



Optimized ensemble machine learning model for software bugs prediction

Femi Johnson¹ · Olayiwola Oluwatobi² · Olusegun Folorunso¹ · Alomaja Victor Ojumu³ · Alatishe Quadri²

Received: 4 May 2022 / Accepted: 22 November 2022 / Published online: 3 December 2022
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2022

Abstract

Software accuracy and efficiency checks are becoming of paramount interest to system users before utilization. As a result, twenty-first-century programmers are consciously developing less buggy, highly efficient, and robust software with a higher degree of accuracy. Occasionally, undetected bugs in large software due to the complexity of codes and other associated parametric attributes cause hardware to malfunction. In this paper, an ensemble model of Logistic Regression and Extra tree classifier algorithms is deployed on parametric software attributes for the accurate classification and prediction of software bugs. The implementation was performed on different platforms (WEKA, MATLAB and PyCharm) to determine the rate of memory utilization, optimize prediction time, maximize the model's efficiency and compare accuracy rankings among similar machine models. A publicly available software defects dataset from the National Aeronautics and Space Administration (NASA) containing 16,962 instances and 38 attributes for software defects prediction was collected, pre-processed and used in the implementation of this study. The collected data were vectorized, subjected to principal component analysis (PCA) for dimension reduction based on ranking values and divided in the ratio 3:2 for training and testing of the ensemble model classifier, respectively, on new sets of buggy software datasets. The result from the ensemble model showed a significant increase from 96.7–97.8% in the prediction accuracy of the un-vectorized dataset to vectorized dataset. An appreciable decrease in the prediction time (19.7 s) of the vectorized dataset was also observed against the initial time (26.9 s) recorded for the un-vectorized dataset. In addition, memory utilization for vectorized datasets increased during the training phase due to the number of bits but got reduced at the final testing phase of the software bug prediction. However, the overall accuracy of 97.8% recorded by the optimized ensemble model for buggy software prediction proved the model's capability to accurately classify and predict buggy software with efficient memory utilization at optimal time duration.

Keywords Software · Bugs detection · Vectorization · Logistic regression · Extra tree classifier

1 Introduction

The use of software has grown exponentially over the years due to the increase in the demand for automated and efficient systems (electronic gadgets). Embedded software in digital and electronic systems has helped to minimize faults and

make tasks easier and faster with little or no human intervention. The benefits derived from the utilization of accurate and efficient software cannot be achieved without the utilization of software bug detectors.

The role of software bug predictors cannot be over-emphasized [1, 2] in software quality assurance [3, 4]. In recent years, researchers have implemented several machine learning techniques for the prediction of software defects [5–7]. Common models have combined variants of Artificial Neural Networks (ANNs) with regression or classification models [8, 9]. Recent studies also revealed that the use of sub-optimal default parameter settings degrades the efficiency of these models over time [10–13], resulting in the high consumption rate of computing resources (memory and time) and reducing the accuracy of software bug predictors [14]. As the prediction time and overall cost of software tests increase

✉ Femi Johnson
femijohnson123@hotmail.com

¹ Department of Computer Science, Federal University of Agriculture, Abeokuta, Nigeria

² Department of Computer Science, Olabisi Onabanjo University, Ago-Iwoye, Nigeria

³ Department of Computer Technology, Yaba College of Technology, Lagos, Nigeria

[15, 16], the primary challenge for every researcher and software developer is to improve the quality of software with limited testing resources through vectorization (a technique adopted by CPU manufacturers to improve the performance of modern computers) [17, 18].

Vectorization is a technique for transforming an algorithm from operating on a single or unitary value to multiple values (vectors) per time unit [19]. It has been adopted to speed up program execution, make programs more efficient and readable. In this paper, an optimized ensemble machine learning model (Logistic Regression and Extra Tree classifier) is implemented at the learning and testing phases to address the limitations of single machine learning techniques [20] to generate a better result, eliminate problems arising from imbalanced classes using a vectorization technique for faster execution, and maximize hardware resources.

The organization of this paper is further divided as follows. Severity-based classification and software bug debugging priorities are explained, immediately followed by a literature review of recent research work in Sect. 2. Section 3 presents the methodology for developing the optimized ensemble machine learning model for software bugs prediction. Implementation and test results derived from the developed optimized ensemble model's evaluation and analysis are documented in Sect. 4, the discussion and conclusion are given in Sect. 5 accordingly.

2 Related works

Large software project development requires modularization for easier and faster implementation, especially when two or more programmers are involved [15, 21]. After the compilation of the various modules, the integration of software bug predictor into various software modules [11, 22, 23] becomes rigorous and time-consuming. Thus, the inclusion of bug predictors at the earliest stage of software development is necessary. Software bug predictors (SBPs) aid in the identification and detection of bugs [24] by using collected software metrics as parameters [25, 26]. Metrics used include comment line codes [27], modularized codes, web-related metrics for web-based programs [21], network metrics [28], cascading style metrics [29], and other metrics based on mutations that have been identified for measuring software quality [30–32]. Software bugs appear with varied levels of severity and debugging priorities as shown in Table 1. These severity levels are also categorized as:

- a. **Critical:** At this stage, a system's overall performance is impacted by the software flaw. There is no workaround for it. It might also lead to the software modules' complete failure or unsuccessful installation

