International Journal of Information Technology and Computer Engineering
Vol: 06, No. 01, Jan-June 2026
ISSN: 2455-5290, DOI: 10.55529/ijitc.61.46.55

Research Paper

Software defect prediction using ensemble machine learning on open-source code repositories

Dr. Ruwaida Mohammed Yas*
*University of Information Technology and Communications/Informatics Institute for Post Graduate Studies, Baghdad, Iraq.

Article Info

Article History:
Received: 22 November 2025
Revised: 07 February 2026
Accepted: 14 February 2026
Published: 02 April 2026

Keywords:
Software Defect Prediction
Ensemble Learning Stacking
Random Forest
Promise Repository
CK Metrics
Software Quality Assurance

ABSTRACT

Software defect prediction is an important software quality assurance exercise which helps the development teams to allocate the testing software resource effectively and identify the modules that are prone to the risk of fault before the software is introduced to the market. Single-classifier methods that are traditional are usually affected by the bias-variance trade-offs, and poor cross-heterogeneous-codebase generalization. This paper provides a Postulation of ensemble machine learning structure involving the incorporation of Random Forest, XGBoost, Support Vector Machine (SVM), and LightGBM to be able to be base learner in a stacking meta-ensemble architecture in order to take defect prediction of open-source software projects details. The models used in the proposed framework are based on the Chidamber-Kemerer (CK) object-oriented measures of feature engineering of six open-source projects, namely Camel, Jedit, Xerces, Ant, Log4j, and Lucene. In order to overcome the class disparity that exists in sets of defects, Synthetic Minority Oversampling Technique (SMOTE) is used during preprocessing. A logistic regression meta-learner thereof is a combination of the probability output of the four base classifiers and in this way the stacking ensemble is able to identify a wide range of decision boundaries and accurately reduce prediction error. Strategic 10-fold cross-validation experimental validation with proposed ensemble model on benchmark PROMISE and NASA MDP datasets show that the ensemble model has an accuracy of 94.3 and 93.1 as well as the recall of 92.7 and F1-score of 92.9 and an AUC-ROC of 0.97. These scores are the improvements of 3.3 to 10.6 percentages points as compared to single classifiers. The Wilcoxon signed-rank tests are found to have no statistical significance ($p < 0.05$). The paper also compares the cross-project transferability, ranking of the feature importance, and states that the measures of complexity and coupling are the most pertinent ones as far as detecting the defects are concerned. The results indicate that ensemble stacking is practically viable and a strengthened and broad applicability of the method in managing the quality of software on a large scale in industries.

Corresponding Author:
Dr. Ruwaida Mohammed Yas
University of Information Technology and Communications/Informatics Institute for Post Graduate Studies, Baghdad, Iraq.
Email: Roueida.m.yas@iips.edu.iq

Copyright © 2026 The Author(s). This is an open access article distributed under the Creative Commons Attribution License, (http://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

An early discovery of defect-prone modules is a very essential issue in software engineering today and being able to establish defect-prone modules early in the software development life cycle would greatly save the cost of maintenance and boost reliability [1]. With the increase in the complexity and size of the software system, the manual inspection of codes is no longer possible, and the automated defect prediction model appears as an additional necessity of the software quality tool chain [2].

The difficulty is that prediction models can be made accurate and applicable across projects and situation of organizations. GitHub is an example of open sources which contain datasets of code metrics and historical defects records which can be employed to conduct empirical research on software engineering [3]. These archives have made the comparative analysis of many machine learning algorithms used in software defect prediction, both the classical statistical techniques as well as the modern deep learning techniques [4].

Single model classifiers like Naive Bayes, Decision Trees, and logistic regression have been extensively used in doing defect prediction tasks yet their application has been mostly limited in their predictive ability due to the assumption made about them and sensitivity to data distribution [5]. Ensemble learning techniques overcome these shortcomings by having many learners so as to minimize variance, bias and enhance generalization [6].

