Journal of Engineering Science and Technology Review 15 (2) (2022) 137 - 155

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

Research Article

Model-based Software Defect Prediction from Software Quality Characterized Code Features by using Stacking Ensemble Learning

P. Suresh Kumar[1], Janmenjoy Nayak[2,*] and H. S. Behera[3]

[1]Dept. of IT, Aditya Institute of Technology and Management (AITAM), Tekkali, AP-532201.
[2]Department of Computer Science, Maharaja Sriram Chandra BhanjaDeo University, Baripada, Odisha-757003, India
[3]Department of IT, Veer Surendra Sai University of Technology, Burla, Odisha, India

Received 15 November 2021; Accepted 7 June 2022

Abstract

Software defect prediction (SDP) is critical in guaranteeing software cost reduction and quality improvement while building a software system. Thus, software defect prediction in the early stages is an essential intrigue in the software engineering discipline. We proposed a stacking ensemble learning method to improve SDP performance based on software quality-defined code characteristics. Stacking combines the base classifiers by using a meta-classifier that learns base-classifiers output. It has certain advantages, such as easy implementation and combining classifiers by investigating various inducers. The proposed method's performance has been evaluated and compared with different Machine Learning (ML) classifiers on ivy2.0, tomcat, and velocity1.6 datasets available in PROMISE. The experimental findings revealed that the proposed approach has better prediction recall, accuracy, precision, AUC-ROC, and f-measure.

Keywords: Ensemble learning, Machine Learning, Software Defect Prediction, Stacking.

1 Introduction

In the present day, modern software systems are enormous and more intricate, leading to defects in the software. A defect can be considered a divergence from the requirement that affects the reliability and leads to failure or unforeseen results. Many activities have been employed, such as code review and testing, to improve the software's quality norms. However, such types of activities cost the financial plan. The number of defects increases in direct proportion to the software's complexity [1][2]. Software managers are concentrating on the modules with the most defects to enhance software quality. This is the rationale software defect prediction came into the scenario. SDP will create models to detect flaws in software components. Primarily, it will give ranks to the modules according to the severity of the modules. So, software managers focus and investigate those modules, which have several defects. By this, software engineers can test only on defect modules. The above-said process reduces software engineers, cost, and time to reduce the project's entire expense [3][4].

In SDP, classification and regression are popular methods. The main objective of regression SDP is to determine the overall number of faults in a given module. In the literature study, there are many regression models in SDP [1][5][6][7]. With classification, it will determine if the module is defect-free or defect-prone. In both cases, machine learning plays an important role. Machine learning algorithms, in particular Support Vector Machine (SVM) [8], Naïve Bayes (NB) [9], K-Nearest Neighbor (KNN) [10], Neural Networks (NN) [11], and Random Forest (RF) [12], is used to predict whether the module is defect-free or defect-prone.

Sometimes machine learning algorithms are not performed well in classification because of their limitations, such as lack of considerable data to train, data imbalance, and biases in the data. The primary goal of this study is to make use of the benefits of these machine learning algorithms while avoiding the drawbacks of ML by using EL techniques. Ensemble learning combines multiple base classifiers so that every classifier's better features will increase accuracy. Ensemble learning increases performance by using various base classifiers to reduce their variance and stop bias error rate.

This work presented an ensemble learning method called the stacking approach for identifying defect-free or defect-prone at the modular level using PROMISE datasets ivy2.0, tomcat, and velocity1.6. The performance of the proposed method stacking classifier has been validated by comparing several ML approaches GNB, SGD, KNN, MLP, DT, SVM, QDA, RF, LR, and LDA. The remaining sections of the paper are organized as follows: The literature review on SDP using different machine learning techniques and the proposed approach is presented in Section 2. The proposed technique's background and algorithm have been presented in Section 3. The experimental data and parameter setup for all of the methods are detailed in Section 4. Section 5 discusses several findings in the case of ivy2.0, tomcat, and velocity1.6. The conclusion is addressed in Section 6, along with future directions.

2 Literature Study

This section discusses the literature on ML and EL algorithms in SDP.

*E-mail address: jnayak@ieee.org

ISSN: 1791-2377 © 2022 School of Science, IHU. All rights reserved.
doi:10.25103/jestr.152.17

2.1 K-nearest neighbor algorithm
KNN is also called an instance-based classifier. KNN is a supervised machine learning (SML) technique that can classify and predict data. Its fundamental idea is to group similar things in proximity. It is based on the minimum distance between the query instance and the training sample to determine the number of neighbors.

Turabieh, Mafarja, and Li, 2019[13] proposed a layered recurrent neural network by selecting features with several techniques such as binary ant colony optimization, binary particle swarm, binary genetic algorithm, and binary ant colony. The PROMISE repository is used for experimentation by nineteen projects. The performance of the proposed approach is compared with various ML algorithms such as Naïve Bayes, decision tree, artificial neural networks, k-nearest neighbors, and logistic regression by considering the ROC-AUC. The outcomes demonstrate the proposed approach's superiority.

Wahono and Suryana, 2013[14] investigated two techniques, particle swarm optimization, for selecting features and bagging classifiers to deal with data imbalance and classification. The proposed approach's performance is compared with various techniques such as LDA, Naïve Bayes, k-nearest neighbors, k*, backpropagation, SVM, LibSVM by considering the NASA repository dataset c4.5, CART, and random forest. From the results, they have witnessed a significant improvement in prediction performance.

Balogun *et al.*, 2018[15] evaluated many individual techniques such as DT, SVM, **k-NN**, MLP, and various ensembles such as Bagging, boosting, stacking, voting classifiers for software defect prediction. Performance validation is done by the analytic network process using 11 SDP datasets. Among all classifiers, stacking and voting performed well.

2.2 Support vector machine
The SVM is useful in both regression and classification. SVM's main aim is to create a hyperplane within n-dimensional space that will categorize data points while maximizing the distance between the two classes.

Elish and Elish, 2008 [8] investigated the support vector machine's performance in SDP. They compared the prediction performance using various machine learning algorithms with four NASA datasets such as KC1, PC1, KC3, and CM1. They concluded that the Support vector machine's performance is better than various machine learning models.

Shuai *et al.*, 2013[16] proposed the cost-sensitive support vector machine (CSSVM) and genetic algorithm CSSVM (GA-CSSVM). In GA-CSSVM, the support vector machine is optimized with the genetic algorithm. They used geometric fitness function and enhanced the performance of GA-CSSVM. Moreover, they validated the performance using AUC values using the various dataset of MDP by NASA.

Yan, Chen and Guo, 2010 [5] proposed a novel fuzzy support vector regressors method. They used fuzzification to handle imbalance data and compared the proposed method with a fuzzy support vector repressor using MIS, RS-DIMU dataset. They discovered that the proposed methodology has a low mean square error and is more accurate.

2.3 Random Forest
It's a Supervised Machine Learning (SML) technique that can be used for both classification and regression. As the name implies, it has several decision trees that operate as a group. Every individual tree gives its prediction accuracy. From these accuracies, it selects the best accuracy based on voting. Magal. R and Gracia Jacob, 2015 [17] enhanced the random forest approach by using a feature selection algorithm based on the correlation among features to choose the best features from the datasets PC1, PC2, PC3, and PC4. They investigated the proposed method's computational framework and found it better than the traditional random forest method.

Kakkar and Jain, 2016[18] built a framework by utilizing attribute selection on different classifiers such as a random tree, KStar, IBk, LWL, and random forest on NASA-MDP datasets. They concluded that LWL performed well compared to other techniques with ten cross-fold validation accuracy and ROC curve.

2.4 Decision Tree
In machine learning, decision trees are the most frequently utilized classification and regression methods. This is a similar tree structure, with an internal node representing an attribute, a branch representing the result, and a leaf node holding the class's label. The decisions are established up of the route from the root node to the leaf.

Rathore and Kumar, 2016 [6] demonstrated the decision tree regressor's capability to forecast the defects in two situations, such as intra-release and inter-release prediction. They carry the experimental study by five open-source projects with various releases provided by the PROMISE repository. They test the performance using absolute error, relative error, the goodness-of-fit measure, and prediction at level l. The proposed decision tree regressor performed well in both scenarios.

Babu *et al.*, 2019 [19] analyzed various performances of ML methods such as **decision trees**, naïve Bayes, artificial neural networks, and linear classifiers in SDP. They utilized the Keel tool to evaluate them using k-fold cross-validation on the PROMISE repository datasets and found that other machine learning methods dominated the linear classifier.

Chug and Dhall, 2013[20] experimented with classification algorithms such as **decision tree** (J48), Naïve Bayes, and random forest. The performance is evaluated on the datasets from NASA by considering the measures such as ROC, RAE, precision, MAE, etc., and concluded that the random forest method outperformed compared to various classification models.

2.5 Neural networks
Several studies based on neural networks for effective SDP can be found in the literature. Kanmani *et al.*, 2007 [21] proposed two variant neural networks, probabilistic and backpropagation neural networks, to predict software defects. They compared these networks with statistical approaches and validated using various metrics such as completeness, effectiveness, and efficiency percentages using the datasets PC1-PC6. As compared to other methods, the probabilistic neural network is robust, and this outperformed well.

Li *et al.*, 2017[22] investigated software defect prediction through convolutional neural networks (DP-CNN). Deep learning has utilized to generate effective features, and the numerical vectors have been sent to CNN to learn structural features. Finally, conventional features are blended with newly learned features. They have considered various datasets such as jedit, camel, lucene, xalan, synapse, poi, and xerces for experimentation. In terms of f-measure, the proposed approach performs substantially better.

Manjula and Florence, 2019 [23] proposed a hybrid method; features have been selected using a genetic algorithm, and a deep neural network is utilized for classification. They enhanced the genetic algorithm performance by changing the design of the chromosome and fitness function. In the same way, they adopted an auto-encoder to improvise the DNN. Experimentation was carried by using the PROMISE repository, and the performance is verified by various measures such as specificity, recall, precision, f-score, sensitivity, accuracy, and recall.

Zhao *et al.*, 2019 [24] presented a Siamese dense neural network for learning similarity features and distance metrics. They utilized the cosine-proximity contract loss function. Using NASA's MDP measurements, they compared the proposed method to several conventional SDP methods. PD, PF, f-measure, MCC, and AUC are used to assess performance. The simulation results concluded that SDNN is stable and outperformed well compared to all the baseline models.

2.6 Logistic regression

It's an SML method that predicts a dependent variable's probability. It falls under two categories: a binary logistic regression model, where the possible values are 0 and 1, and multinomial logistic regression, where the dependent variable may have 3 or more unordered types.

Panichella, Oliveto and De Lucia, 2014 [25] came up with a novel approach Combined DEfect Predictors (CODEP), as an efficient defect prediction model. They experimented on ten open-source projects and used TPR, FPR, precision, and recall to verify the results. The results concluded that the proposed method is significantly superior to standalone applications such as LR, RBFN, ADtree, DT, multi-layer perceptron, and Bayes net.

Zhang *et al.*, 2016 [26] analyzed unsupervised classifiers' performance and supervised classifiers using 26 projects extracted from AEEM, NASA, and PROMISE repository. The connectivity-based classifier is proposed using spectral clustering, and the proposed method obtained better accuracy among logistic regression, naïve Bayes, random forest, logistic model tree, decision tree, and some clustering algorithms.

2.7 Ensemble learning

Ensemble learning is an ML technique that combines several basic classifier models to produce a more accurate prediction model. It is mainly used to enhance classification performance, prediction, and function approximation performance. Bagging, stacking, voting, and boosting are examples of ensemble learning methods.

Alsaeedi and Khan, 2019 [27] evaluated different supervised machine learning techniques and ensemble approaches on 10 NASA Datasets. They utilized SMOTE to deal with the skewed data and evaluated the results using accuracy, f-score, and AUC-ROC. They noticed that bagging, ada-boost, and random forest worked well.

Kaur and Kaur, 2014[28] evaluated bagging, boosting, and random forest using several base learning classifiers and experimented with datasets from the PROMISE repository. Proposed ensemble learning methods such as Bagging, boosting, and the random forest outperformed basic classifiers considerably in AUC.

Sayed and Ramadan, 2018 [29] utilized re-sampling techniques to handle the imbalance dataset and simulated using various ensemble learners such as boosting, bagging and random forest. They have utilized 8 base learners and tested on 7 datasets from the PROMISE repository. After comparing performance, they found that ensemble learning methods outperformed ML algorithms.