Table 1 Software bugs debugging priorities

S/n	Debugging priority	Description
1	Low	The defect needs to be repaired, but the repair can be deferred until more serious defects have been fixed
2	Medium	The defect could be resolved in the normal course of development activities. It may wait until a new build or version is created
3	High	The defect must be resolved immediately because it affects the application or the product severely and makes it unusable

- b. **Major:** A software bug is categorized as having a major severity level if it has an impact on crucial data or major computer system functions. It happens when a module's malfunction prevents other modules from operating effectively, which lowers the system's performance.
- c. **Minor:** A few minor functions or non-critical data are affected. It has an easy workaround. A defective non-functional feature in a module could be improved to function in another module of the same software.
- d. **Trivial:** The functionality of the system or its data is not affected. The system's efficiency is not affected, but it may merely cause a form of inconvenience to the user.

The impact of bugs in software is very challenging, and the cost of fixing the bugs varies significantly with the number of bugs and the time of detection [3, 20, 26]. Detecting bugs at an early stage of development would significantly lower the cost incurred in fixing them [27, 33].

Improving software defect prediction accuracy in recent times involves the combination of two or more techniques known as the ensemble or hybrid approach [6, 8, 34]. The systematic review of literature conducted on the forty-six most relevant papers published by four well-organized libraries [35], including IEEE and Springer, from 2012–2021 led to the discovery of WEKA as the most widely used platform for machine learning and the listing of frequently used evaluation metrics in software bug evaluation metrics to include *F*-measure, Area under curve (AUC), Recall, and Precision score. Rarely used ensemble approaches were identified, including Extra Tree, voting, and stacking.

Authors [36] implemented and tested a genetic algorithm-based optimization model for the prediction of software defects on six attributes of the NASA dataset. They formulated two hypotheses and four linear equations to justify their claim on the need for optimization techniques. The model was implemented on MATLAB and the evaluation metric was

simply performed on accuracy and sensitivity level which yielded an average value of 95–97%.

Yi et al. [37] proposed an automated machine learning model for the automatic reconstruction of buildings using photogrammetric and panchromatic images derived from satellite images. Their approach adopted a combination of machine learning models to filter out non-building-related images, building images were refined with edged-vectorized and conditional Corner Generative Adversarial Networks (CGAN). Collected images were trained and tested on Worldview-1 data from Berlin, Germany. Standard metrics used for qualitative evaluation of their model include normalized Median Absolute Deviation (NMAD), Mean Absolute Error (MAE) and Root Mean Square.

Considering the deployment of a hybrid approach to software defects prediction, Mohd and Mohd [6] applied principal component analysis (PCA) for the appropriate feature selection and time complexity reduction in a developed support vector machine model to predict and detect software defects on PROMISE (a collection of three software defect dataset from NASA). The model was implemented on the Spyder program platform. The accuracy values recorded were claimed to be improvements over the previous ones.

Yuzhou [19] implemented a vectorized deep-learning approach for enhancing malware detection. A combination of two networks, the Convolution and Re-Current Neural Networks, was fused to achieve better prediction result accuracy. Effective features extracted were done through the semantic and instruction layers of their proposed architecture. Their proposed model named the Glove model represented each opcode sequence as a K -dimensional real vector given in matrix form. The results obtained showed a higher-class accuracy of 77.38%, feature detection of 83.95% and in comparison, with other machine models, generated the highest result of 88.89% to outperform others.

The collective work of people [10] concentrated on the effects of employing biased performance to forecast and measure defective software prediction models. They documented their findings on several software classification criteria in their thorough analysis of more than thirty peer-reviewed studies from 2012–2020. It was found that a popular metric for assessing defective software in prediction experiments is the F_1 -Measure.

Estimating errors in computing systems due to software defects can help improve performance and software assurance levels [36, 38]. A combination of classification techniques (naïve Bayes, decision tree) and Particle Swarm Optimization technique was used in developing a software flaw prophecy system [39]. The system was able to determine the interaction between quality software metrics and defective modules.

The development of a Neuro-Vectorizer for end-to-end reinforcement learning in software development by

researchers [40] proved efficient in overcoming errors caused by compilers while generating loops and determining the number of instruction packs. A designed framework model which supported multiple machine learning models was trained using a dataset containing loops and non-loop functions with about 5,00,000 steps to achieve a better reward in terms of compilation time, execution time, and generated assembly code size. Their model's performance was benched marked with other models including random search, decision tree, and brute-force algorithms. However, their result showed an average of 3% value loss than the brute-force solution.

In a similar research conducted [17] to determine the performance of microprocessors with compilers to efficiently perform parallelism techniques on processes. Available vectorization policies including the Super Wood Level parallelism (SLP) on integer linear programming solver were tested for better performance comparison. A graph vectorized formulation was performed using Markov Decision Process (MDP) and trained on a Gated Graph Neural Network with 35,635 data functions derived from benchmarked suites (SPEC 2006, SPEC 2017, and NAS). The goSLP vectorized dataset representing 11.2% was used for training and un-vectorized samples were further grouped in the ratio of 4:1 for both training and testing of the model, respectively. Their work showed that the auto-vectorization technique is better adopted for the compiler to improve parallelism within programs.