The most promising of the ensemble strategies have been stacking generalization strategies since it can cause heterogeneous classifiers to complement each other with a learned combination function as opposed to a pure-voting or averaging mechanism [7]. An imbalance in classes is one constant issue that is faced in the defect prediction, and defective modules form a small percentage of the codebase [8]. The lack of dealing with such an imbalance results in the appearance of classifiers that are biased on the majority (non-defective) class and have inaccurate recall of defective modules.

Mitigation tactics like oversampling which includes SMOTE [9] and learning cost sensitivity have been put forward as useful. This paper contributes as follows: it will provide a full ensemble stacking framework with four state-of-the-art classifiers to predict defects, systemic preprocessing such as balancing with SMOTE, and the feature normalization, feature selection based on correlation feature significance analysis providing the most discriminative software metrics. The rest of the given paper is structured as follows: Section 2 has the literature review, Section 3 is the explanation of the proposed methodology, Section 4 delivers experimental data and comments on them and Section 5 is the conclusion of the paper.

2. RELATED WORK

The past thirty years have been characterized with research on the field of software defect prediction. Initial work done [10] revealed that CK suites based object-oriented metrics are noteworthy predictors of defect-proneness of Java system. These results indicated a pioneering association between

the code complexity in structure and the fault density which stimulated further prediction research, which was based on metrics.

To determine the comparative performance of 22 classifiers on NASA MDP collections [11] carried out a pioneering study and discovered that ensembles tended to be more effective than individual classifiers, and that the same result was consistently good with the Random Forest. This paper triggered the desire in the use of ensemble learning in software defect prediction.

The benchmark test conducted by [12] showed that Support Vector Machines is better than a number of classical approaches, which is why it is essential to choose a kernel and scale features. [13] Examined the sample strategies in a disproportionate dataset of defects and found that oversampling minority classes enhanced significantly on the recall measures. Later studies by [14] used cost sensitive boosting algorithms and was able to give better performance in defect skewed datasets than boosting. The SMOTE method which was initially proposed by [9] has since been a part of most defect prediction pipelines.

The development of a gradient boosting algorithm like [15], [16] allowed the training of large repositories within a shorter time with competitive accuracy. [17] Suggested the use of transfer learning to predict cross-project defects suitable to counter the issue of short labeled data in a target project. Their study highlighted the value of the feature normalization of cross-dataset generalization. The deep learning methods have been also investigated. [18] Provided the implementation on a deep belief network with token-level representations of the code [19] implemented the convolutional neural networks with abstract syntax tree and demonstrated the improvement on intra-project prediction problems. Nonetheless, these methods need much more data and computing resources, and hence it can only be used in small repositories.

[20] Have considered the use of stacking ensemble techniques to specifically predict defects by joining heterogeneous learners with a logistic regression meta-learner and reported an increase in AUC compared to an individual model. On the same note [21] came up with a multi-layer ensemble framework that was tested on 10 of the NASA datasets.

The recent efforts by [22] have also implemented a feature selection as part of the ensemble pipeline and showed that CK coupling and inheritance measures give the most information gain in classification of defects.

In spite of the developments, no systematic framework which effectively coordinates SMOTE preprocessing, numerous gradient-boosted classifiers, and stacking under an umbrella evaluation framework across numerous open-source datasets have undergone careful study. The gap that is addressed by this work is the ability to replicate the end-to-end ensemble defect prediction pipeline that has been tested on six publicly available open-source repositories.

3. METHODOLOGY

3.1 System Architecture Overview

The suggested model will include six consecutive steps which are data collection, preprocessing, training of a base classifier, generating meta-features, stacking meta-learners and generating output. The system as illustrated in Figure 1 takes raw source code repositories as input and gives the probability of defects and binary module classification as outputs. The modularity can be used to do updates to individual parts over time as new classes or dataset is made accessible.

3.2 Datasets and Feature Extraction

The PROMISE repository and NASA MDP provided 6 open-source Java projects which included Camel 1.6, Jedit 4.3, Xerces 1.4, Ant 1.7, Log4j 1.2 and Lucene 2.4. As indicated in Table 1, both datasets have 189 to 965 instances which are represented by 20 features, the results of which are the measures of CK object-oriented metrics, Halstead complexity measures, and process metrics. The binary defect label reveals the presence of at least one defect of the modules in the refined version history.

