0.0176 | 0.5386 | 0.98 | 0.1482 | 0.53 |
| KNN | 83.09 | 843 | 577 | 8202 | 1263 | 0.4002 | 0.0657 | 0.5936 | 0.93 | 0.4781 | 0.66 |
| DT | 91.77 | 1631 | 430 | 8349 | 475 | 0.7744 | 0.0489 | 0.7913 | 0.95 | 0.7828 | 0.86 |
| LDA | 81.25 | 266 | 200 | 8579 | 1840 | 0.1263 | 0.0227 | 0.5708 | 0.97 | 0.2068 | 0.55 |
| MLP | 55.85 | 55 | 33 | 8746 | 2051 | 0.0261 | 0.0037 | 0.625 | 0.99 | 0.0501 | 0.51 |
| QDA | 80.04 | 419 | 485 | 8294 | 1687 | 0.1989 | 0.0552 | 0.4634 | 0.94 | 0.2784 | 0.57 |
| **Proposed WABE** | 93.51 | 1571 | 174 | 8605 | 535 | 0.7459 | 0.0198 | 0.9002 | 0.98 | 0.8158 | 0.86 |

We may compute a weighted sum of the impurity for each partition individually of the proposed approach updated the gini index $P^2(c_i)$ and maxdepth of the tree is 5, and find the accuracy of the proposed method is 93.51 with highest F1 score and ROC-AUC, which shows that the proposed model is capable to handle the imbalance class problem of software defect prediction data as compared to other models. The ROC curve of all the simulated models is shown in **Figure 3.** In a close ROC analysis, it is noticed that micro-average ROC and macro-average ROC covers the area of 0.97 and 0.94 respectively. Further, the both of ROC of class 0 (normal) and ROC of class 1 (defective) covered the area of 0.94, which is found better as compared to other models.