Laradji, Alshayeb, and Ghouti (2015) [30] integrated feature selection and EL methods to enhance software defect prediction performance. They experimented with various feature selection techniques and observed correlation-based forward selection to select features to produce better AUC. The enhanced version APE with greedy forward selection attained better performance using various NASA and PROMISE repository datasets.

3 Proposed Method

The framework of this proposed research is an integration of several independent methods (Figure 1). Data collection is collected from PROMISE repository, data preprocessing has been done based on correlation-based feature selection, and then the data has been supplied in an 80:20 split for training and testing.

3.1 Stacking Classifier

Stacking is a supervised ML technique for combining a set of predictions for binary classification, multi-classification, and regression. Stacking is also called a stacked regression [31] or super learner [32] developed in the year 1992 [33]. Though it was introduced many years ago, bagging and boosting are utilized widely compared to staking, which is difficult to examine theoretically. Stacking differs from bagging and boosting in these it utilizes the same kind of base learner while bagging and boosting use distinct types of base learners. It involves second-level training called meta-learner that will find optimal prediction from the combination of base learners. Base-level learners are generated by applying various learning algorithms to a stated dataset[34].

Algorithm: Stacking Classifier

```
Input: Data for training $DS = \{x_i, y_i\}_{i=1}^{m}$
Output: Ensemble Classifier H
Level 1: Learning algorithm at the base level classifier
for $l = 1$ to $L$ do
    learn $h_l$ based on $DS$
end for
Level 2: Creating various datasets for predictions
for $i = 1$ to $m$ do
    $DS_h = \{x_i', y_i\}$, where, $x_i' = \{h_1(x_i), \dots, h_L(x_i)\}$
end for
Level 3: meta-classifier learning
learn $H$ based on $DS_h$
return $H$
```

Considered features $X = \{x_i \in R^m\}$, set of class labels $Y = \{y_i \in N\}$ and data for training is given as $DS = \{x_i, y_i\}_{i=1}^{m}$, here the learning model is M on the training data DS. In the first level, learning is performed on the original training dataset with distributed weights, and learning parameters have tuned on the base classifier. At the second level, new datasets are created and predicted the labels from the output of first-level classifiers that are considered as new features. In place of using predicted labels, we can use probability estimators of the said first-level classifiers.

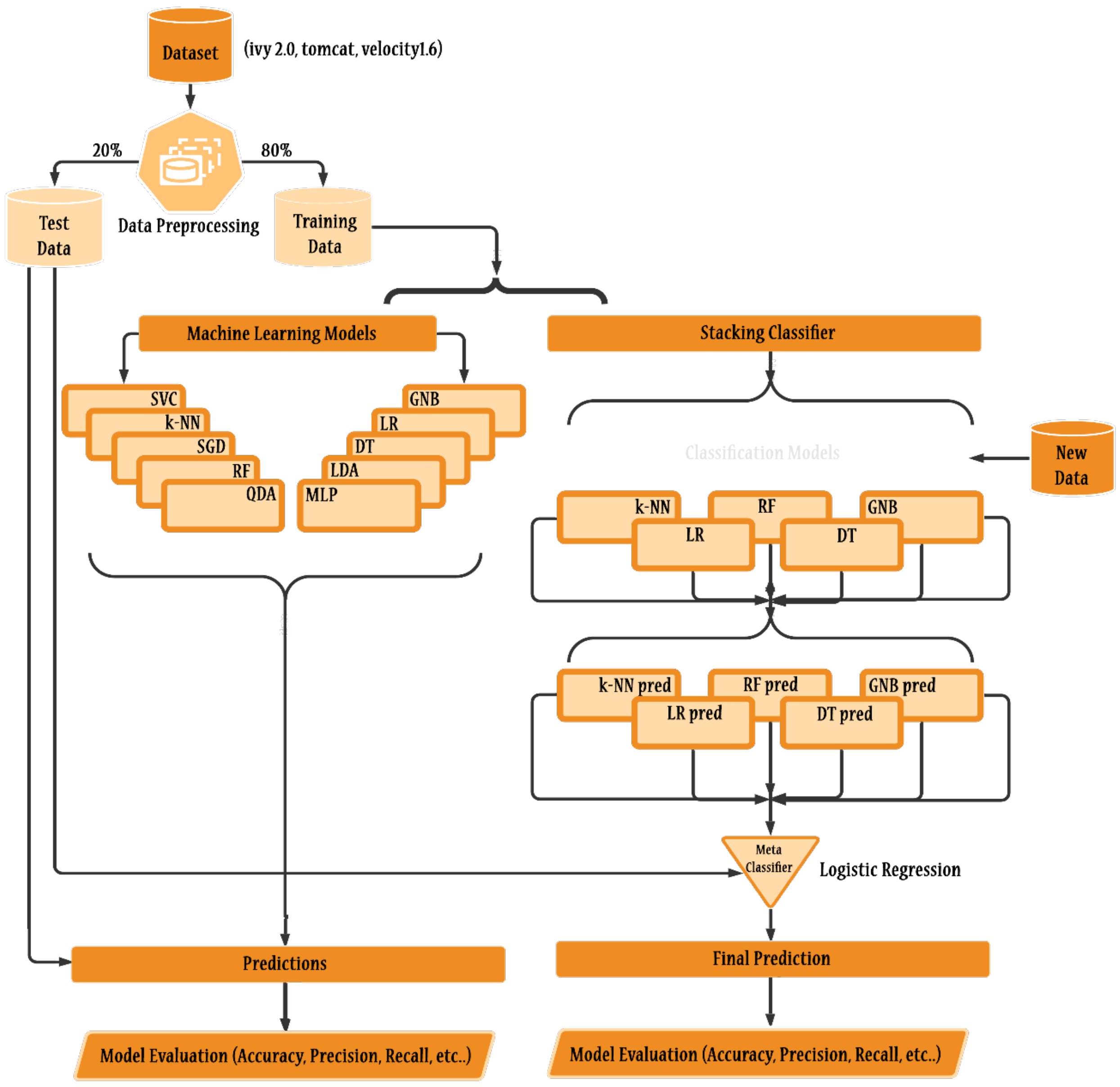


Fig.1. The framework of the Proposed Stacking classifier.

4 Experimental setup

This section discussed the datasets considered for experimentation, performance measures considered to evaluate the model, and experimental setups such as hardware and software setup for the working environment and parameter setting of the various machine learning models.

4.1 Empirical data

Experimentation is carried out in this article using open-source software code measurements. The PROMISE repository contains 44 datasets associated with 13 different software projects [35]. These are developed using object-oriented programming: java, and every feature in the dataset represents a java class. The experiment is conducted using datasets ivy2.0, tomcat, and velocity1.6. Table 1. shows the descriptive analysis of these datasets, containing 20 features of software metrics [36]. A software class is deemed defect-prone if it has one or more defects. Otherwise, it is free of defects [37].

4.2 Performance measure

This section described various classifiers used to validate the performance, such as precision, recall, accuracy, AUC-ROC, and f1-score. AUC obtains a superior assessment in all classifications because it is unaffected by changes in data distributions. [38]. Therefore, we used AUC as one of the main metrics to assess the proposed approach. The confusion matrix, also known as the error matrix, is used to compute the AUC based on the trade-off between false positive and true negative rates. It is a combination of various predicted and actual values. The confusion matrix for the two classes is as shown in **Table 2.** It is very useful in calculating several evaluating factors such as recall, true positive rate (TPR), false-positive rate (FPR), precision, f1-score, AUC, etc. [39].

Table 1. Dataset statistics.

Dataset	Corpus	Defective Ratio	#Modules	#Defective Modules
Ivy2.0	ck	11.363636	352	40
Tomcat	ck	8.97439	858	77
Velocity1.6	ck	34.061135	229	78

Table 2. Confusion matrix.

Actual Label	**Predicted Label**	
	Defect-Free	**Defect-Prone**
Defect-Free	TP	FN
Defect-Prone	FP	TN
TP: True Positive; **TN**: True Negative; **FP**: False Positive; **FN**: False Negative		

4.3 Simulation environment and parameter setup

This study examined an ensemble learning approach stacking classifier and various ML approaches such as SGD, SVC, KNN, QDA, RF, DT, LR, GNB, MLP, and LDA with ivy2.0, tomcat, velocity1.6 that are available at PROMISE repository. On the Windows 10 operating system, we utilized an Intel i5 CPU with 6 GB RAM. The proposed approach and several ML algorithms are implemented using Scikit-learn, an open-source machine learning library based on python. We explored the dataset features using a correlation matrix to find the relation among features and feature distribution. **Table 3** shows each classifier's distinct parameter setting with respective datasets ivy2.0, tomcat, and velocity1.6.

Table 3. Proposed method parameters in Jedit4.0, camel1.4, Ant1.7, ivy2.0, tomcat, and velocity1.6.

Techniques	**Parameter Setting**		
	Ivy2.0	**Tomcat**	**Velocity1.6**
Stacking	Classifiers : [KNeighborsClassifier('n_neighbors' = 50, 'algorithm' = 'kd_tree', 'weights' = 'distance'), RandomForestClassifier('random_state'=1), GaussianNB()] 'meta_classifier' : LogisticRegression(), 'use_probas' : 'True', 'use_clones' : 'False'	Classifiers : [KNeighborsClassifier('algorithm'='kd_tree' 'n_neighbors'=15,), RandomForestClassifier('random_state'=1), GaussianNB()] 'meta_classifier' : LogisticRegression(), 'use_probas' : 'True', 'use_clones' : 'False'	Classifiers : [DecisionTreeClassifier('criterion'='entropy', 'max_depth'=7, 'random_state'=2 'splitter'='best',), RandomForestClassifier('random_state'=1), BaggingClassifier(DecisionTreeClassifier(), 'n_estimators' = 400, 'random_state' = 1)] 'meta_classifier' : LogisticRegression(), 'use_probas' : 'True', 'use_clones' : 'False'

5 Result analysis

This segment portrayed the outcomes gained on ivy2.0, tomcat, velocity1.6 software metrics from the PROMISE repository with stacking. The proposed Stacking has been compared against several ML techniques such as KNN, SVM, LR, RF, MLP, SGD, GNB, LDA, DT, and QDA. Evaluation metrics such as confusion matrix, TP, FN, FP, TN, FPR, TPR, TNR, Accuracy, Precision, Recall, F1-score, and AUC-ROC have been considered to compare the models. We partition the rest of the segment into three cases for ivy2.0, tomcat, and velocity1.6 separately.

5.1 Test case 1: ivy2.0 dataset

In this case, the ivy2.0 dataset from the PROMISE repository is considered for experimentation. The dataset has 352 modules. From these, 88 percent are defect-free modules, and 12 percent are defect-prone modules represented in **Figure 2**. Descriptive statistics of the dataset has presented in **Table 4**.

The feature distribution is important for understanding the dataset's features. From the feature distribution, we can determine the data's possible temporal range and occurrences. Figure 3 depicts the ivy2.0 dataset's feature distribution. The feature distribution plots show that the metrics 'lcom3', 'cam', 'dam' is normally distributed features that imply these features are distributed at the same interval and may improve the classification accuracy. Features such as 'ca', 'npm', 'wmc', 'dit', 'cbo', 'rfc', 'loc', 'mfa', 'amc', 'avg_ccare' partially skewed and leaving the rest features 'lcom', 'moa', 'noc', 'ic', 'cbm', 'max_cc' are fully skewed. Ordinarily normal distributed features are extremely valuable in getting good accuracy than the partially skewed and fully skewed features.

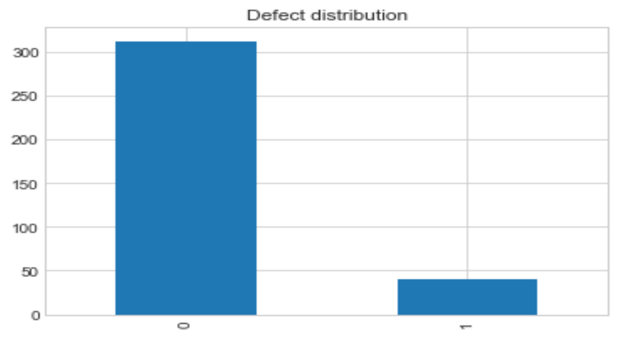


Fig. 2. Class distribution of the ivy2.0 dataset.

Proposed method Stacking contrasted with diverse ML techniques such as KNN, SVM, LR, RF, MLP, SGD, GNB, LDA, DT, and QDA. The confusion matrix of the proposed method over ivy2.0 is as shown in **Figure 4**.