The proposition of Maria et al. [28] about the quality of software not being judged by a single metric value but an aggregate of software characteristics and jointly distributed metric values led to the development of a probabilistic approach (COPULA) which was trained and tested on a software metric dataset. The model was evaluated and compared with correlation, agreement and distance ranking measurements. The results derived proved the model was accurate and scalable in performance.

Seven clustering approaches including K -means, Expectation Maximization (EM), and Sequential Information Bottleneck (SIB) were evaluated [38] to detect software defects on eight different NASA datasets. The results from their comparison analysis show the varying accuracy level and the initiated a caution on the choice of clustering technique to be adopted in developing software defects predictors.

As different classification techniques become more useful in machine learning for differentiating between buggy software [21] a similar technique was successfully utilized for multi-class classification of online products based on machine learning models (SVM and K -NN) [41]. Exploratory data analysis was performed to identify salient attributes and normalized them. The hashing vectorizer and the term frequency-inverse document frequency (Tf-idf)

vectorizers were evaluated on three test phases (recall classifications, quality and speed text). The accuracy results derived from the evaluation of both models were recorded as 79 and 2% for both SVM and K -NN, respectively.

3 Research methodology

Several machine learning models have been developed from works of literature. The technique adopted in this paper began with the transformation of the collected textual National Aeronautics and Space Administration (NASA) program software dataset to meaningfully explored vectors, trained and tested for predicting and detecting software bugs.

3.1 Data description and modeling

The collected National Aeronautics and Space Administration (NASA) program dataset for software bug prediction comprises fourteen (14) sub-datasets. Each subset contains software parametric attributes as shown in Table 2 and is categorized by name ranging from CM1, JM1, KC1, KC2, KC3, MC1, MC2, MW1, MW2, PC1, PC2, PC3, PC4, to PC5. The data have been formatted in CSV file format and zipped for use. From the available collection of the NASA dataset, the PC5 was selected for use due to the availability of the largest number of data items and a precisely complete dataset with no missing value. The total number of instances available in the PC5 dataset is sixteen thousand nine hundred and sixty-two (16,962) with thirty-eight (38) attributes for prediction.

3.2 Data dimension and class vector representation

A set of independent variables (v_n) associated with each instance in the PC5 dataset is collected to form set a new set (S) as shown in Eq. (1). Two output classes specified for the proposed ensemble model are the defective and non-defective classes represented as β_1 and β_2 , respectively (Fig. 1).

$$S = \begin{Bmatrix} V_{11} & V_{12} & V_{13} & \dots & V_{138} \\ V_{21} & V_{22} & V_{23} & \dots & V_{238} \\ V_{31} & V_{32} & V_{33} & \dots & V_{338} \\ V_{16291} & V_{16292} & V_{16293} & \dots & V_{162938} \end{Bmatrix} \quad (1)$$

where v -independent variable attribute.

3.3 Feature selection and dimension reduction

The principal component analysis (PCA) is used to extract the required features to generate a reduced dataset having a strong relationship with the output from the entire vectorized PC5 dataset to enhance better prediction.

Table 2 Parametric software bugs attributes and numerical range

S/n	Parametric input	Description	Data range
1	Halstead level	Mental effort for software design	0–2
2	Loc blank	Blank lines of code that are available in a program	0–704
3	Node count	The counts of data structures applied at the software development phase	2–1078
4	Cyclomatic complexity	The number of decision counts in software	1–366
5	Multiple condition count	The number of count conditions	0–578
6	Parameter count	The number of variables for passing information between functions	1–500
7	Operator numbers	The number of operators in the software	0–10,862
8	Number of operands	Count of instructions specifying data to be operated on	0–5169
9	Comment percent:	Percentage number of annotations in the source code	0–95
10	LoC code and comment	Available lines of code and comments in a program	0–180
11	Maintenance severity	The degree of software error correction	0–1
12	Call pairs	The number of codes available to both driver and navigator at the same time	0–69
13	Essential complexity	The availability of non-reducible codes, entry points and termination points	1–290

Given the vectorized PC5 dataset with a 38-dimensions vectorized variables (v) and mathematically represented in Eq. (1), a lower-dimensional representation of S called the reduced set $[R]$ is derived using Eq. (2). A total of five attributes derived from the vectorized dataset are used for the prediction of software bugs.