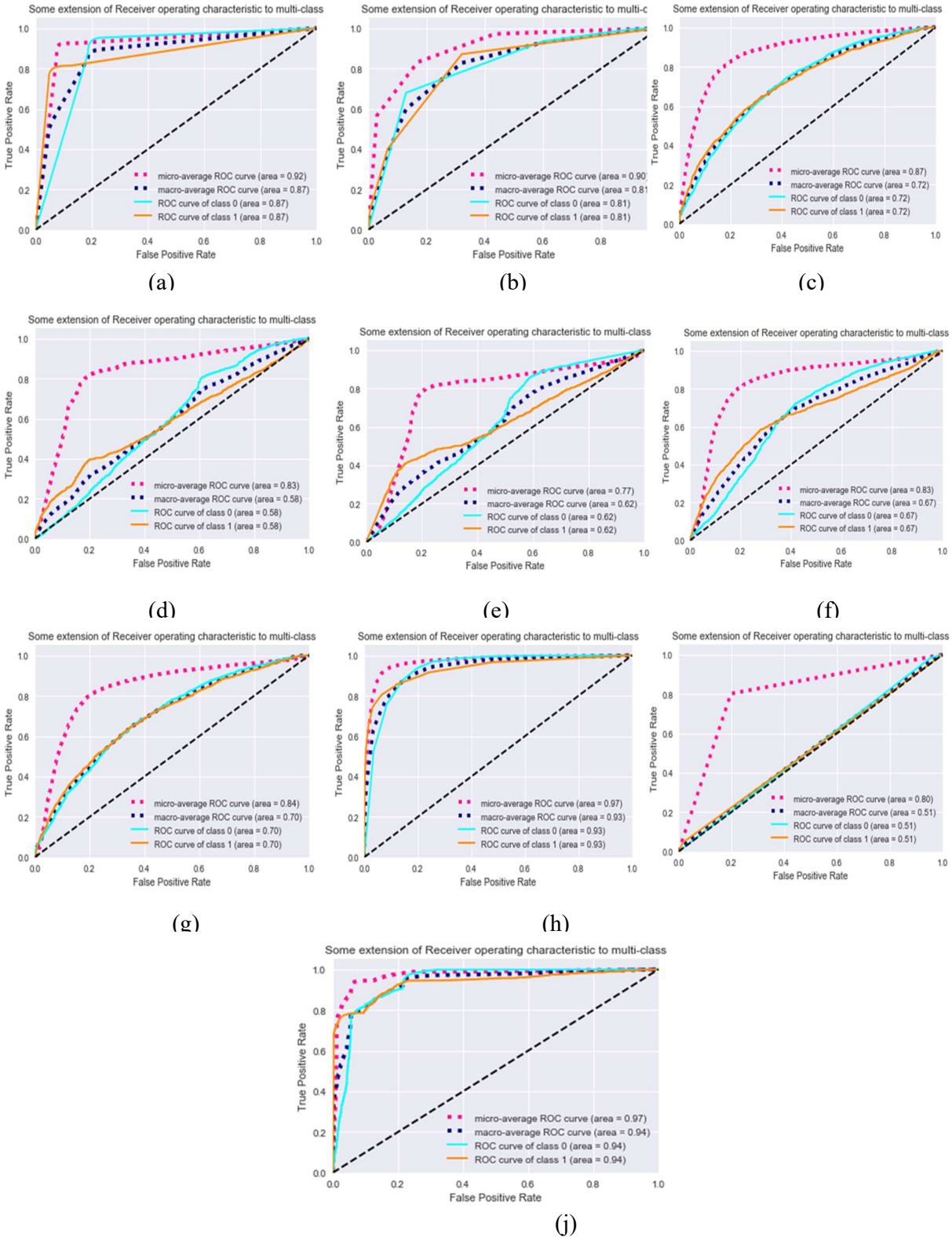


**Figure 3.** ROC Curve Analysis of a) DT, b) KNN, c) LDA, d) LR, e) MLP, f) NB, g) QDA, h) RF i) SGD and j) Proposed WABE

After update the weights of adaboost the classification report suggests that our proposed model resulted into less error rate (Training and Testing error) as shown in **Figure 4 and 5** comparisons to all models in state-of-art. In our model we used decision tree classifier as an estimator by changing their weights than calculate error of real and discrete WABE model results shown in **Table 5**.

**Table 5**. Training and testing error of real and discrete WABE methods

| **Models** | **Performance Metrics** | | |
|---|---|---|---|
| WABE Real | Train Error | 0.1710 | 0.0070 |
| | Test Error | 0.1861 | 0.0016 |
| WABE Discrete | Train Error | 0.1866 | 0.0015 |
| | Test Error | 0.1842 | 0.0018 |


**Figure 4.** Comparison of Training and Testing error of all the models with discrete WABE


**Figure 5.** Comparison of Training and Testing error of all the models with real WABE model

The dataset "Learning Curve" and "number of estimators" are shown in **Figure** 6 and 7, respectively, if the recommended methodology is to be seen. The training data set score and prediction accuracy are combined to represent the learning curve of a Decision Tree classifier in **Figure** 6. This graph demonstrates how accuracy and learning rates may be displayed simultaneously. These two figures, **Figure** 6 and 7, and the decision tree were used as an estimator to split the entire dataset using the best-Split function. Following the division of the data into training and test sets, which in this study are 75% and 25% respectively, the outcomes are automatically assessed and the findings are generated. WABE parameter learning rates are declared to be 0.1 and 1.0, respectively, and success rates are provided in the paper. It is reported that after changing gini and max depth these two values as specified, WABE had a fast convergence speed, implying that it gets extremely close to the final result relatively soon. It is also reported that WABME had a higher success rate in learning due to the other algorithms listed. Further, the learning rate determines how much each model contributes to the prediction. Smaller rates may necessitate more decision trees in the , whereas bigger rates may have fewer trees in the. It is usual to investigate learning rate values on a log scale, such as between 0.1 and 1.0.

In our study **Figure** 6 shows the WABE model mean accuracy for each learning rate is reported. Here, we find that higher learning rates lead to greater performance on training data. More trees in the with slower learning rates could improve performance even further.