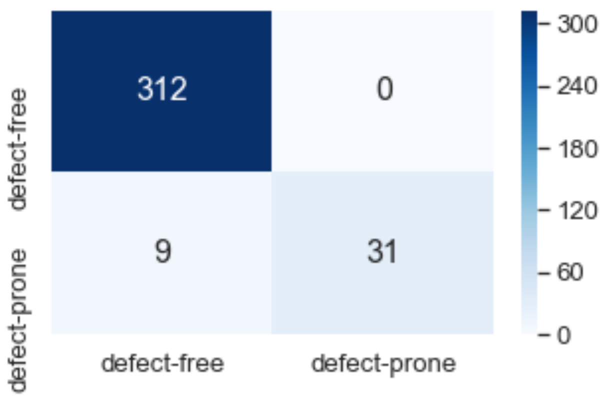


Fig. 4. Confusion matrix for ivy2.0 dataset on proposed method stacking classifier

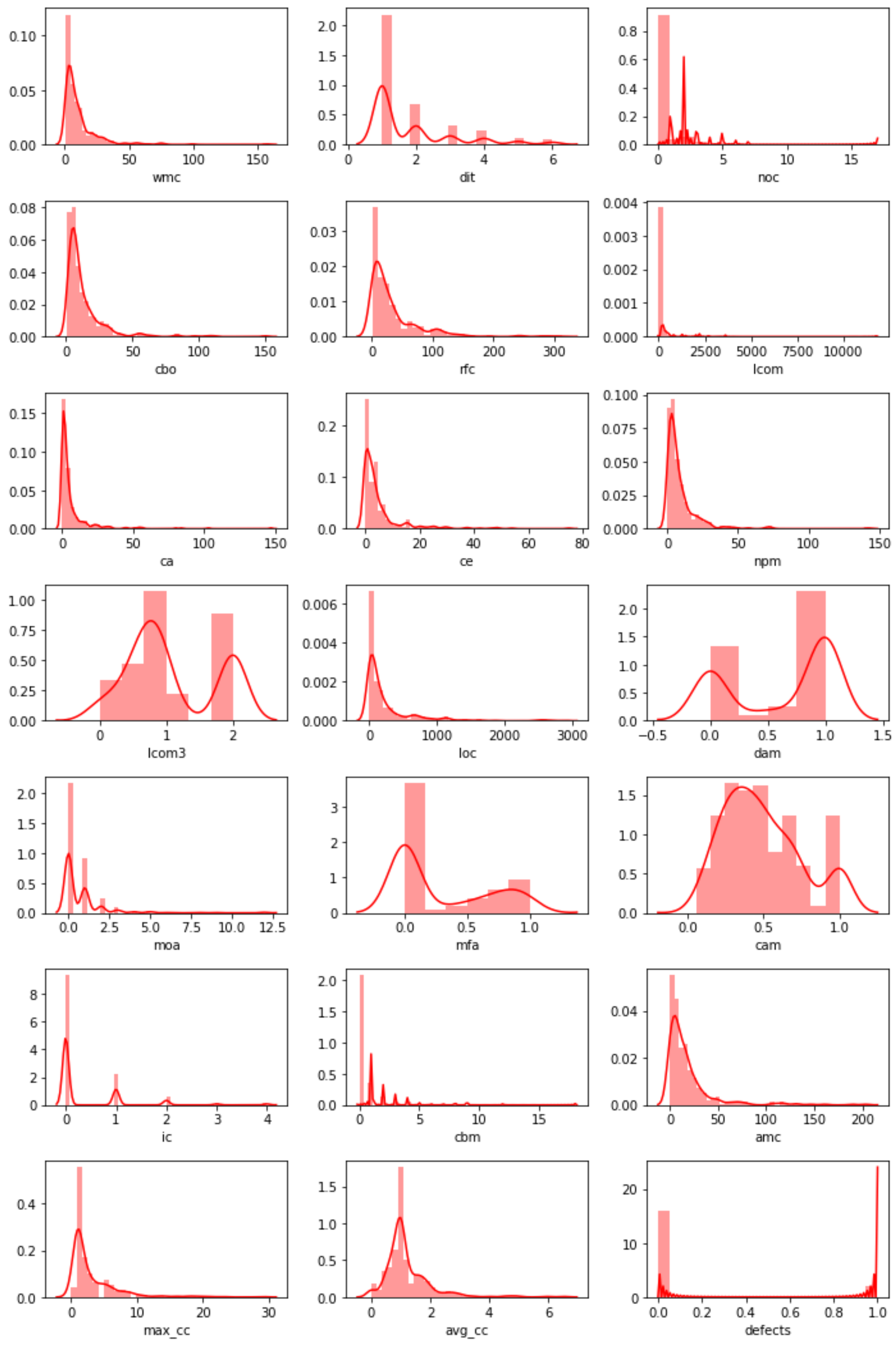


Fig.3. Feature distribution of ivy2.0 dataset.

Table 4. Description of the dataset - ivy2.0.

	MEAN	STD	MIN	25%	50%	75%	MAX
WMC	11.284091	15.148232	1	3	6	13	157
DIT	1.792614	1.244773	1	1	1	2	6
NOC	0.369318	1.318279	0	0	0	0	17
CBO	13.232955	16.571085	1	5	8	16	150
RFC	34.036932	44.679566	1	6	19	40	312
LCOM	131.579545	712.192029	0	0	6	45.25	11794
CA	6.880682	13.938917	0	1	3	6	147
CE	5.164773	8.931273	0	1	2	5	75
NPM	9.036932	12.636099	0	2	5	11	142
LCOM3	1.059352	0.660123	0	0.625	0.85	2	2
LOC	249.34375	428.259698	1	20	85.5	267	2894
DAM	0.616224	0.45994	0	0	1	1	1
MOA	0.715909	1.441737	0	0	0	1	12
MFA	0.290908	0.385164	0	0	0	0.670918	1
CAM	0.490831	0.254585	0.055223	0.299074	0.444444	0.666667	1
IC	0.357955	0.733601	0	0	0	0.25	4
CBM	0.636364	1.781077	0	0	0	0.25	18
AMC	18.489722	27.032755	0	4.666667	10.388199	21.434615	203.5
MAX_CC	3.1875	3.848123	0	1	2	4	29
AVG_CC	1.214294	0.816136	0	0.8	1	1.446925	6.5
DEFECTS	0.113636	0.317821	0	0	0	0	1

From the insights acquired from the confusion matrix, we figured different classification measures of the proposed technique and different machine learning strategies are presented in **Table 5**.

The proposed technique stacking classifier obtained 97.44 percentage, which is better contrasted with other standard machine learning algorithms. Other than the proposed technique, methods such as decision tree, random forest, quadratic discriminant analysis, linear regression, and support vector machine performed well with accuracy 94.6, 93.46, 92.61, 90.62, 90.05 individually. Out of 352 modules, 312 are properly classified as defect-free modules based on the findings of the proposed approach. 31 defect-prone modules have been properly identified as defect-prone. Next, the true positive rate is that the pace of positive examples is correctly classified, the TPR value of the stacking classifier is 0.77, and the remaining are in the middle of 0.01 to 0.6.

The stacking classifier and random forest are better regarding false positive rate and precision. All methods performed well w.r.t. the true negative rate. The stacking classifier beat well on account of f-measure and ROC-AUC. By considering all the performance measures, the proposed method stacking classifier is superior to the other machine learning models with stable results.

Comparative analysis of the proposed stacking and several ML techniques concerning the FPR, TPR, TNR, Accuracy, Precision, Recall, F1-score, and AUC-ROC on ivy2.0 is represented graphically in **Figure 5**.

The suggested technique's AUC-ROC curve, as well as other ML approaches are plotted in the center of true positive and false-positive rates, as shown in Figure 6 based on the ivy2.0 dataset (i) through (xi).

Table 5. Statistical performance analysis on ivy2.0.

Prediction Models	Performance Metrics										
	Accuracy	TP	TN	FP	FN	FPR	TPR	TNR	Precision	F-mes	ROC-AUC
KNN	88.92	4	309	3	36	0.009	0.1	0.99	0.571	0.17	0.545
SGD	87.215	7	300	12	33	0.038	0.175	0.961	0.368	0.237	0.568
RF	93.465	17	312	0	23	0	0.425	1	1	0.596	0.712
GNB	85.511	18	283	29	22	0.092	0.45	0.907	0.382	0.413	0.678
LR	90.625	13	306	6	27	0.019	0.325	0.98	0.684	0.44	0.652
DT	94.602	25	308	4	15	0.012	0.625	0.987	0.862	0.724	0.806
LDA	90.625	15	304	8	25	0.025	0.375	0.974	0.652	0.476	0.674
MLP	86.931	16	290	22	24	0.07	0.4	0.929	0.421	0.41	0.664
QDA	92.613	24	302	10	16	0.032	0.6	0.967	0.705	0.648	0.783
SVC	90.056	6	311	1	34	0.003	0.15	0.996	0.857	0.255	0.573
Stacking (Proposed)	97.443	31	312	0	9	0	0.775	1	1	0.873	0.887

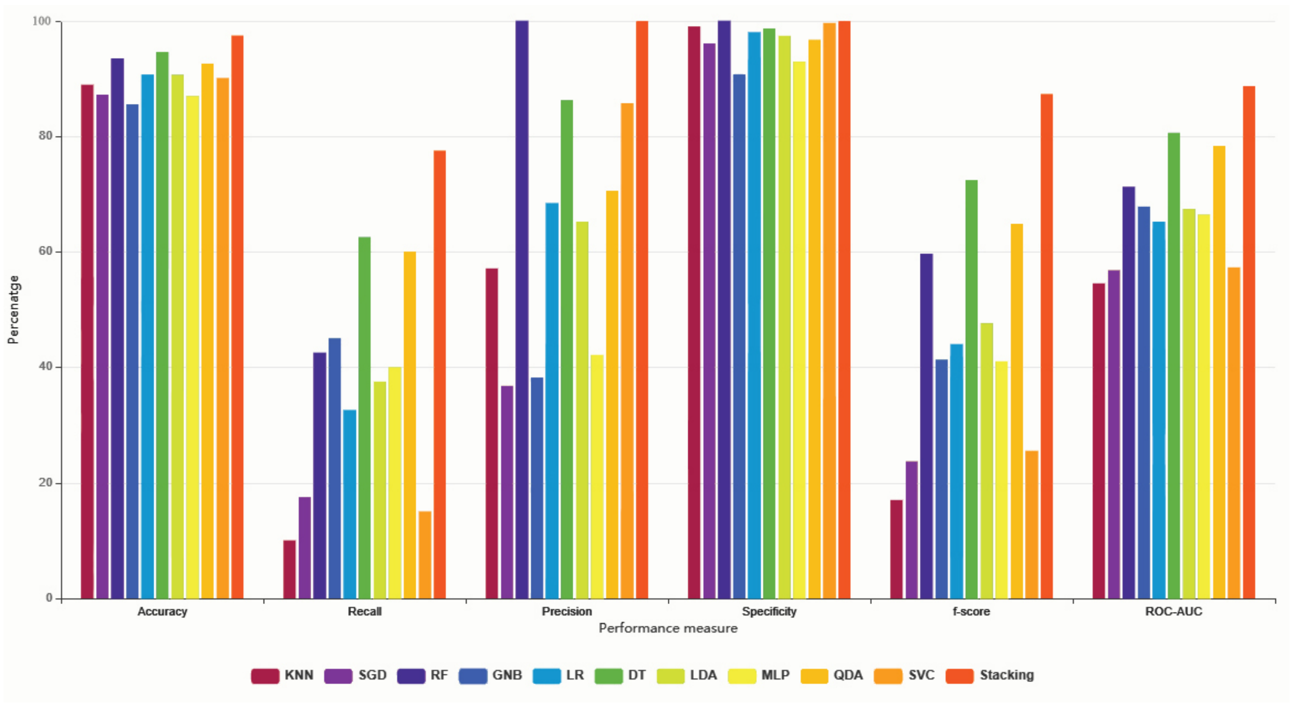
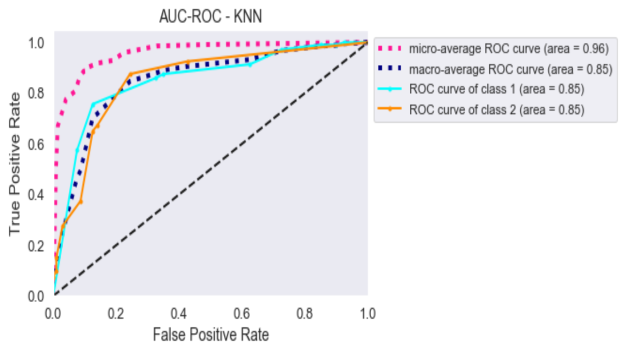
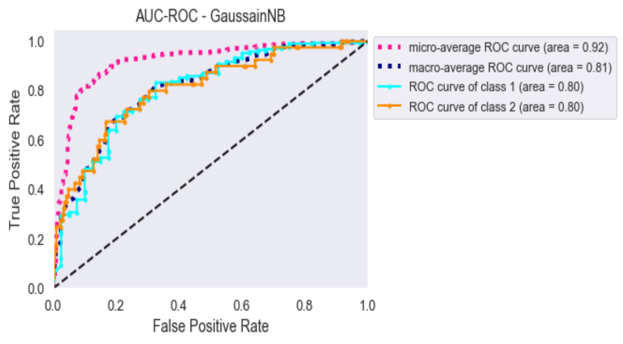


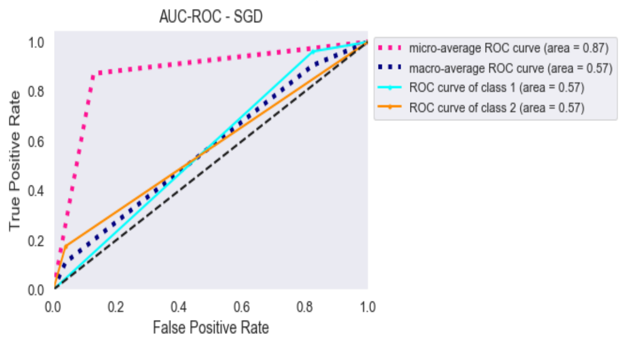
Fig. 5. Comparison of prediction results from various classifiers on ivy2.0 dataset.