$$R = \{r_1, r_2, r_3, \dots, r_k\}, \text{ with } k \leq n \quad (2)$$

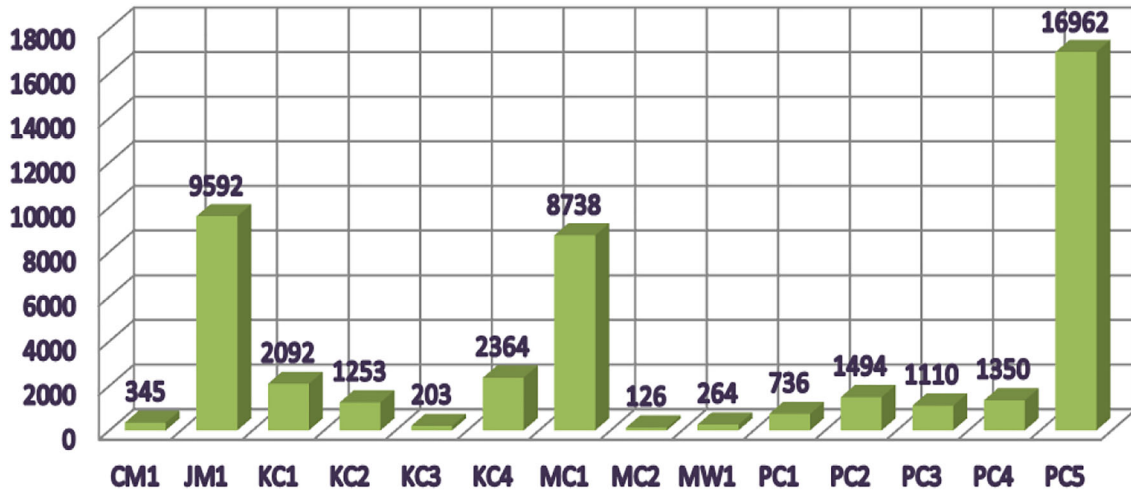


Fig. 1 Instances classification of the NASA dataset

Table 3 PCA selected features for software bugs prediction

S/n	Features	Ranking value
1	EDGE_COUNT	0.218
2	NODE_COUNT	0.217
3	LOC_TOTAL	0.212
4	NUMBER_OF_LINES	0.210
5	LOC_EXECUTABLE	0.210

where R captures the data attributes (v) in the PC5 dataset according to the variance maximization criterion. Elements in R are the hidden components, k is the number of features and n is the number of instances. In this, the PCA maps data from the space of v -variables to a new space of uncorrelated k - variables over the dataset. With the dimension reduction algorithm adopted the best five features were selected. The features and their corresponding rank values are tabulated in Table 3.

3.4 Ensembled software bug predictor flow diagram description

The research flow diagram in Fig. 2 depicts the implementation stages of the ensembled (Logistic regression and Extra tree) classifier model for software bug prediction.

The initial process involves collecting the PC5 software metric dataset. The dataset was vectorized into an array of $M \times N$ matrix where M denotes the initial thirty-eight (38) software variables and N represents a total of 16,962 instances. The vectorized dataset is subjected to PCA for attributes reduction. Thus, yielding five (5) significant software attributes of the pre-processed dataset that was grouped in the ratio 3:2 for the training and testing of the ensemble

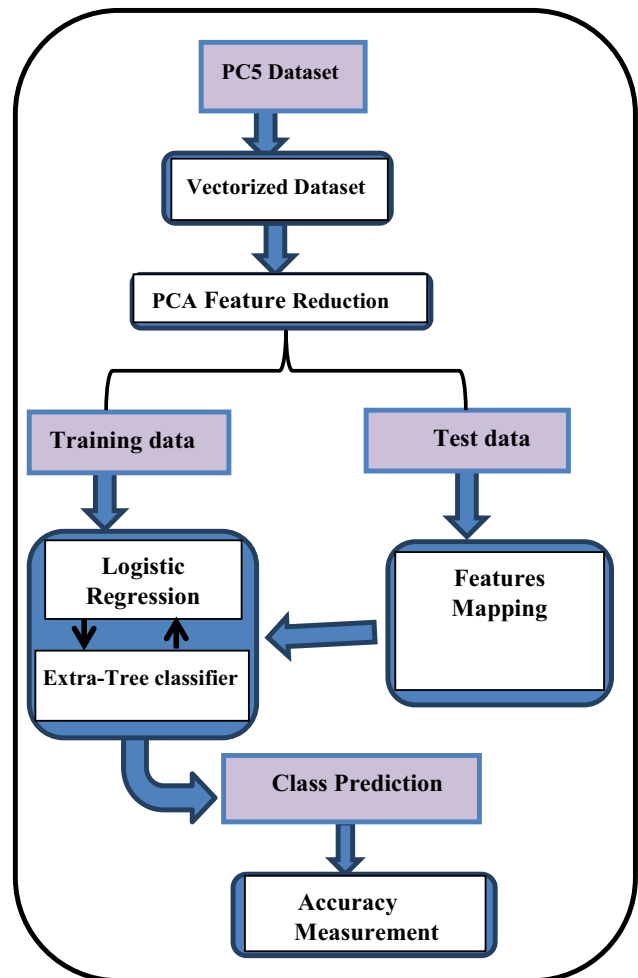


Fig. 2 Optimized Ensemble Software Bugs Prediction flow diagram



Fig. 3 Software bug attributes visualized in WEKA

software bug predictor model. The ensemble model classifier is developed and trained with the sixty percent vectorized data until a well-optimized value of the proposed model is obtained.

3.5 Model's ensembled classifiers training and testing algorithms

The operation of the ensembled software bugs prediction model is dependent on some algorithms. These algorithms are implemented at successive intervals for the effective operation and accurate prediction of the model. Each instance of the dataset is vectorized and converted to an array of $m \times n$ matrix. The PCA algorithm reduced the vectorized array to a 5-dimensional feature array. The ensemble algorithm combining logistic regression and Extra Trees classifiers were further used for the training, testing, prediction and detection of software bugs.

The Logistic regression classifier models the probabilities for the software bugs classification problem with two possible outcomes. Its application is to determine the class (defective, non-defective) of each software based on the available software prediction metrics. The equation representing the logistic regression model is given as:

$$\text{logistic}(\emptyset) = \frac{1}{1 + \exp(-\emptyset)} \quad (3)$$

where \emptyset = the probability of a successful expectation.