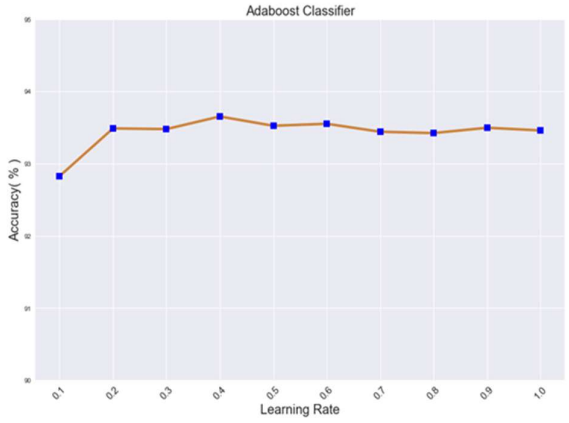


**Figure 6** Learning rate versus accuracy for WABE model

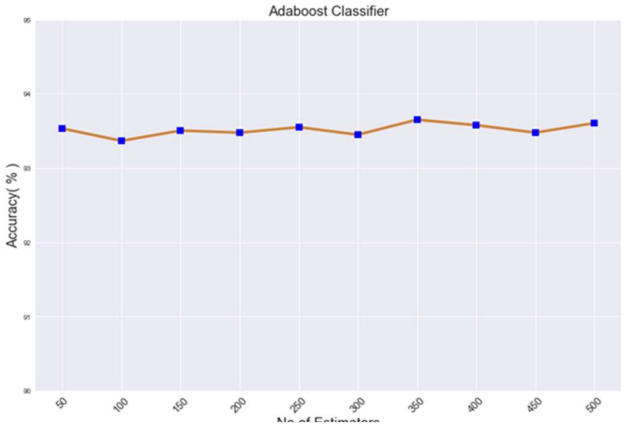


**Figure 7** No of estimators versus accuracy for WABE Model

For better prediction accuracy during the modeling stage, ML methods that use s are well-known. They play an important function since they are adaptable and can be used by a variety of estimators. These estimators are ideal for regression and classification issues because they can reduce bias and variance while also improving model performance. The effect of the selection of no. of estimators in WABE model on the software defect prediction accuracy has been presented in **Figure** 7 respectively. From the first round onwards the no. of estimator's range has to increase the outcome of WABE model performs well.

## 5. Conclusion

In this paper, a Decision Tree estimator-based weighted adaptive boost algorithm has been proposed for effectively predicts the defects. After performing trials, a classifier is tuned by updating their weights, and the binary classification problem is resolved. With an efficient identification of weak classifiers and updated their weights, the proposed method is able to predict the software defects with a high degree of precision. It claims to predict and perform in short periods of time, apart from its competition. The accuracy of the proposed model is 93.51% and is higher than other compared models. As a conclusion, WABE has the greatest accuracy for predicting output classes. WABE can be updated in the future to get better results with multi-class classifications. As future work, the proposed method can be used to predict the defects of some new datasets from various programming language based(C/ C++/JAVA) open-source projects. The proposed method can be simulated for the defect prediction at code and function change levels. Also, with the generation of automatic features from the dataset, some deep learning models like RNN may be used for defect prediction. Moreover, predicting the type of defect such as security threats, safety-critical factors, planning failure, etc. will be a challenging future area with the proposed approach.

## References


1. Z. Tian, J. Xiang, S. Zhenxiao, Z. Yi and Y. Yunqiang, "Software Defect Prediction based on Machine Learning Algorithms," 2019 IEEE 5th International Conference on Computer and Communications (ICCC), 2019, pp. 520-525, doi:10.1109/ICCC47050.2019.9064412
2. Eken, B., & Tosun, A. (2021). Investigating the performance of personalized models for software defect prediction. *Journal of Systems and Software*, *181*, 111038. https://doi.org/10.1016/j.jss.2021.111038
3. Qi, J., Du, J., Siniscalchi, S. M., Ma, X., & Lee, C. H. (2020). On mean absolute error for deep neural network based vector-to-vector regression. *IEEE Signal Processing Letters*, *27*, 1485-1489. **DOI:** 10.1109/LSP.2020.3016837
4. Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., & Ubayashi, N. (2014, May). An empirical study of just-in-time defect prediction using cross-project models. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (pp. 172-181). https://doi.org/10.1145/2597073.2597075
5. Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., & Husen, A. (2019). Performance analysis of machine learning techniques on software defect prediction using NASA datasets. *International Journal of Advanced Computer Science and Applications*, *10*(5).
6. Ren, J., Qin, K., Ma, Y., & Luo, G. (2014). On software defect prediction using machine learning. *Journal of Applied Mathematics*, *2014*. https://doi.org/10.1155/2014/785435
7. Prabha, C. L., & Shivakumar, N. (2020, June). Software defect prediction using machine learning techniques. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)* (pp. 728-733). IEEE. **DOI:** 10.1109/ICOEI48184.2020.9142909
8. Alsaeedi, A., & Khan, M. Z. (2019). Software defect prediction using supervised machine learning and techniques: a comparative study. *Journal of Software Engineering and Applications*, *12*(5), 85-100. DOI: 10.4236/jsea.2019.125007
9. Sangeetha, M., & Chandrasekar, C. (2019). An empirical investigation into code smells rectifications through ADA_BOOSTER. *Ain Shams Engineering Journal*, *10*(3), 549-553. https://doi.org/10.1016/j.asej.2018.10.005.
10. Zhao, L., Shang, Z., Zhao, L., Zhang, T., & Tang, Y. Y. Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. Neurocomputing, 352, 64-74,( 2019). https://doi.org/10.1016/j.neucom.2019.03.076
11. Ni, C., Chen, X., Wu, F., Shen, Y., & Gu, Q. An empirical study on pareto based multi-objective feature selection for software defect prediction. Journal of Systems and Software, 152, 215-238,(2019). https://doi.org/10.1016/j.jss.2019.03.012
12. Huang, Q., Xia, X., & Lo, D. (2019). Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empirical Software Engineering*, *24*(5), 2823-2862. https://doi.org/10.1007/s10664-018-9661-2
13. Fan, G., Diao, X., Yu, H., Yang, K., & Chen, L. (2019). Software defect prediction via attention-based recurrent neural network. *Scientific Programming*, *2019*. https://doi.org/10.1155/2019/6230953