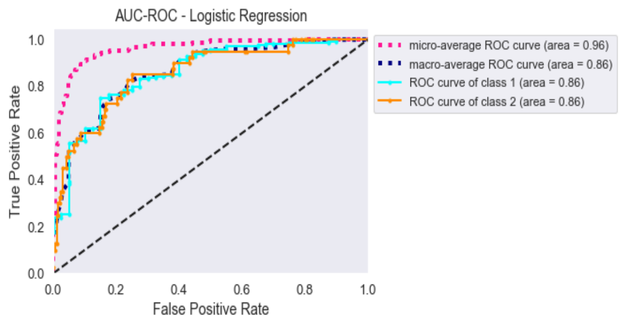
(i)



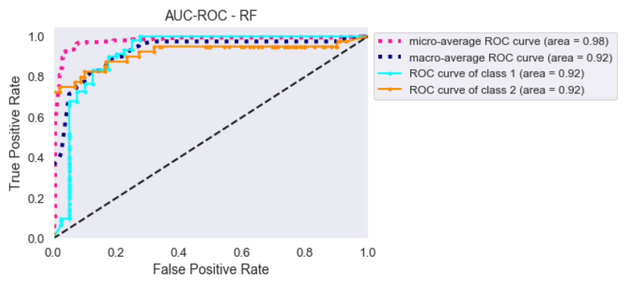
(iv)



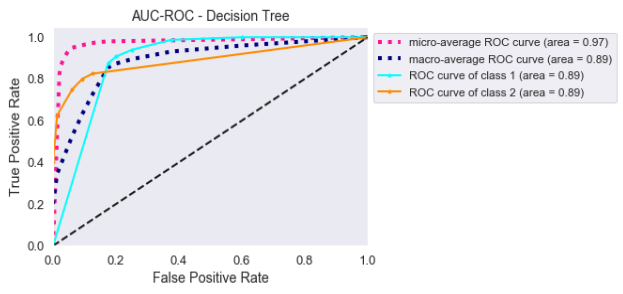
(ii)



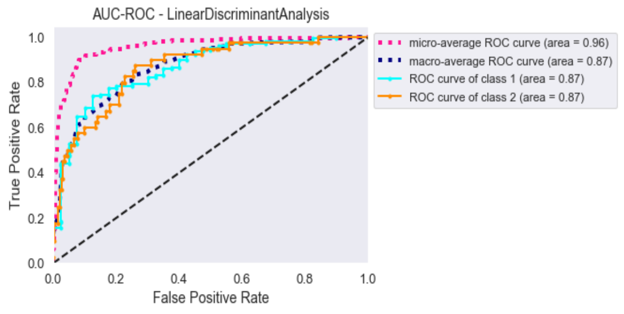
(v)



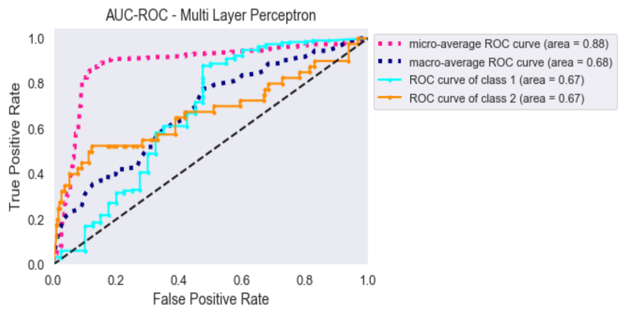
(iii)



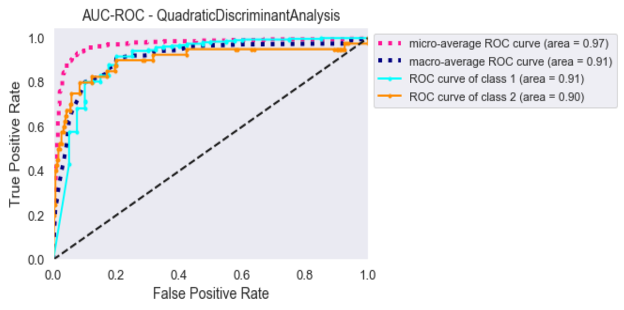
(vi)



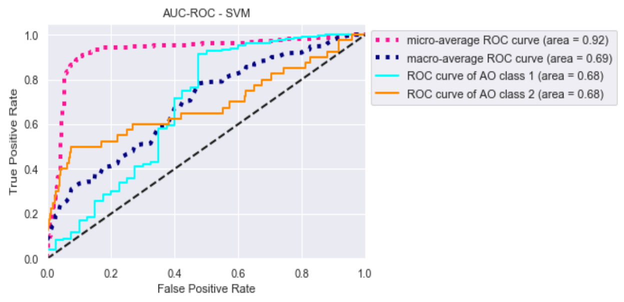
(vii)



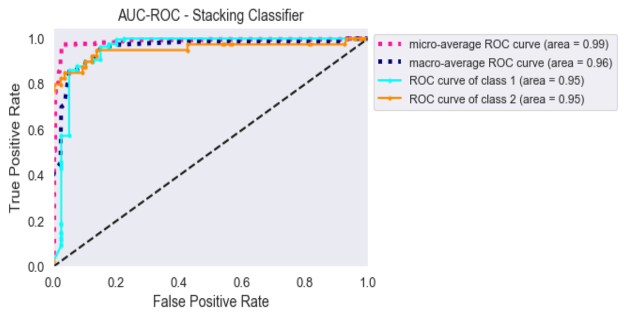
(viii)



(ix)



(x)



(xi)

Fig. 6. AUC-ROC curve of various classifier: i) KNN, ii) SGD, iii) RF, iv) GNB, v) LR, vi) DT, vii) LDA, viii) MLP, ix) QDA, x) SVM, and xi) stacking on ivy2.0.

5.2 Test case 2: Tomcat dataset

Here, an experimental study is conducted by the tomcat dataset from the PROMISE repository. Tomcat has 858 instances; among these, 91 percent of modules are defect-free modules, and the remaining 8 percent are defect-prone modules. The graphical portrayal of defect-free and defect-prone modules is shown in **Figure 7**. The investigations on the dataset's insights, such as mean, min, max, standard deviation, etc., are presented in **Table 6**.

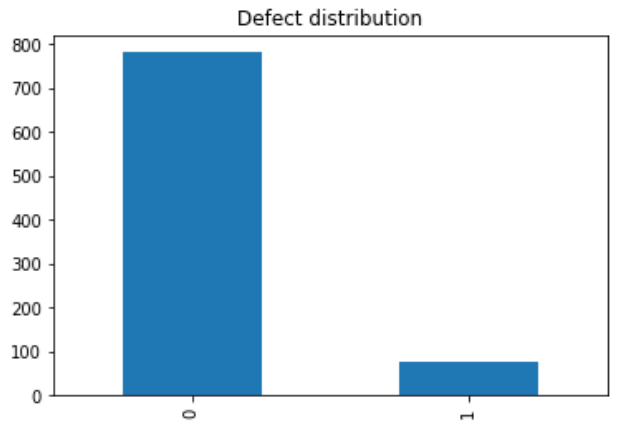


Fig. 7. Class distribution of the tomcat dataset.

Feature distribution gives us a better overview of the data, and it is shown in Figure 8. The features 'lcom3' and 'cam' are distributed normally; 'wmc', 'cbo', 'rfc', 'npm', 'dam', 'avg_cc' are skewed partially; and the remaining features 'ce', 'dit', 'noc','moa','mfa', 'lcom', 'ca', 'amc', 'loc', 'ic', 'cbm', 'max_cc' are skewed fully.

The proposed method stacking classifier and different machine learning algorithms have been experimented on the tomcat dataset. The confusion matrix for the proposed method is shown in **Figure 9.**

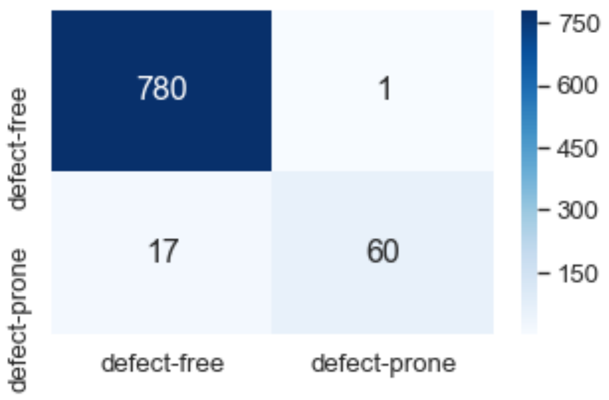


Fig. 9. Confusion matrix for tomcat dataset on proposed method stacking classifier.

From the confusion matrix, metrics such as true positive, false positive, true negative, and false negative of the proposed method are identified, and we figured the performance measures such as TP, FN, FP, TN, FPR, TPR, TNR, Accuracy, Precision, Recall, F1-score, and AUC-ROC in Table.7.

Table 6. Description of the dataset - tomcat.

	MEAN	STD	MIN	25%	50%	75%	MAX
WMC	12.95921	18.61893	0	3	7	14	252
DIT	1.687646	1.053022	1	1	1	2	6
NOC	0.363636	1.973732	0	0	0	0	31
CBO	7.573427	11.09689	0	2	4	9	109
RFC	33.47086	44.97656	0	7	17	39.75	511
LCOM	176.2762	1159.188	0	0	4	42	29258
CA	3.862471	8.90333	0	0	1	3	109
CE	0	0	0	0	0	0	0
NPM	10.77622	16.71321	0	2	5	12	231
LCOM3	1.086168	0.660434	0	0.625	0.871795	2	2
LOC	350.4359	644.839	0	25.25	112	373	7956
DAM	0.574051	0.471406	0	0	0.994048	1	1
MOA	0.944056	2.107749	0	0	0	1	24
MFA	0.293751	0.386574	0	0	0	0.666667	1
CAM	0.486463	0.253582	0	0.286667	0.444444	0.666667	1
IC	0.275058	0.578783	0	0	0	0	4
CBM	0.59324	1.74214	0	0	0	0	19
AMC	25.57757	46.64224	0	4.5	14.79048	29.96563	894.5
MAX_CC	4.271562	6.954404	0	1	1	5	95
AVG_CC	1.250478	1.002395	0	0.723225	1	1.5	10
DEFECTS	0.089744	0.285981	0	0	0	0	1

Table 7. Statistical performance analysis on tomcat.