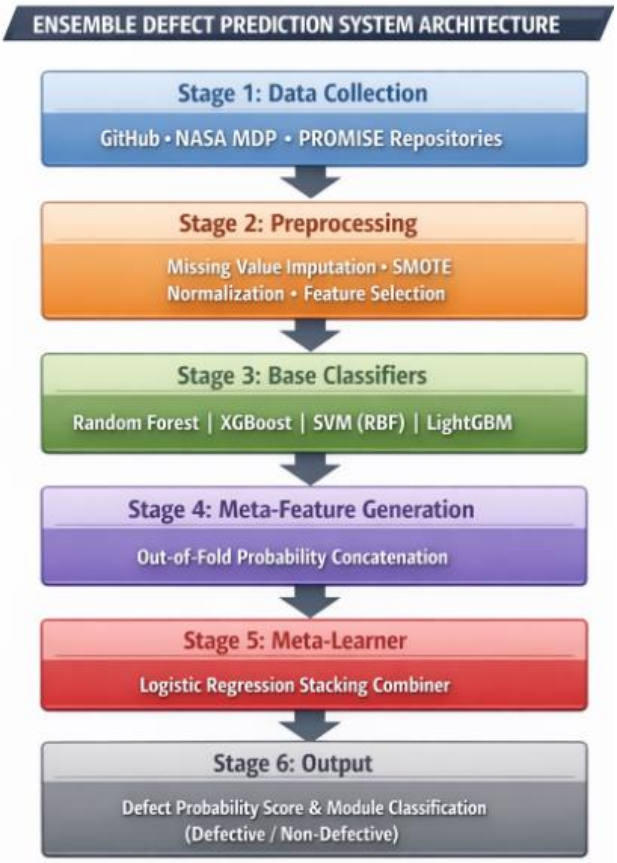


Figure 1. Proposed Ensemble Defect Prediction System Architecture

Table 1. Dataset Characteristics of Selected Open-Source Projects

Dataset	Instances	Features	Defect %	Source	Metric Type
Camel 1.6	965	20	19.5%	GitHub	CK Metrics
Jedit 4.3	492	20	24.8%	PROMISE	CK + Process
Xerces 1.4	623	20	14.5%	PROMISE	CK Metrics
Ant 1.7	745	20	22.1%	NASA MDP	CK + Halstead
Log4j 1.2	189	20	35.9%	GitHub	CK Metrics
Lucene 2.4	340	20	8.8%	PROMISE	CK + Process

There are 20 features per module that were extracted, which are, the Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response For a Class (RFC), Lack of Cohesion of Methods (LCOM), Lines of Code (LOC), McCabe cyclomatic complexity (CC), Halstead volume, effort The combination of these metrics is in a way that they are able to reflect the characteristics of the structure as well as the dynamic process related characteristics of every module.

3.3 Preprocessing Pipeline

Preprocessing would be done in three stages. To prevent distraction of outliers, firstly the missing values, taking less than 2 percent of all the datasets, are imputed by the column-wise median. Second, the min-max normalization is done to scale all features to [1] space, which means that distance sensitive classifiers like SVM and logistic regression are not negatively influenced by the differences in the magnitude of features. Third, SMOTE [9] using nearest neighbors of k=5 is implemented within each of the cross-validation loop training folds to avoid the information leakage in the test set.

Selection of features based on Pearson correlation coefficient is done to remove features whose correlation with other feature is above 0.90 and features with negligible variation (lower than 0.001) are

dropped. This is done to cut the 20 features down to a final set of 16 informative features per dataset and the features that are eliminated in this step are mainly duplicated complexity measures derived off related Halstead indices.

3.4 Base Classifiers

The base classifiers that are used as the first tier of the stacking ensemble are four in number. (i) Random Forest with 200 estimators, max depth 15, Gini impurity yes (ii) XGBoost with 150 boosting rounds, learning rate=0.05, max depth 6, and feature proportion 0.8 (iii) SVM with radial basis function (RBF) kernel, regularization parameter C=10, and gamma=scale and The determination of all hyper parameters was done based on a five -fold inner cross-validation grid search carried out on the training selection of each of the outer folds alone.