In addition, the randomized trees classifier (Extra Tree classifier) combines the result of multiple decision trees collated as a forest for the software defects classification problem. The software bugs prediction is performed by computing the average from the Logistic regression classifier and the majority count from decision trees. Training is done by each tree with the best-selected attributes and split randomly based on predetermined functions.

4 Implementation and results

All algorithms are implemented on an Intel Core™ i5-5020U CPU @ 2.20GHZ workstation with 8 Gb RAM, 1 TB of hard disk drive and installed WEKA, MATLAB and PyCharm software. The implementation stages are:

Loading and fitting the dataset. The NASA dataset is loaded and attributes are perfectly fitted in equivalent rows and columns (Fig. 3).

Exploratory data analysis: Hidden information derived from the NASA data is analyzed using a correlation table. Figure 4 reflects the scatter plot of significant attributes and highly ranked attributes based on their representative value to the dataset.

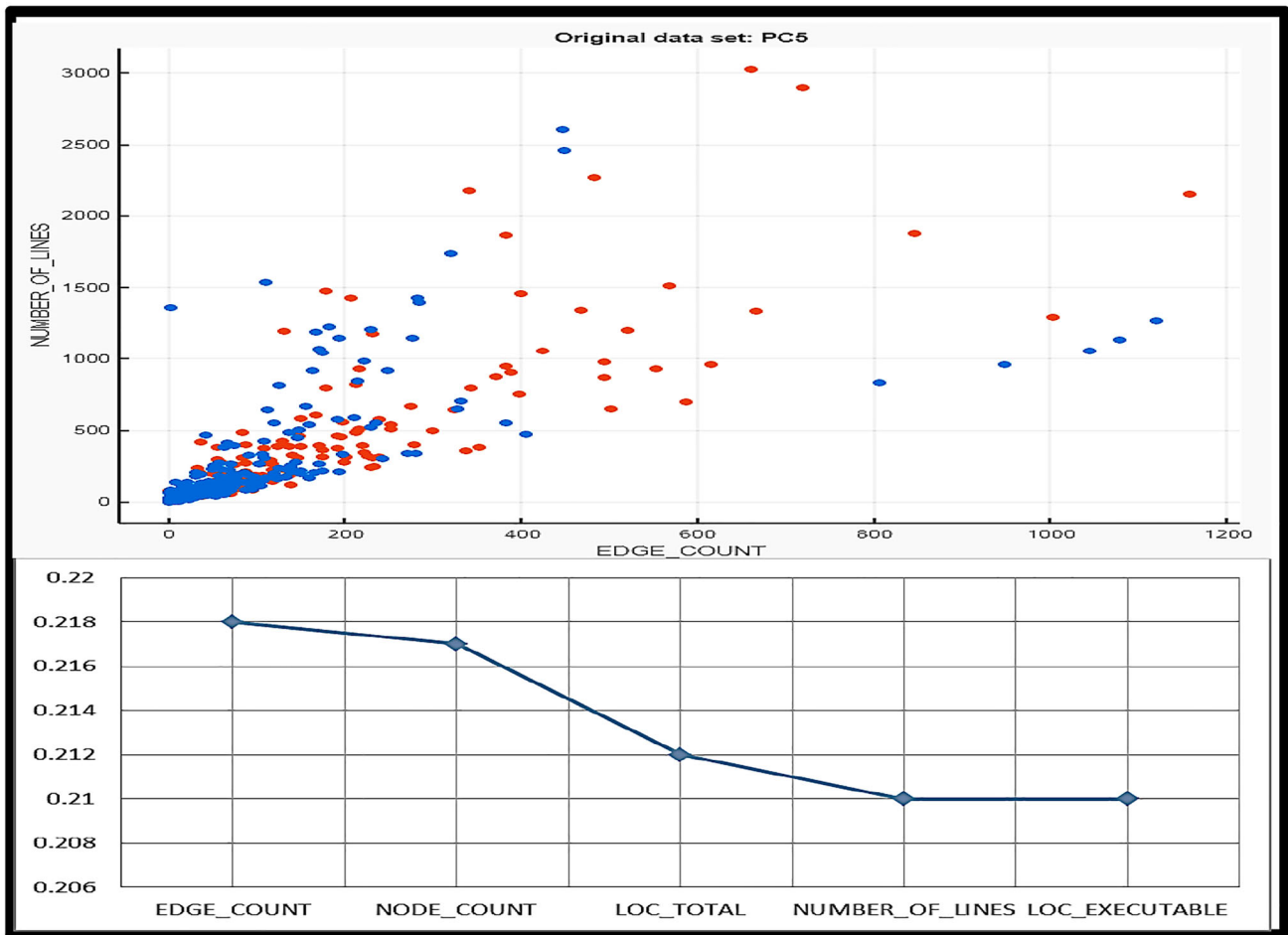
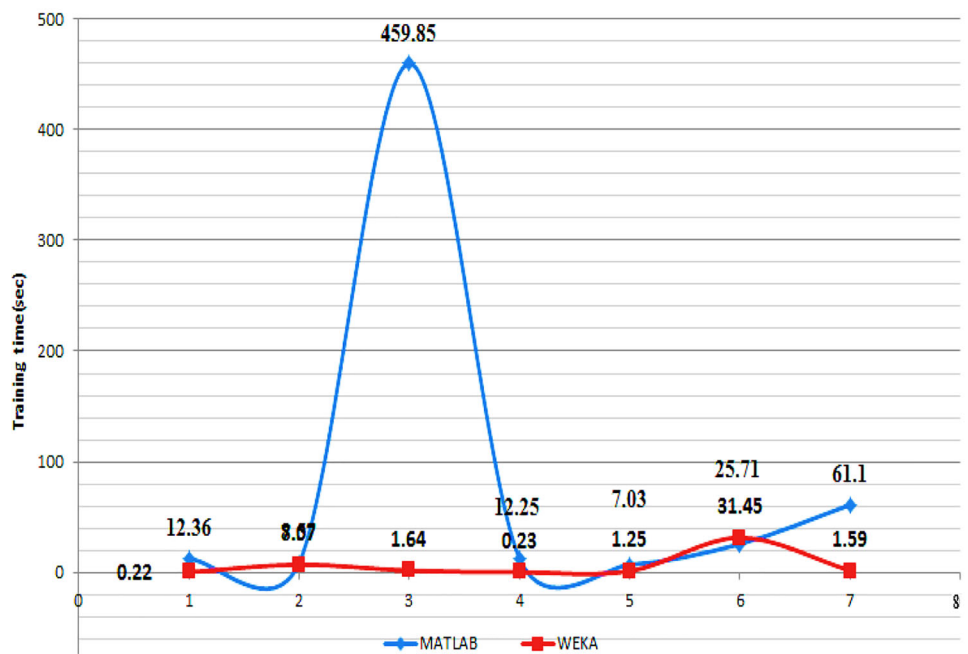