Prediction Models	**Performance Metrics**										
	Accuracy	**TP**	**TN**	**FP**	**FN**	**FPR**	**TPR**	**TNR**	**Precision**	**F-mes**	**ROC-AUC**
KNN	91.142	1	781	0	76	0	0.012	0	1	0.025	0.506
SGD	83.799	24	695	86	53	0.11	0.311	0.889	0.218	0.256	0.6
RF	93.24	19	781	0	58	0	0.246	0	1	0.395	0.623
GNB	91.142	1	781	0	76	0	0.012	0	1	0.025	0.506
LR	91.724	14	773	8	63	0.01	0.181	0.989	0.636	0.282	0.585
DT	92.307	11	781	0	66	0	0.142	0	1	0.25	0.571
LDA	91.491	22	763	18	55	0.023	0.285	0.976	0.55	0.376	0.631
MLP	85.547	47	687	94	30	0.12	0.61	0.879	0.333	0.43	0.745
QDA	88.578	18	742	39	59	0.049	0.233	0.95	0.315	0.268	0.591
SVC	91.375	3	781	0	74	0	0.038	1	1	0.075	0.519
Stacking (Proposed)	97.902	60	780	1	17	0.001	0.779	0.998	0.983	0.869	0.888

The outcomes show that the proposed model stacking classifier outperformed well with accuracy 97.90, random forest, decision tree with accuracy 93.24 and 92.30 percent individually, and KNN, GNB, LR, and LDA have accuracy with 91 percent. The performance of the stacking classifier is significantly better than all the machine learning algorithms regarding the true positive rate. The stacking classifier, k-nearest neighbor, random forest, Gaussian naive Bayes, decision trees, and support vector machine are better in false-positive rate and precision. Performance of stacking classifier, quadratic discriminant analysis, linear discriminant analysis, and linear regression are remarkably predominant in true negative rate. The stacking classifier is strikingly one step better than both f-measure and ROC-AUC. Considering all performance measures, it is evident that the proposed technique stacking classifier dominated the performance on a greater scale over the other machine learning models. **Figure 10.** shows the predictions Accuracy, Precision, Recall, F1-score, and AUC-ROC of the proposed approach and different ML approaches on tomcat.

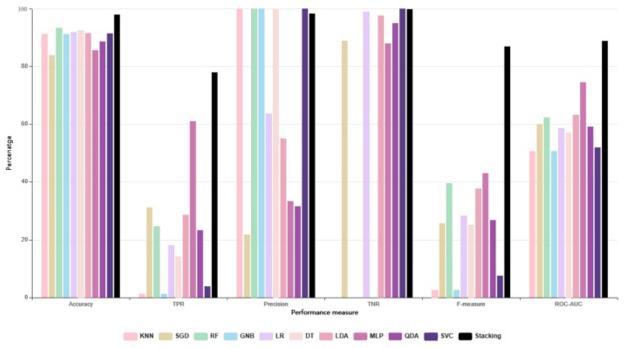


Fig. 10. Comparison of prediction results in various classifiers on the tomcat dataset

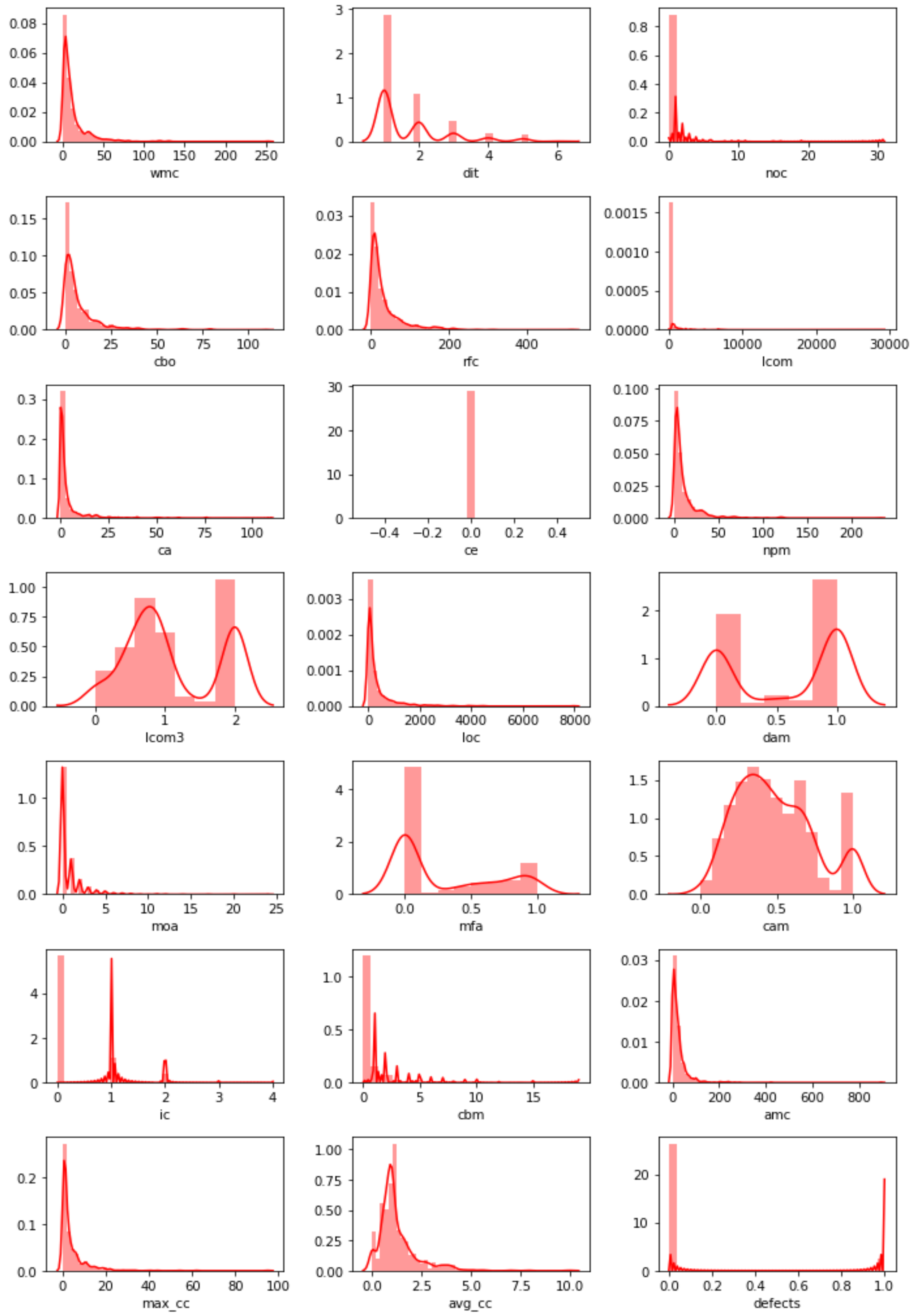
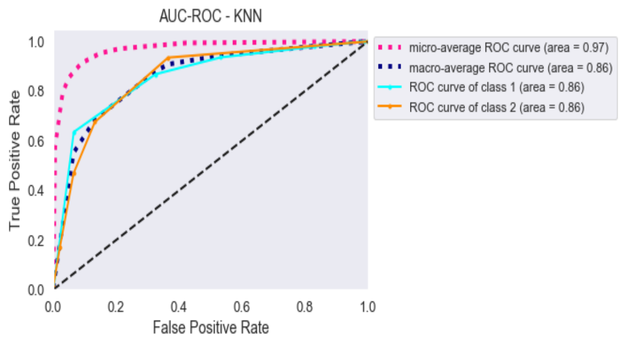
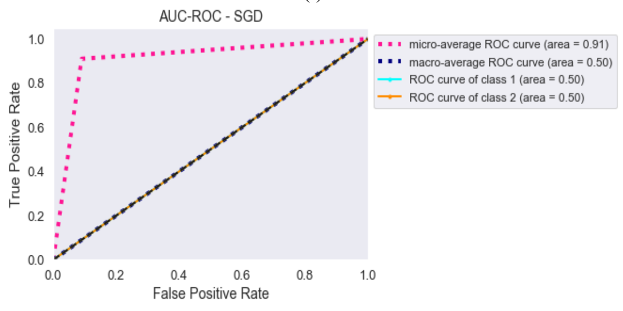


Fig. 8. Feature distribution of tomcat dataset.

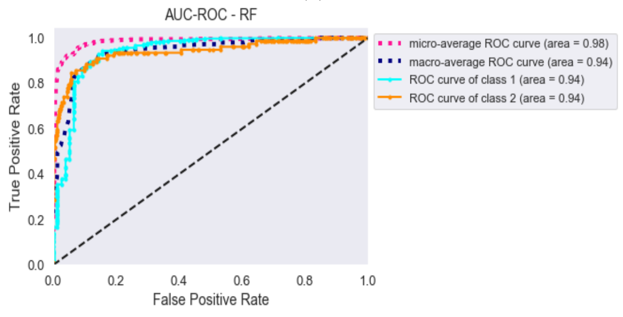
AUC-ROC curves of the suggested technique and different ML approaches are plotted in the middle of true positive rate and false-positive rates; those are presented in Figure 11: (i) to (xi) on account of the tomcat dataset.



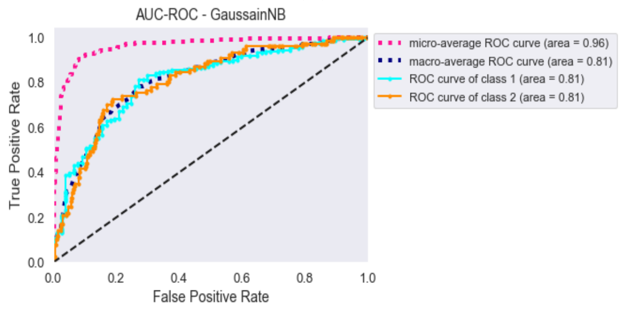
(i)



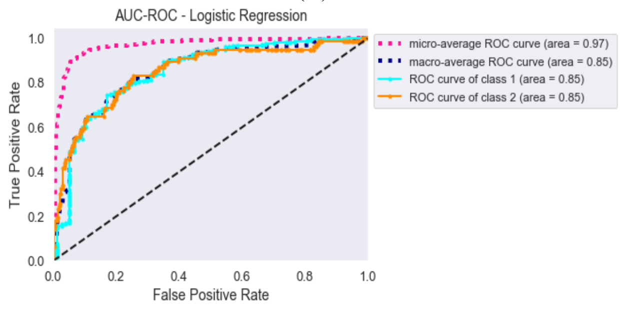
(ii)



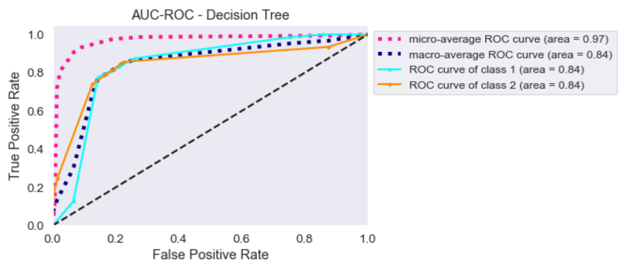
(iii)



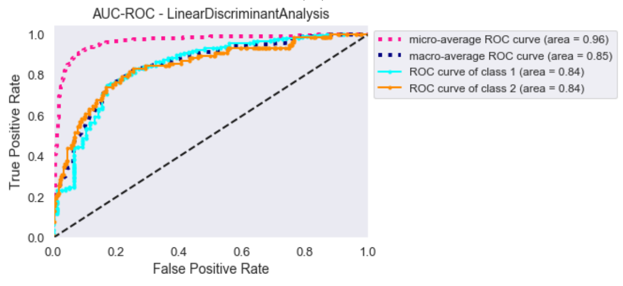
(iv)



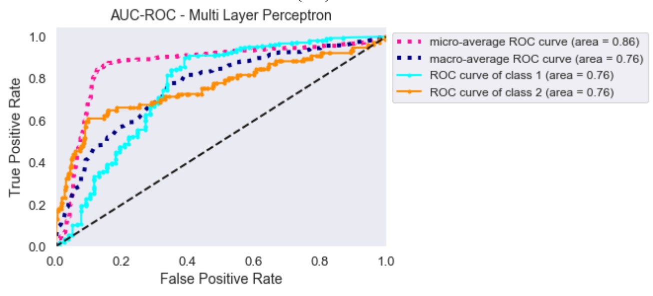
(v)



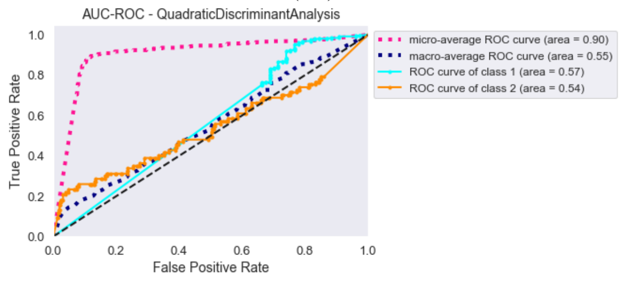
(vi)



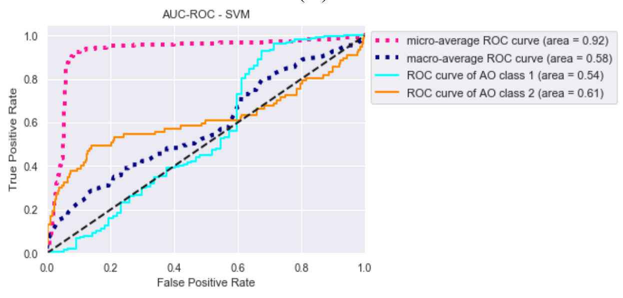
(vii)



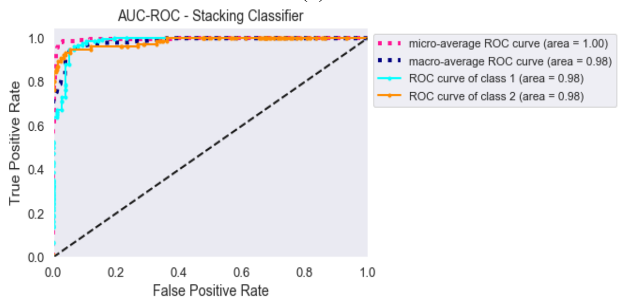
(viii)



(ix)



(x)



(xi)

Fig. 11. AUC-ROC curve of various classifier: i) KNN, ii) SGD, iii) RF, iv) GNB, v) LR, vi) DT, vii) LDA, viii) MLP ix) QDA, x) SVM, and xi) stacking on tomcat.