14. L. Gong, S. Jiang and L. Jiang, "Tackling Class Imbalance Problem in Software Defect Prediction Through Cluster-Based Over-Sampling With Filtering," in *IEEE Access*, vol. 7, pp. 145725-145737, 2019, **DOI:** 10.1109/ACCESS.2019.2945858
15. Cai, X., Niu, Y., Geng, S., Zhang, J., Cui, Z., Li, J., & Chen, J. (2020). An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. *Concurrency and Computation: Practice and Experience*, *32*(5), e5478. https://doi.org/10.1002/cpe.5478
16. Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising autoencoders and two-stage learning. *Information and Software Technology*, *96*, 94-111. https://doi.org/10.1016/j.infsof.2017.11.008
17. Bowes, D., Hall, T. & Petrić, J. Software defect prediction: do different classifiers find the same defects?. *Software Qual J* 26, 525–552 (2018). https://doi.org/10.1007/s11219-016-9353-3.
18. T. J. Wang, F. Wu and X. Y. Jing, "Semi-supervised Learning Based Software Defect Prediction," Pattern Recognition and Artificial Intelligence, vol. 30, no. 7, pp. 646-652, 2017
19. X. Zhang and L. M. Wang, "Semi-supervised Learning Approach for Software Defect Prediction," Journal of Chinese Computer Systems, vol. 39, no. 10, pp. 2138-2145, 2018.
20. Xu, Z., Liu, J., Luo, X., Yang, Z., Zhang, Y., Yuan, P., ... & Zhang, T., Software defect prediction based on kernel PCA and weighted extreme learning machine. Information and Software Technology, 106, 182-200,(2019). https://doi.org/10.1016/j.infsof.2018.10.004
21. Yalçıner, B., & Özdeş, M., Software Defect Estimation Using Machine Learning Algorithms. In 2019 4th International Conference on Computer Science and Engineering (UBMK), September, (pp. 487-491). IEEE,(2019). **DOI:** 10.1109/UBMK.2019.8907149
22. Kakkar, M., Jain, S., Bansal, A., & Grover, P. S., Evaluating Missing Values for Software Defect Prediction. In 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), February, (pp. 30-34). IEEE, (2019) **DOI:** 10.1109/COMITCon.2019.8862444
23. Turabieh, H., Mafarja, M., & Li, X. (2019). Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. *Expert systems with applications*, *122*, 27-42. https://doi.org/10.1016/j.eswa.2018.12.033
24. Kassaymeh, S., Abdullah, S., Al-Betar, M. A., & Alweshah, M. (2022). Salp swarm optimizer for modeling the software fault prediction problem. *Journal of King Saud University-Computer and Information Sciences*, *34*(6), 3365-3378. https://doi.org/10.1016/j.jksuci.2021.01.015
25. Kondo, M., Bezemer, CP., Kamei, Y. *et al.* The impact of feature reduction techniques on defect prediction models. *Empir Software Eng* 24, 1925–1963 (2019). https://doi.org/10.1007/s10664-018-9679-5 .
26. S. Anuradha, K. E. R. G. R. (2019). Gradient Tree Boosting Approach for Software Defect Prediction. *International Journal of Advanced Science and Technology*, *28*(20), 750 -. Retrieved from http://sersc.org/journals/index.php/IJAST/article/view/2912.
27. Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep forest. *Information and Software Technology*, *114*, 204-216. https://doi.org/10.1016/j.infsof.2019.07.003

28. Goel, L., Sharma, M., Khatri, S. K., & Damodaran, D. (2020). Defect Prediction of Cross Projects Using PCA and Learning Approach. In *Micro-Electronics and Telecommunication Engineering* (pp. 307-315). Springer, Singapore. https://doi.org/10.1007/978-981-15-2329-8_31
29. Sohan, M. F., Kabir, M. A., Jabiullah, M. I., & Rahman, S. S. M. M. (2019, February). Revisiting the class imbalance issue in software defect prediction. In 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE) (pp. 1-6). IEEE. **DOI:** 10.1109/ECACE.2019.8679382
30. Sayyad Shirabad, J. and Menzies, T.J., The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, Available: http://promise.site.uottawa.ca/SERepository.
31. Prabha, Chander & Shivakumar, N. (2020). Software Defect Prediction Using Machine Learning Techniques. Conference: 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI). 728-733. https://doi.org/10.1109/ICOEI48184.2020.9142909.
32. M atloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., ... & Soomro, T. R. (2021). Software defect prediction using learning: A systematic literature review. *IEEE Access*.
33. A lsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Radaideh, A. A., Aljarah, I., & Alshamaileh, Y. (2020). Software defect prediction using heterogeneous classification based on segmented patterns. *Applied Sciences*, *10*(5), 1745. https://doi.org/10.3390/app10051745