Fig. 4 Scatter diagram and attributes rank spread in the dataset

Fig. 5 Training time plot of the vectorized dataset in both software



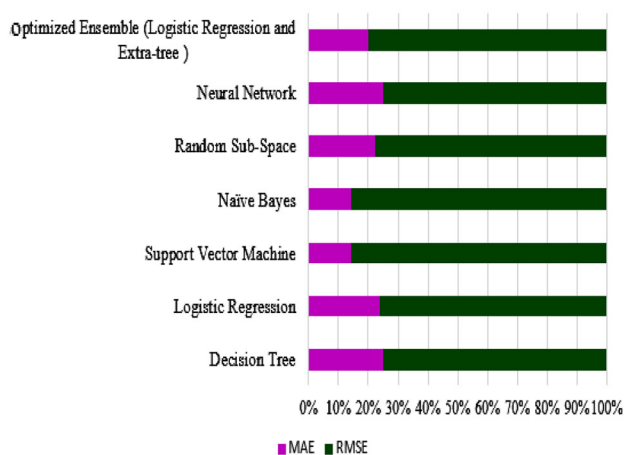


Fig. 6 MAE and RMSE result plot from different machine learning models

Training, testing and model evaluation: Training is performed on sixty percent of the well-prepared standardized dataset using the ensemble model's algorithm at repeated intervals. The holdout data corresponding to forty percent is also tested to determine the model's accuracy, precision, recall and F-measure scores. The mathematical equations for these evaluation metrics are given in Eqs. (4–7), respectively.

$$\text{Accuracy} = ((TP + TN) / (TP + TN + FP + FN)) * 100 \quad (4)$$

$$\text{Precision } (P) = (TP / (TP + FP)) * 100 \quad (5)$$

$$\text{Recall } (R) = (TP / (TP + FN)) * 100 \quad (6)$$

$$F - \text{Measure } (F - \text{Meas.}) = 2 * [(P * R) / (P + R)] \quad (7)$$

where True Positive (TP): Bugs detected by the system, False Negative (FN): Bugs not detected by the system, False Positive (FP): Codes detected as bugs by the system, True Negative (TN): Codes not detected as bugs by the system (Fig. 5).

4.1 Software predictions, result analysis and evaluation

Additionally, automatically generated classification metrics from the two software platforms (WEKA and MATLAB) deployed for the model's simulation include Mean Absolute Error (MAE), Root-Mean-Square Error (RMSE), training, validation and testing durations as shown in Figs. 6, 5, Tables 4 and 5.

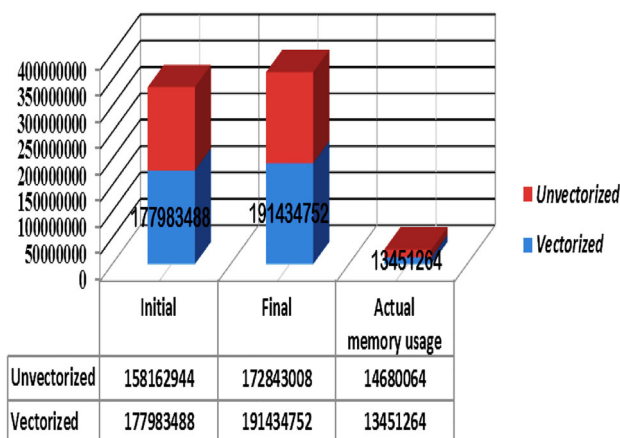


Fig. 7 Memory usage (bytes) for vectorized and unvectorized datasets at testing phase

5 Discussion and conclusion

This study has effectively adopted a non-widely used optimized ensemble learning approach logistic regression and extra tree classifier to classify and predict software bugs on the National Aeronautics and Space Administration (NASA) Metrics Data Program defect dataset with implementation on two different machine learning platforms. The main aim of this approach is to compare the effectiveness of the developed ensemble model on both platforms, discover trade-offs and make a better choice for software selection during software bug prediction modeling.