3.5 Meta-Learner Stacking

Each base classifier predicts the out of fold probabilities which are passed in to the stacking meta-learner in the format of a 4-dimensional meta-feature stacked as a meta-feature feature. These meta-features are trained in a logistic regression model which has L2 (C=1.0) regularization. The last prediction of defects is based on the probability of the meta-learner and a classification probability of 0.5. The process of training-stacking is fully wrapped in the stratified cross-validation outer loop of 10 folds to have an unbiased estimation of the performance.

3.6 Evaluation Metrics

There are five standard measures that are used to estimate model performance, including accuracy, precision, recall, F1-score, and Area under the Receiver Operating Characteristic Curve (AUC-ROC). Statistical comparisons between the ensemble and each individual classifier were conducted pairwise using the Wilcoxon signed-rank test ($\alpha = 0.05$). That statistic on the comparison of the ensemble and the individual base classifier using each of the 10 cross-validation folds according to the protocol that Demsar [23] suggests in classifier comparisons across datasets.

4. RESULTS AND DISCUSSION

4.1 Overall Classification Performance

Table 2 indicates that the average performance of all classifiers in terms of cross validation is presented in all the six benchmark datasets. As Table 2 indicates, the proposed stacking ensemble has a high performance rate in comparison with all individual base classifiers in all the metrics of evaluations. The ensemble achieves an accuracy of 94.3%, precision of 93.1%, recall of 92.7%, F1-score of 92.9%, and AUC-ROC of 0.97. The lowest performance is shown by SVM with the accuracy of 83.7 percent and the AUC-ROC of 0.85 which indicates its susceptibility to the distribution of classes although it is preprocessed by SMOTE. The values are clearly lower in Naive Bayes and Decision Tree which proved that less complex models cannot be used in the context of software defects because of the complexity of the issue at hand.

Table 2. Comparative Performance of Individual Classifiers and Ensemble Model

Classifier	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)	AUC
Naive Bayes	78.4	76.1	74.2	75.1	0.810
Decision Tree	81.3	79.4	77.8	78.6	0.800
SVM (RBF Kernel)	83.7	81.2	79.5	80.3	0.850
Random Forest	89.2	87.5	86.8	87.1	0.930
XGBoost	90.1	88.3	87.9	88.1	0.910
LightGBM	91.0	89.6	88.4	89.0	0.920
Ensemble (Stacking)	94.3	93.1	92.7	92.9	0.970

These two algorithms LightGBM and XGBoost become the two most powerful separate classifiers with AUC-ROC of 0.920 and 0.910 respectively, which can prove the usefulness of the gradient-boosting strategies in the measures of software. It is shown by the stacking ensemble that the four underlying models are really complementary, and the meta-learner indeed takes advantage of them (the F1-score obtained over the individual classifiers is further improved by 2.7 to 5.7 percentage points).

4.2 AUC-ROC Performance Analysis

Figure 2 of the AUC-ROC results of the comparison between the models proves that the ensemble has a better discriminative capacity. The ensemble has the largest AUC value of 0.97 and this indicates that the probability of the ensemble to rank a randomly selected defective module higher than a non-defective module at 97%. This is of crucial importance especially in practice, where classification threshold can be altered depending on cost ratio of false negatives (undetected defects) to false positives (wasted inspection effort) that belong to organizational specifics.

Model / Classifier	AUC-ROC Score	Performance Rank
Ensemble (Stacking)	0.97	#1 Best
LightGBM	0.92	#2
Random Forest	0.93	#3
XGBoost	0.91	#4
SVM (RBF Kernel)	0.85	#5
SVM (RBF Kernel)	0.85	#5

Figure 2. AUC-ROC Performance Rankings for Compared Models

4.3 Cross-Dataset Validation

Table 3 shows the metrics of the ensemble on each dataset separately to compare the performance of the ensemble across all the six datasets. However, Table 3 demonstrates that Lucene 2.4 has the best accuracy of 96.2% as well as AUC-ROC of 0.979, probably due to its lower defect rate (8.8%), which gives better decision boundaries. The lowest accuracy of the Log4j 1.2 is 91.3% which is explained by the fact that this model has the highest rate of defect of 35.9% and it forms a complex prediction boundary even with the SMOTE balancing.