5.3 Test case 3: velocity1.6 dataset

Here, object-oriented dataset velocity1.6 is examined for experimentation. It has 229 modules that are 151 defect-free modules and 78 defect-prone modules, presented graphically in **Figure 12**. The dataset that consists of mean, standard deviation, min, max, percentiles of 25, 50, and 75 are shown in **Table 12.**

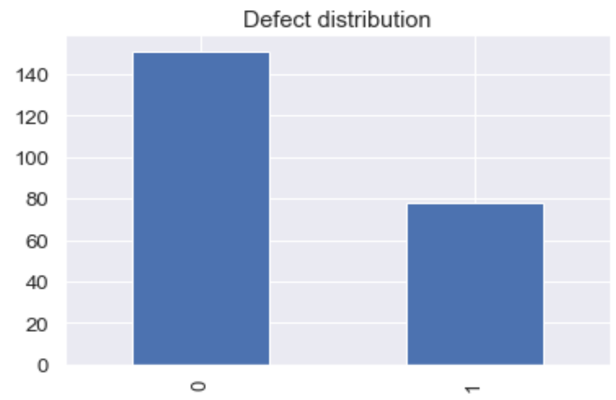


Fig. 12. Class distribution of the velocity1.6dataset.

Table 8. Description of the dataset – velocity1.6.

	MEAN	STD	MIN	25%	50%	75%	MAX
WMC	9.021834	14.13514	0	3	5	9	153
DIT	1.676856	0.888739	1	1	1	2	5
NOC	0.436681	2.754846	0	0	0	0	39
CBO	10.80786	1272.267	0	4	7	11	80
RFC	22.97817	27.36906	0	6	14	30	250
LCOM	80.34061	554.868	0	0	3	10	8092
CA	5.606987	11.17968	0	1	2	5	76
CE	5.982533	7.675985	0	1	4	8	61
NPM	7.218341	8.799193	0	3	5	7	50
LCOM3	1.232531	0.710674	0	0.625	0.956522	2	2
LOC	248.9607	1034.079	0	20	77	220	13175
DAM	0.432095	0.462687	0	0	0.166667	1	1
MOA	0.471616	1.145291	0	0	0	1	10
MFA	0.387857	0.411527	0	0	0	0.833333	1
CAM	0.465075	0.222449	0	0.333333	0.4375	0.541667	1
IC	0.31441	0.551563	0	0	0	1	2
CBM	0.489083	1.041302	0	0	0	1	9
AMC	19.60873	28.11434	0	4.333333	10.83333	26.66667	276
MAX_CC	3.9869	14.58925	0	1	1	4	209
AVG_CC	1.270836	1.855954	0	0.6667	1	1.3226	23
DEFECTS	0.340611	0.474953	0	0	0	1	1

Figure 13 depicts the dataset velocity feature distribution. The distribution of the features gives us a good overview of the data. The feature 'cam' is normally distributed; 'ce', 'cbo', 'rfc', 'lcom3','mfa', 'dam', 'amc' are skewed partially; and the other features 'noc', 'wmc', 'dit', 'lcom', 'ca', 'npm', 'loc', 'moa', 'ic', 'max_cc', 'cbm', 'avg_cc' are skewed fully. Normally distributed features play a critical role in increasing the accuracy than partially skewed features followed by fully skewed features.

The proposed method stacking and different ML techniques are tested with velocity1.6. The confusion matrix of the proposed method is shown in **Figure 14**.

Using confusion matrix estimates, we computed performance metrics such as true positive rate, false-positive rate, precision, true negative rate, f-measure, and ROC-AUC are shown in **Table 9.**

Table 9. Statistical performance analysis on velocity1.6.

Prediction Models	**Performance Metrics**										
	Accuracy	**TP**	**TN**	**FP**	**FN**	**FPR**	**TPR**	**TNR**	**Precision**	**F-mes**	**ROC-AUC**
KNN	77.292	28	149	2	50	0.013	0.358	0.986	0.933	0.518	0.672
SGD	65.938	14	137	14	64	0.092	0.179	0.907	0.5	0.264	0.543
RF	93.449	71	143	8	7	0.052	0.91	0.947	0.898	0.904	0.928
GNB	69.868	23	137	14	55	0.092	0.294	0.907	0.621	0.4	0.601
LR	78.602	42	138	13	36	0.086	0.538	0.913	0.763	0.631	0.726
DT	91.703	69	141	10	9	0.066	0.884	0.933	0.873	0.878	0.909
LDA	74.672	34	137	14	44	0.092	0.435	0.907	0.708	0.539	0.671
MLP	80.349	42	142	9	36	0.059	0.538	0.94	0.823	0.651	0.739
QDA	76.419	29	146	5	49	0.033	0.371	0.966	0.852	0.517	0.669
SVC	94.323	65	151	0	13	0	0.833	1	1	0.909	0.916
Stacking (Proposed)	95.196	71	147	4	7	0.026	0.91	0.973	0.946	0.928	0.941

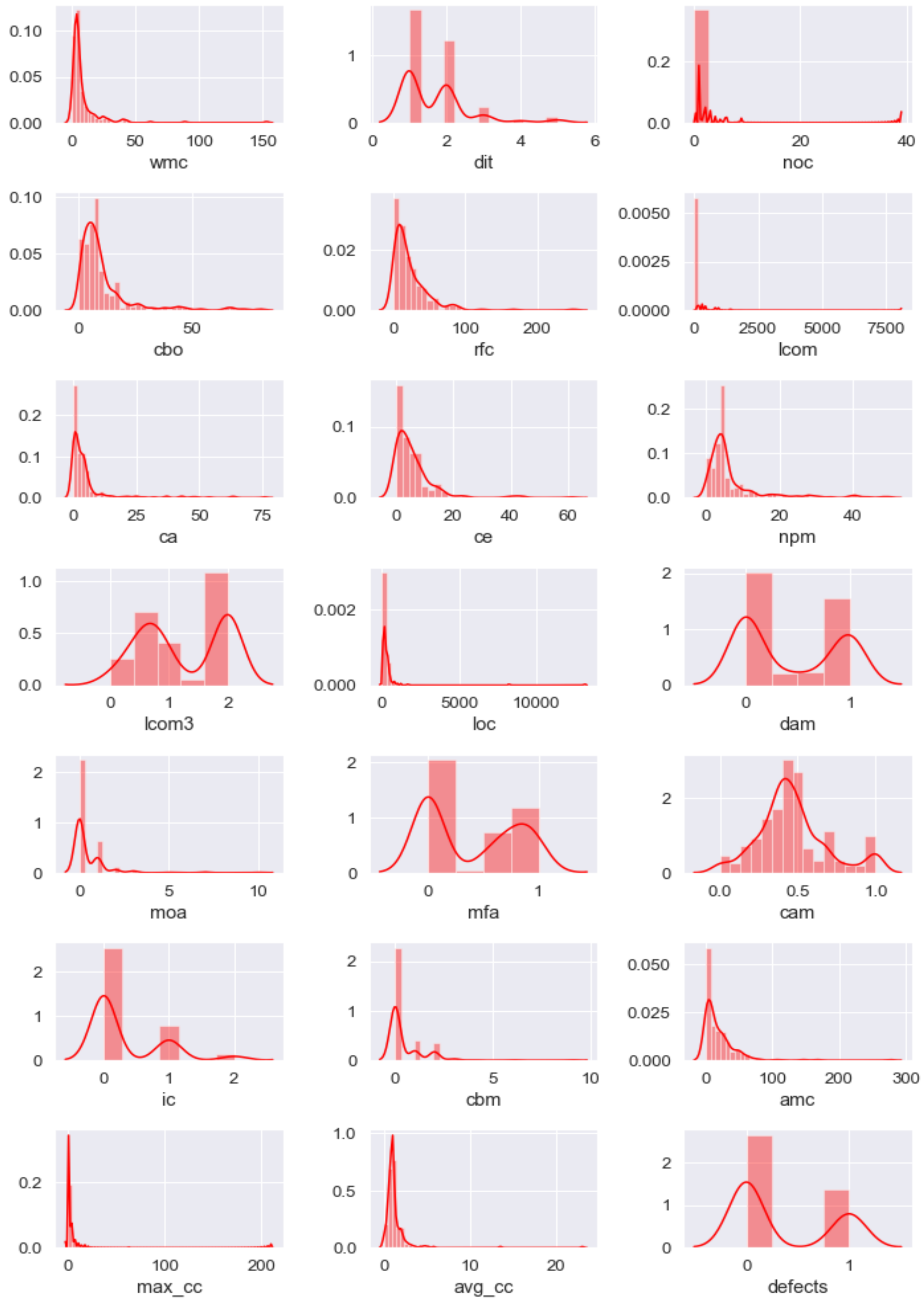


Fig. 13. Feature distribution of velocity1.6 dataset.

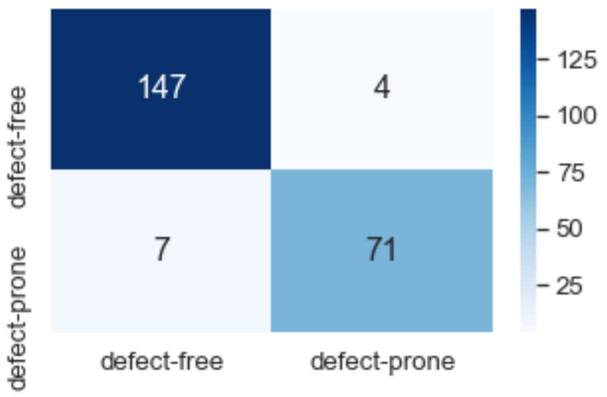


Fig. 14. Confusion matrix for velocity1.6 dataset on proposed method stacking classifier.

The results show that the proposed ensemble learning technique stacking outperformed various machine learning methods on the velocity1.6 dataset. The proposed method's performance in accuracy is 95.19% followed by SVM, RF, and DT with accuracy 94.32, 93.44, and 91.70, respectively. The stacking classifier and random forest are performed well in terms of true positive rate. KNN, SVM, and stacking are superior in false-positive rate and precision. SVM, KNN, stacking, and quadratic discriminant analysis outperformed well; besides, all other models performed well in terms of true negative rate. The stacking classifier, SVM, and RF performed well in f-measure and AUC-ROC. By considering all measures, the proposed technique stacking outperformed others, according to the results. The prediction performance of various measures such as accuracy, true positive rate, precision, true negative rate, f-measure, and ROC-AUC by the proposed method and different machine learning algorithms on velocity1.6 are shown in **Figure 15**.