In addition, the comparative analysis of the results recorded in Tables 4 and 5 showed that the MATLAB-generated values were higher than the values obtained from the WEKA platforms with differences ranging from 0.01–3.7. The highest value (3.7) was obtained in the F-measure score of the tested Logistic regression model. Observable records in Fig. 5 also showed that the highest duration of time (459 s) was utilized at the training phase of the ensemble model on the MATLAB platform. This implied that the training of the model with the software bugs dataset was faster on the WEKA platform than on the MATLAB.

The ensemble model was trained with scalar and vectorized datasets for memory space utilization comparison at both the training and testing phases as shown in Fig. 7. At the testing phase, the actual memory usage for the prediction with the vectorized dataset is lesser than the un-vectorized data. It was noted that there exists a trade-off between memory space utilization and time duration with the use of scalar and vectorized data for software bug predictions.

Furthermore, Fig. 3 displays some software bug attributes as visualized on the WEKA platform. Two classes ($Y = \text{Yes}$ and $N = \text{No}$) were denoted as the output for the classification of the software bug prediction model. If the prediction

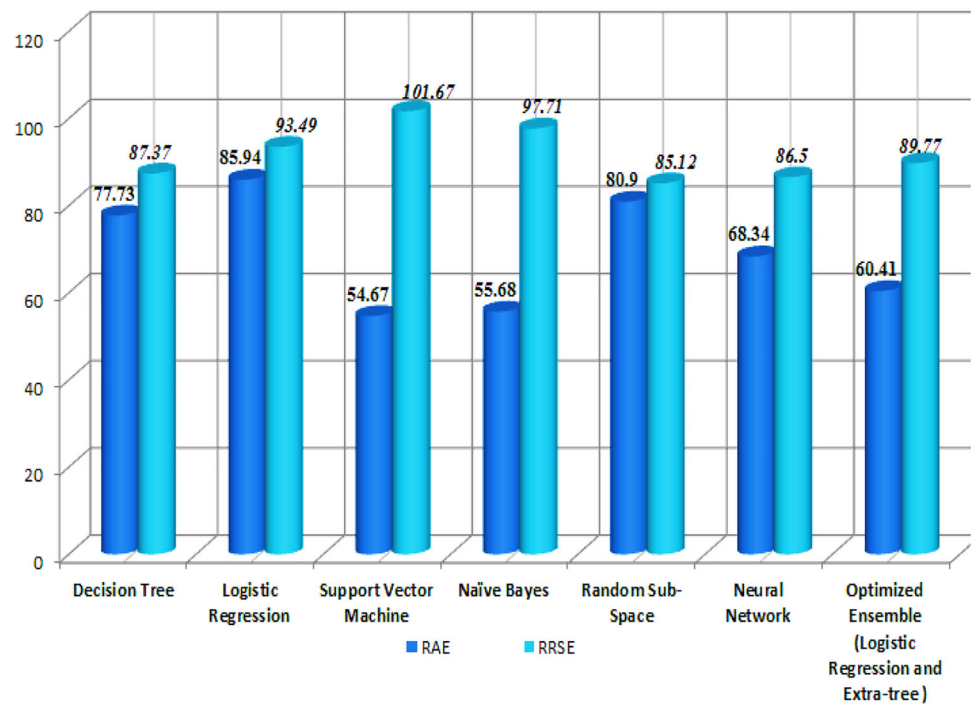
Table 4 WEKA classification metric of machine learning models

S/n	Model	True positive rate	Recall	F-measure	AUC
1	Decision tree	96.8	96.8	98.4	0.91
2	Logistic regression	97.0	97.0	96.0	0.95
3	Support vector machine	96.8	96.8	98.4	0.94
4	Naïve Bayes	96.8	96.8	96.6	0.96
5	Random subspace	97.1	97.1	98.4	0.94
6	Neural network	96.9	96.1	96.6	0.87
7	Optimized ensemble (Logistic Regression and Extra tree)	98.9	97.8	97.5	0.96

Table 5 MATLAB classification metric of machine learning models

S/n	Model	True positive rate	Recall	F-measure	AUC
1	Decision tree	96.3	97.8	98.2	0.89
2	Logistic regression	97.0	97.1	99.7	0.96
3	Support vector machine	97.0	97.1	98.4	0.96
4	Naïve Bayes	96.6	98.1	98.4	0.97
5	Random subspace	97.1	97.2	99.5	0.96
6	Neural network	97.1	97.2	99.4	0.92
7	Optimized ensemble (Logistic Regression and Extra tree)	98.5	97.5	99.7	0.98

Fig. 8 Evaluation results compared to the optimized ensemble model



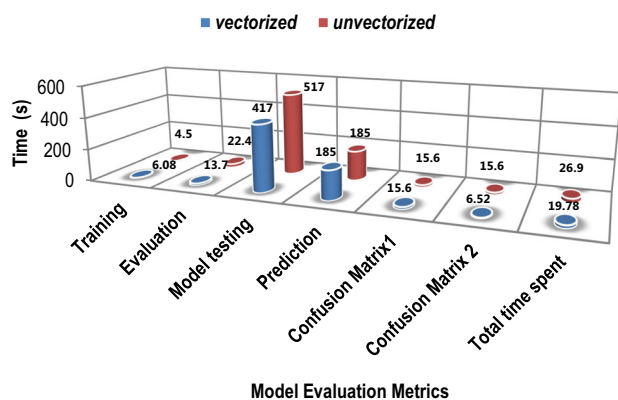


Fig. 9 Time analysis of ensembled optimized with scalar and vectorized dataset

outcome is a “yes” then the tested software is categorized as defective, else it is categorized as non- defective.