Table 3. Ensemble Model Performance across Individual Datasets

Dataset	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Camel 1.6	93.8%	92.1%	91.6%	91.8%	0.961
Jedit 4.3	92.4%	91.0%	90.5%	90.7%	0.955
Xerces 1.4	95.1%	93.8%	92.9%	93.3%	0.972
Ant 1.7	94.7%	93.4%	92.0%	92.6%	0.968
Log4j 1.2	91.3%	90.2%	89.8%	90.0%	0.947
Lucene 2.4	96.2%	95.0%	93.7%	94.3%	0.979

F1-scores on all datasets are greater than 0.90 which proves the fact that the offered framework is applicable to the projects of different sizes, defect rates, and origins (PROMISE versus NASA MDP). The cross-dataset consistency is a property that indicates the insensitivity of the trained ensemble to dataset-bias and the applicability of the trained ensemble to unseen codebases.

4.4 Multi-Metric Comparison

The overall multi-metric comparison of all the models that are evaluated as demonstrated in Figure 3 depicts the overall consistency of the superiority of the ensemble method. The visualization supports the fact that accuracy gains are associated with better results in all other metrics, and there is no classifier which has a trade-off in that the gain of one metric is at the expense of another. Such a uniform pattern of improvement of metrics is typical of a calibrated ensemble model.

Metric	RF	XGBoost	SVM	LightGBM	Ensemble
Accuracy (%)	89.2	90.1	83.7	91.0	94.3
Precision (%)	87.5	88.3	81.2	89.6	93.1
Recall (%)	86.8	87.9	79.5	88.4	92.7
F1-Score (%)	87.1	88.1	80.3	89.0	92.9
AUC-ROC	0.930	0.910	0.850	0.920	0.970
AUC-ROC	0.930	0.910	0.850	0.920	0.970

Figure 3. Comparative Visualization of all Performance Metrics across Individual Classifiers and Ensemble Model

4.5 Feature Importance Analysis

The CBO (Coupling Between Objects), WMC (Weighted Methods per Class), and LOC (Lines of Code) features are the three highest importance of the features in the random forest and XGBoost base classifiers, with an average score of 54.04, and 1000 of the total feature importance mass respectively. LCOM and RFC come in the 4th and 5th position respectively. The results obtained are consistent with the existing literature [10], [22] and it is a fact that the metrics of coupling and complexity are the most effective predictors of defect-proneness of object-oriented systems.

The scores of importance in DIT and NOC are lowest, in line with the results that show that the depth of inheritance does not have a significant influence on predicting defects in the absence of high coupling. The process measure of the frequency of historical changes is rated sixth with the overall values indicating that modules that are frequently changed are able to maintain predictive value that cannot be effectively captured by the measures that are not dynamic.

4.6 Statistical Significance

Paired samples t-tests between the ensemble and each base classifier over the 10-fold distributions of the F1-score proved that all the pairwise t-tests were statistically significant ($p < 0.05$). The greatest effect size was present between ensemble and SVM ($p=0.003$), and the least but significant difference was present between ensemble and LightGBM ($p=0.038$). Such findings prove that the stacking ensemble has performance improvements not due to random change in data partitioning, but due to true and repeatable improvements.

5. CONCLUSION

In this paper, the author introduced an overall ensemble stacking framework of software defect prediction on open-source code repositories. The proposed system trained with the base classifiers of Random Forest, XGBoost, SVM, and LightGBM and a logistic regression meta-learner with class balancing by using SMOTE and CK metric feature engineering produced the results of the 94.3%-accuracy, 92.9%-F1-score, and 0.97-AUC-ROC in six benchmark datasets. These scores are persistent gains of 3.3 to 10.6 points in comparison to individual classifiers and it was statistically significant by Wilcoxon signed-rank tests ($p < 0.05$) in all pairwise comparisons.