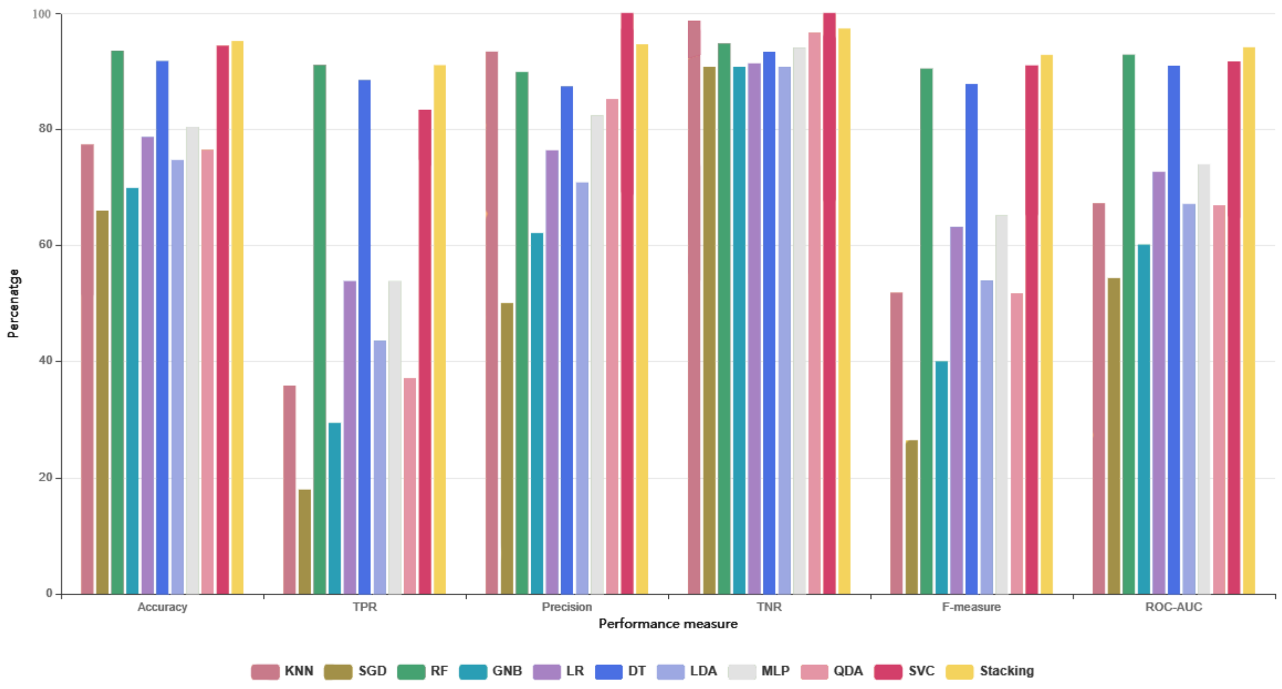
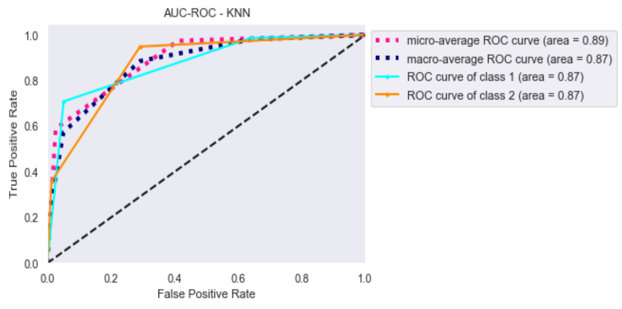
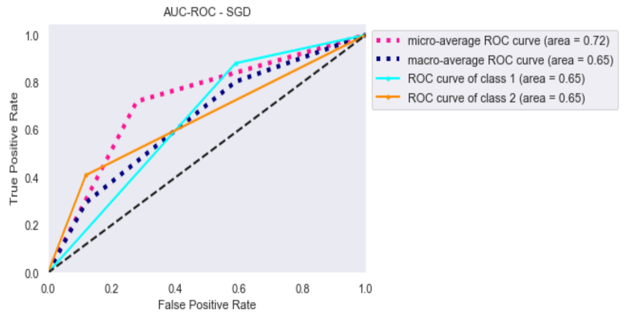


Fig. 15. Comparison of prediction results in various classifiers on velocity1.6 dataset.

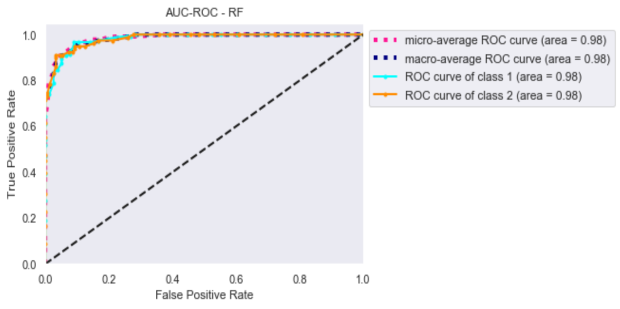
AUC- ROC curves of the proposed technique and different ML methods are shown in **Figure 16: (i) to (xi)** in the velocity1.6 dataset.



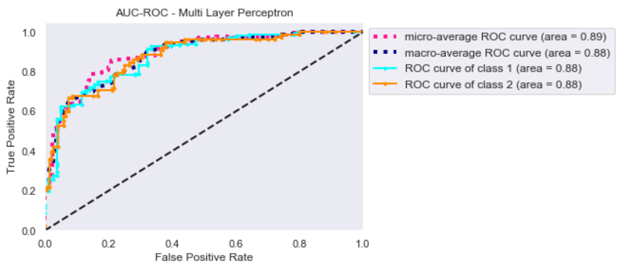
(i)



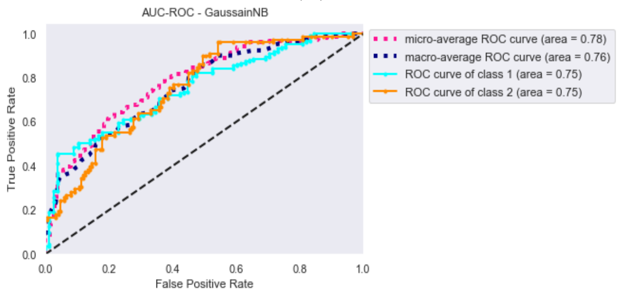
(ii)



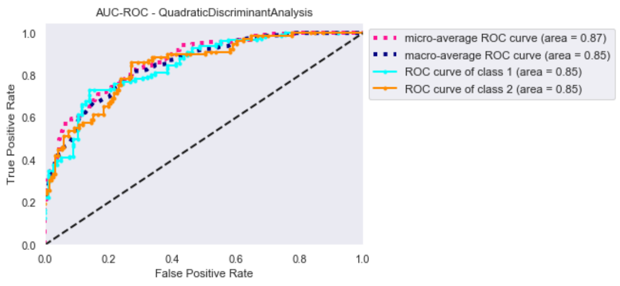
(iii)



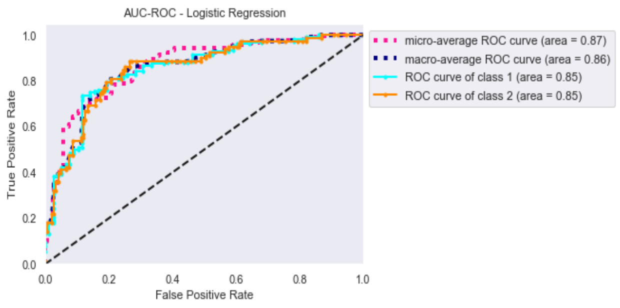
(viii)



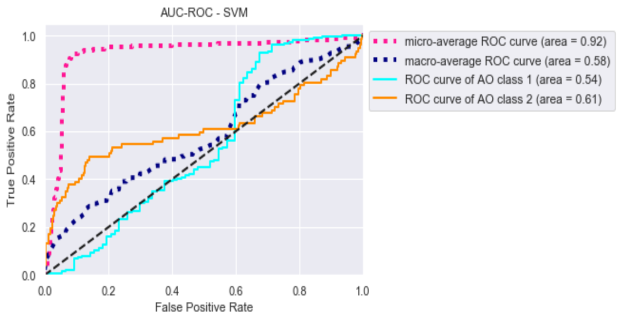
(iv



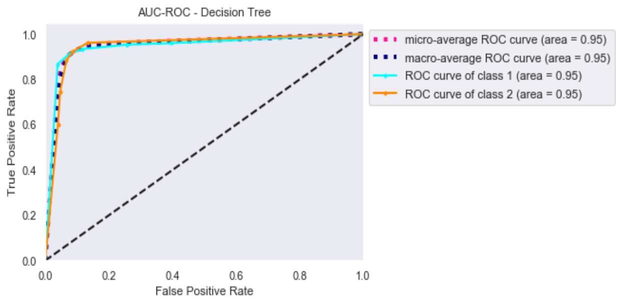
(ix)



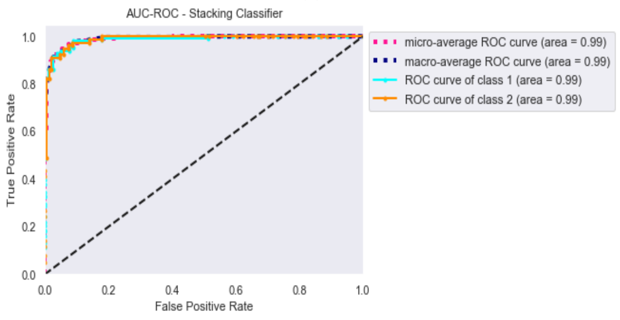
(v)



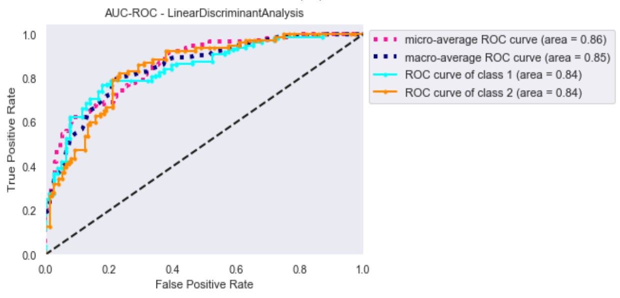
(x)



(vi)



(xi)



(vii)

Fig. 16. AUC-ROC curve of various classifier: i) KNN, ii) SGD, RF, iv) GNB, v) LR, vi) DT, vii) LDA, viii) MLP ix) QDA, x) SVM, and xi) king on velocity1.6.

Table 10 compares performance metric AUC values in the cases of ivy2.0, tomcat, and velocity1.6 with previous studies. In the case of ivy2.0 data, the proposed method's AUC value is 0.887, and from the literature highest AUC value is given by the Bayesian network with 0.846. Next in tomcat data, the proposed method AUC value is 0.869, and the highest AUC values from the literature are SC, NB, and LR, having 0.8. In velocity1.6 data, the AUC value is 0.928 in the proposed method, and from the literature, MLP performed well with 0.782. In all three cases, the proposed methodology outperformed prior research and several well-known ML models considerably.

Table 10. Previous studies in ivy2.0, tomcat, and velocity1.6

Dataset	Various classifier performance in terms of AUC							Ref
Ivy2.0	RF	SC	LMT	NB				[26]
	-0.71	-0.7	-0.7	-0.68				
	AUC							
	LR	DT	NB	RF	MLP	J48	SVM	[37]
	-0.774	-0.704	-0.769	-0.761	-0.697	-0.558	-0.534	
	AUC							
	Bayesian network							[40]
	-0.846							
	AUC							
	Bagging with							[28]
	NB	MLP	LR	J48	SVM			
	-0.762	-0.686	-0.76	-0.782	-0.589			
	AUC							
	Boosting with							
	NB	MLP	LR	J48	SVM			
	-0.65	-0.716	-0.65	-0.695	-0.698			
	AUC							
	Stacking Classifier							**Our Proposed Method**
	Accuracy: 97.443							
	AUC: 0.887							
Tomcat	SC	RF	NB	LMT				[26]
	-0.8	-0.78	-0.8	-0.77				
	AUC							
	Bayesian network							[40]
	-0.766							
	AUC							
	LR	DT	NB	RF	MLP	J48	SVM	[37]
	-0.747	-0.702	-0.799	-0.752	-0.762	-0.619	-0.5	
	AUC							
	Bagging with							[28]
	NB	MLP	LR	SVM	J48			
	-0.798	-0.663	-0.801	-0.517	-0.792			
	AUC							
	Boosting with							
	NB	LR	MLP	SVM	J48			
	-0.705	-0.705	-0.705	-0.77	-0.751			
	AUC							
	Stacking Classifier							**Our Proposed Method**
	Accuracy: 97.902							
	AUC: 0.869							
Velocity1.6	Bayesian network							[40]
	-0.678							
	AUC							
	LR	DT	NB	J48	MLP	RF	SVM	[37]
	-0.747	-0.679	-0.709	-0.665	-0.782	-0.785	-0.589	
	AUC							
	Bagging with							[28]
	LR	NB	MLP	SVM	J48			
	-0.767	-0.719	-0.583	-0.65	-0.761			
	AUC							
	Boosting with							
	NB	LR	MLP	SVM	J48			
	-0.752	-0.752	-0.751	-0.711	-0.768			
	AUC							
	Stacking Classifier							**Our Proposed Method**
	Accuracy: 95.196							
	AUC: 0.928							

6 Conclusion

SDP at an early stage helps to maintain software quality measures and improves software management procedures. This paper focused on an ensemble learning technique called stacking classifier and ML approaches such as SVC, SGD, KNN, RF, QDA, DT, GNB, LR, LDA, and MLP for SDP. The projects' software metrics are considered features; training and testing are performed on these data and extracted the input patterns provided by the datasets ivy2.0, tomcat, and velocity1.6 from the PROMISE repository.