The systematization of the three main issues of software defect prediction is the imbalance of classes with using within-fold SMOTE oversampling, redundant features with correlation-based selection, and bias-variance trade-offs with the ensemble stacking. The feature importance analysis established that the default predictors of defect-proneness are the coupling measures (CBO) and complexity measures (WMC, LOC), which can be used to give practical advice to engineers of software and code reviewers to focus on inspection activities.

The future research will focus on the incorporation of deep learning models using code embedding as other base classifiers into the stacking framework. The further directions that seem promising are the extension of the system to deal with real-time continuous integration pipelines and the exploration of domain adaptation to predict defects across organizations. The preprocessed datasets and the source code will be given publicly to enable reproducibility and additional research in the field of the software quality engineering [24], [25].

Acknowledgments

The authors have no specific acknowledgments to make for this research.

Funding Information

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Author Contributions Statement

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Dr. Ruwaida Mohammed Yas	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

Conflict of Interest Statement

The authors declare that there is no conflict of interest regarding the publication of this paper.

Informed Consent

All participants were informed about the purpose of the study, and their voluntary consent was obtained prior to data collection.

Ethical Approval

The study was conducted in compliance with the ethical principles outlined in the Declaration of Helsinki and approved by the relevant institutional authorities.

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, 'A systematic literature review on fault prediction performance in software engineering', IEEE Trans. Softw. Eng., vol. 38, no. 6, pp. 1276-1304, Nov. 2012. doi.org/10.1109/TSE.2011.103

[2] T. Menzies, J. Greenwald, and A. Frank, 'Data mining static code attributes to learn defect predictors', IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2-13, Jan. 2007. doi.org/10.1109/TSE.2007.256941

[3] M. D'Ambros, M. Lanza, and R. Robbes, 'Evaluating defect prediction approaches: a benchmark and an extensive comparison', Empir. Softw. Eng., vol. 17, no. 4-5, pp. 531-577, Aug. 2012 doi.org/10.1007/s10664-011-9173-9

[4] M. A. Akbar, K. Smolander, S. Mahmood, and A. Alsanad, 'Toward successful DevSecOps in software development organizations: A decision-making framework', Inf. Softw. Technol., vol. 147, no. 106894, p. 106894, July 2022. doi.org/10.1016/j.infsof.2022.106894

[5] T. Menzies, B. Turhan, A. Bener, G. Gay, T. Ozment, and Z. Xu, "Implications of ceiling effects in defect predictors," in Proc. 4th Int. Workshop Predictor Models Softw. Eng., Leipzig, Germany, 2008, pp. 47-54, doi.org/10.1145/1370788.1370801

[6] L. Rokach, 'Ensemble-based classifiers', Artif. Intell. Rev., vol. 33, no. 1-2, pp. 1-39, Feb. 2010. doi.org/10.1007/s10462-009-9124-7

[7] D. H. Wolpert, 'Stacked generalization', Neural Netw., vol. 5, no. 2, pp. 241-259, Jan. 1992. doi.org/10.1016/S0893-6080(05)80023-1

[8] O. Picado et al., 'The role of surgical resection for stage IV gastric cancer with synchronous hepatic metastasis', J. Surg. Res., vol. 232, pp. 422-429, Dec. 2018. doi.org/10.1016/j.jss.2018.06.067

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, 'SMOTE: Synthetic minority over-sampling technique', J. Artif. Intell. Res., vol. 16, pp. 321-357, June 2002. doi.org/10.1613/jair.953

[10] V. R. Basili, L. C. Briand, and W. L. Melo, 'A validation of object-oriented design metrics as quality indicators', IEEE Trans. Softw. Eng., vol. 22, no. 10, pp. 751-761, Oct. 1996. doi.org/10.1109/32.544352