Stacking classifier is an incredible machine learning paradigm that has displayed efficacy in many applications. Initially, all the attention has been on choosing the "best" single classification model from different models, where the proposed method can frame the combinations of models utilizing level one data. By combining various base classifiers, we achieved a low error rate through stacking.

Extensive experimental work is carried out through three cases, performance comparision has been done on the proposed method and ML methods on ivy2.0 and obtained the classification accuracy is 88.920, 87.215, 93.465, 85.511, 90.625, 94.602, 90.625, 86.931, 92.613, 90.056, and 97.443 for KNN, SGD, RF, GNB, LR, DT, LDA, MLP, QDA, SVC, and proposed stacking classifier respectively. Similarly, on the tomcat dataset,the achieved accuracies are

91.142, 83.799, 93.240, 91.142, 91.724, 92.307, 91.491, 85.547, 88.578, 91.375, and 97.902 for KNN, SGD, RF, GNB, LR, DT, LDA, MLP, QDA, SVC, and proposed stacking classifier respectively. Likewise, we experimented with these models on the velocity1.6 dataset and obtained accuracy are 77.292, 65.938, 93.449, 69.868, 78.602, 91.703, 74.672, 80.349, 76.419, 94.323, and 95.196 for KNN, SGD, RF, GNB, LR, DT, LDA, MLP, QDA, SVC, and proposed stacking classifier respectively. By utilizing an ensemble learning technique stacking, attained improved results over conventional machine learning algorithms. Finally, we attained higher accuracy, precision, recall, and AUC-ROC values using the proposed method. Further analysis is encouraged with various feature selection paradigm and optimization algorithms on the ensemble classifiers in future work. Moreover, a deep learning-based approach is to be developed to analyze the features and prediction of software defects effectively.

References

1. T. Ravi Kumar, T. Srinivasa Rao, and S. Bathini, “A predictive approach to estimate software defects density using weighted artificial neural networks for the given software metrics,” *Smart Innov. Syst. Technol.*, vol. 105, pp. 449–457, 2019, doi: 10.1007/978-981-13-1927-3_48.
2. P. Suresh Kumar and H. S. Behera, “Role of Soft Computing Techniques in Software Effort Estimation: An Analytical Study,” in *Computational Intelligence in Pattern Recognition*, 2020, pp. 807–831, doi: 10.1007/978-981-13-9042-5_70.
3. L. Qiao, X. Li, Q. Umer, and P. Guo, “Deep learning based software defect prediction,” *Neurocomputing*, vol. 385, pp. 100–110, Apr. 2020, doi: 10.1016/j.neucom.2019.11.067.
4. P. Suresh Kumar and H. S. Behera, “Estimating Software Effort Using Neural Network: An Experimental Investigation,” in *Computational Intelligence in Pattern Recognition*, 2020, vol. 1120, pp. 165–180, doi: 10.1007/978-981-15-2449-3_14.
5. Z. Yan, X. Chen, and P. Guo, “Software Defect Prediction Using Fuzzy Support Vector Regression,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6064 LNCS, no. PART 2, 2010, pp. 17–24.
6. S. S. Rathore and S. Kumar, “A Decision Tree Regression based Approach for the Number of Software Faults Prediction,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 41, no. 1, pp. 1–6, 2016, doi: 10.1145/2853073.2853083.
7. S. S. Rathore and S. Kumar, “An empirical study of some software fault prediction techniques for the number of faults prediction,” *Soft Comput.*, vol. 21, no. 24, pp. 7417–7434, Dec. 2017, doi: 10.1007/s00500-016-2284-x.
8. K. O. Elish and M. O. Elish, “Predicting defect-prone software modules using support vector machines,” *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, May 2008, doi: 10.1016/j.jss.2007.07.040.
9. T. Wang and W. Li, “Naive Bayes Software Defect Prediction Model,” in *2010 International Conference on Computational Intelligence and Software Engineering*, Dec. 2010, no. 2006, pp. 1–4, doi: 10.1109/CISE.2010.5677057.
10. H. Wang, T. M. Khoshgoftaar, and N. Seliya, “How many software metrics should be selected for defect prediction?,” *Proc. 24th Int. Florida Artif. Intell. Res. Soc. FLAIRS - 24*, no. Mi, pp. 69–74, 2011.
11. Y. Jiang, J. Lin, B. Cukic, and T. Menzies, “Variance analysis in software fault prediction models,” *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 99–108, 2009, doi: 10.1109/ISSRE.2009.13.
12. S. Aleem, L. F. Capretz, and F. Ahmed, “Benchmarking Machine Learning Techniques for Software Defect Detection,” *Int. J. Softw. Eng. Appl.*, vol. 6, no. 3, pp. 11–23, May 2015, doi: 10.5121/ijsea.2015.6302.
13. H. Turabieh, M. Mafarja, and X. Li, “Iterated feature selection algorithms with layered recurrent neural network for software fault prediction,” *Expert Syst. Appl.*, vol. 122, pp. 27–42, May 2019, doi: 10.1016/j.eswa.2018.12.033.
14. R. S. Wahono and N. Suryana, “Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction,” *Int. J. Softw. Eng. Its Appl.*, vol. 7, no. 5, pp. 153–166, Sep. 2013, doi: 10.14257/ijseia.2013.7.5.16.
15. A. O. Balogun, A. O. Bajeh, V. A. Orie, and A. W. Yusuf-asaju, “Software Defect Prediction Using Ensemble Learning: An ANP Based Evaluation Method,” *J. Eng. Technol.*, vol. 3, no. 2, pp. 50–55, 2018.
16. B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, “Software defect prediction using dynamic support vector machine,” in *Proceedings - 9th International Conference on Computational Intelligence and Security, CIS 2013*, Dec. 2013, pp. 260–263, doi: 10.1109/CIS.2013.61.
17. K. Magal.R and S. Gracia Jacob, “Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques,” *Int. J. Comput. Appl.*, vol. 117, no. 23, pp. 18–22, 2015, doi: 10.5120/20693-3582.
18. M. Kakkar and S. Jain, “Feature selection in software defect prediction: A comparative study,” in *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, Jan. 2016, pp. 658–663, doi: 10.1109/CONFLUENCE.2016.7508200.
19. N. Babu, Himagiri, V. Vamshi Krishna, A. Anil Kumar, and M. Ravi, “Software defect prediction analysis by using machine learning algorithms.,” *Int. J. Recent Technol. Eng.*, vol. 8, no. 2 Special Issue 11, pp. 3544–3546, 2019, doi: 10.35940/ijrte.B1438.0982S1119.
20. A. Chug and S. Dhall, “Software defect prediction using supervised learning algorithm and unsupervised learning algorithm,” *IET Conf. Publ.*, vol. 2013, no. 647 CP, pp. 173–179, 2013, doi: 10.1049/cp.2013.2313.
21. S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, “Object-oriented software fault prediction using neural networks,” *Inf. Softw. Technol.*, vol. 49, no. 5, pp. 483–492, 2007, doi: 10.1016/j.infsof.2006.07.005.
22. J. Li, P. He, J. Zhu, and M. R. Lyu, “Software Defect Prediction via Convolutional Neural Network,” in *2017 International Conference on Software Quality, Reliability and Security (QRS)*, Jul. 2017, pp. 318–328, doi: 10.1109/QRS.2017.42.
23. C. Manjula and L. Florence, “Deep neural network based hybrid approach for software defect prediction using software metrics,” *Cluster Comput.*, vol. 22, no. S4, pp. 9847–9863, Jul. 2019, doi: 10.1007/s10586-018-1696-z.
24. L. Zhao, Z. Shang, L. Zhao, A. Qin, and Y. Y. Tang, “Siamese Dense Neural Network for Software Defect Prediction With Small Data,” *IEEE Access*, vol. 7, no. c, pp. 7663–7677, 2019, doi: 10.1109/ACCESS.2018.2889061.
25. A. Panichella, R. Oliveto, and A. De Lucia, “Cross-project defect prediction models: L’Union fait la force,” in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, Feb. 2014, pp. 164–173, doi: 10.1109/CSMR-WCRE.2014.6747166.
26. F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, “Cross-project defect prediction using a connectivity-based unsupervised classifier,” in *Proceedings of the 38th International Conference on Software Engineering - ICSE ’16*, 2016, vol. 14-22-May-, pp. 309–320, doi: 10.1145/2884781.2884839.
27. A. Alsaeedi and M. Z. Khan, “Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study,” *J. Softw. Eng. Appl.*, vol. 12, no. 05, pp. 85–100, 2019, doi: 10.4236/jsea.2019.125007.
28. A. Kaur and K. Kaur, “Performance analysis of ensemble learning for predicting defects in open source software,” in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2014, pp. 219–225, doi: 10.1109/ICACCI.2014.6968438.
29. A. Sayed and N. Ramadan, “Early Prediction of Software Defect using Ensemble Learning: A Comparative Study,” *Int. J. Comput. Appl.*, vol. 179, no. 46, pp. 29–40, 2018, doi:

10.5120/ijca2018917185.

30. I. H. Laradji, M. Alshayeb, and L. Ghouti, “Software defect prediction using ensemble learning on selected features,” *Inf. Softw. Technol.*, vol. 58, pp. 388–402, 2015, doi: 10.1016/j.infsof.2014.07.005.
31. L. Breiman, “Stacked regressions,” *Mach. Learn.*, vol. 24, no. 1, pp. 49–64, Jul. 1996, doi: 10.1007/BF00117832.
32. M. J. van der Laan, E. C. Polley, and A. E. Hubbard, “Super Learner,” *Stat. Appl. Genet. Mol. Biol.*, vol. 6, no. 1, Jan. 2007, doi: 10.2202/1544-6115.1309.
33. D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–259, Jan. 1992, doi: 10.1016/S0893-6080(05)80023-1.
34. G. Wang, J. Hao, J. Ma, and H. Jiang, “A comparative assessment of ensemble learning for credit scoring,” *Expert Syst. Appl.*, vol. 38, no. 1, pp. 223–230, Jan. 2011, doi: 10.1016/j.eswa.2010.06.048.
35. O. T. Boetticher G, Menzies T, “PROMISE Repository of empirical software engineering data.” http://promisedata.org/repository.
36. P. Suresh Kumar, H. S. Behera, J. Nayak, and B. Naik, “Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature,” *Innov. Syst. Softw. Eng.*, no. September 2019, pp. 1–22, May 2021, doi: 10.1007/s11334-021-00399-2.
37. A. Kaur and K. Kaur, “An Empirical Study of Robustness and Stability of Machine Learning Classifiers in Software Defect Prediction,” vol. 320, E.-S. M. El-Alfy, S. M. Thampi, H. Takagi, S. Piramuthu, and T. Hanne, Eds. Cham: Springer International Publishing, 2015, pp. 383–397.
38. B. Ghotra, S. McIntosh, and A. E. Hassan, “Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, May 2015, vol. 1, pp. 789–800, doi: 10.1109/ICSE.2015.91.
39. P. S. Kumar, A. K. K, S. Mohapatra, B. Naik, J. Nayak, and M. Mishra, “CatBoost Ensemble Approach for Diabetes Risk Prediction at Early Stages,” in *2021 1st Odisha International Conference on Electrical Power Engineering, Communication and Computing Technology(ODICON)*, Jan. 2021, no. vi, pp. 1–6, doi: 10.1109/ODICON50556.2021.9428943.
40. A. Okutan and O. T. Yıldız, “Software defect prediction using Bayesian networks,” *Empir. Softw. Eng.*, vol. 19, no. 1, pp. 154–181, Feb. 2014, doi: 10.1007/s10664-012-9218-8.