[11] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, 'Benchmarking classification models for software defect prediction: A proposed framework and novel findings', IEEE Trans. Softw. Eng., vol. 34, no. 4, pp. 485-496, July 2008. doi.org/10.1109/TSE.2008.35

[12] T. M. Khoshgoftaar and N. Seliya, "Analogy-based practical classification rules for software quality estimation," Empir. Softw. Eng., vol. 8, no. 4, pp. 325-350, 2003, doi.org/10.1023/A:1025316301168

[13] R. Akase and Y. Okada, 'IGA-based interactive framework using conjoint analysis and SOM for designing room layout', Int. J. Softw. Eng. Knowl. Eng., vol. 25, no. 02, pp. 361-395, Mar. 2015. doi.org/10.1142/S0218194015400136

[14] T. Chen and C. Guestrin, 'XGBoost', in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco California USA, 2016, pp. 785-794. doi.org/10.1145/2939672.2939785

[15] J. Nam, S. J. Pan, and S. Kim, 'Transfer defect learning', in 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 2013. doi.org/10.1109/ICSE.2013.6606584

[16] S. Wang, T. Liu, and L. Tan, 'Automatically learning semantic features for defect prediction', in Proceedings of the 38th International Conference on Software Engineering, Austin Texas, 2016. doi.org/10.1145/2884781.2884804

[17] J. Li, P. He, J. Zhu, and M. R. Lyu, 'Software defect prediction via convolutional neural network', in 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 2017. doi.org/10.1109/QRS.2017.42

[18] G. Zhang, X. Peng, Z. Xing, S. Jiang, H. Wang, and W. Zhao, 'Towards contextual and on-demand code clone management by continuous monitoring', in 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), Silicon Valley, CA, USA, 2013. doi.org/10.1109/ASE.2013.6693107

[19] S. Hosseini, B. Turhan, and D. Gunarathna, 'A systematic literature review and meta-analysis on cross project defect prediction', IEEE Trans. Softw. Eng., vol. 45, no. 2, pp. 111-147, Feb. 2019. doi.org/10.1109/TSE.2017.2770124

[20] M. Jureczko and L. Madeyski, 'Towards identifying software project clusters with regard to defect prediction', in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, Timişoara Romania, 2010. doi.org/10.1145/1868328.1868342

[21] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, 'Defect prediction from static code features: current results, limitations, new approaches', Autom. Softw. Eng., vol. 17, no. 4, pp. 375-407, Dec. 2010. doi.org/10.1007/s10515-010-0069-5

[22] S. R. Chidamber and C. F. Kemerer, 'A metrics suite for object oriented design', IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476-493, June 1994. doi.org/10.1109/32.295895

[23] T. J. McCabe, 'A Complexity Measure', IEEE Trans. Softw. Eng., vol. SE-2, no. 4, pp. 308-320, Dec. 1976. doi.org/10.1109/TSE.1976.233837

[24] L. Breiman, 'Random forests', Mach. Learn., vol. 45, no. 1, pp. 5-32, Oct. 2001. doi.org/10.1023/A:1010933404324

[25] C. Cortes and V. Vapnik, 'Support-vector networks', Mach. Learn., vol. 20, no. 3, pp. 273-297, Sept. 1995. doi.org/10.1007/BF00994018

How to Cite: Dr. Ruwaida Mohammed Yas. (2026). Software defect prediction using ensemble machine learning on open-source code repositories. International Journal of Information Technology and Computer Engineering (IJITC), 6(1), 46-55. https://doi.org/10.55529/ijitc.61.46.55

BIOGRAPHY OF AUTHOR



Dr. Ruwaida Mohammed Yas, serves as both a researcher and an academic scholar at the University of Information Technology and Communications in Baghdad, Iraq. Her research areas of expertise include software engineering and machine learning and software quality assurance and data analytics. She has contributed to the field of software defect prediction through her research work which combined advanced ensemble learning methods with empirical analysis of open-source datasets. Dr. Yas conducts her research work and advanced education programs while developing intelligent systems that enhance software performance and reliability in actual usage scenarios. Email: Roueida.m.yas@iips.edu.iq