Although users may exhibit a few doubts in the choice of software and selection of the best model, there exists a perfect fit (the developed optimized ensemble) model that balances the limitations experienced in the individual model. The results generated by the optimized ensembled model in comparison with other related machine learning models as shown in Figs. 6, 7 and 8 proved the model’s efficiency and accuracy (Fig. 9).

Conclusively, the optimized ensemble model is a very simple implementable model with a high bug prediction accuracy, efficient memory utilization and minimum execution time.

References

- Jin C, Jin SW (2015) Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization. *Appl Soft Comput* 35:717–725
- Zhang ZW, Jing XY, Wang TJ (2017) Label propagation based semi-supervised learning for software defect prediction. *Autom Softw Eng* 24(1):47–69
- Chen X, Zhang D, Zhao Y, Cui Z, Ni C (2019) Software defect number prediction: unsupervised vs. supervised methods. *Inf Softw Technol* 106:161–181
- Phan H, Andreotti F, Cooray N, Chén OY, De Vos M (2018) Joint classification and prediction CNN framework for automatic sleep stage classification. *IEEE Trans Biomed Eng* 66(5):1285–1296
- Chatterjee S, Maji B (2016) A new fuzzy rule-based algorithm for estimating software faults in early phase of development. *Soft Comput* 20(10):4023–4035
- Mustaqeem M, Saqib M (2021) Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection. *Clust Comput*. <https://doi.org/10.1007/s10586-021-03282-8>
- Sasidharan R, Sriram P (2014) Hyper-quadtrees-based k -means algorithm for software fault prediction. *Computational intelligence cyber security and computational models*. Springer, New Delhi, pp 107–118
- Turabieh H, Mafarja M, Li X (2019) Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. *Expert Syst Appl* 122:27–42
- Yang X, Lo D, Xia X, Zhang Y, and Sun J (2015) Deep learning for just-in-time defect prediction. In: *IEEE international conference on software quality, reliability and security*, p 17–26
- Ji H, Huang S, Wu Y, Hui Z, Zheng C (2019) A new weighted naive Bayes method based on information diffusion for software defect prediction. *Softw Qual J* 27(3):923–968
- Rathore SS, Kumar S (2015) Predicting number of faults in software system using genetic programming. *Procedia Comput Sci* 62:303–311
- Soleimani A and Asdaghi F (2014) An AIS based feature selection method for software fault prediction. In: *Iranian conference on intelligent systems (ICIS)*, IEEE, p 1–
- Tong H, Liu B, Wang S (2018) Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *Inf Softw Technol* 96:94–111
- Ricky MY, Purnomo F, and Yulianto B, (2016) Mobile application software defect prediction. In: *IEEE symposium on service-oriented system engineering (SOSE)*, p 307–313
- Nam J and Kim S (2015) Clami: defect prediction on unlabeled datasets. In: *IEEE/ACM international conference on automated software engineering (ASE)*, p 452–463
- Okutan A, Yıldız OT (2014) Software defect prediction using Bayesian networks. *Empir Softw Eng* 19(1):154–181
- Mendis C, Yan C, Pu Y, Amasinghe S, Carbin M (2019) Auto-vectorization with imitation learning. In: *Conference on neural information processing systems, (NEURIPS)*, p 1–12
- Hall T, Zhang M, Bowes D, Sun Y (2015) Some code smells have a significant but small effect on faults. *ACM Trans Softw Eng Methodol (TOSEM)* 23(4):1–39
- Yuzhou L (2021) A novel DL approach to PE malware detection: exploring GLOVE vectorization, MCC-RNN and feature fusion. p 1–19
- Yadav HB, Yadav DK (2015) A fuzzy logic-based approach for phase-wise software defects prediction using software metrics”. *Inf Softw Technol* 63:44–57
- Biçer MS and Diri B (2015) Predicting defect prone modules in web applications. In: *International conference on information and software technologies*, Springer, Cham, p 577–591
- Elish KO, Elish MO (2008) Predicting defect-prone software modules using support vector machines. *J Syst Softw* 81(5):649–660
- Maua G, Grbac GT (2017) Co-evolutionary multi-population genetic programming for classification in software defect prediction. *Appl Soft Comput* 55:331–351
- Parag CP (2010) Exhaustive and heuristic search approaches for learning software defect prediction model. *Eng Appl Artif Intell* 23(1):34–40
- Abaei G, Selamat A, Fujita H (2015) An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. *Knowl Based Syst* 74:28–39
- Li B, Shen B, Wang J, Chen Y, Zhang T, and Wang J (2014) A scenario-based approach to predicting software defects using compressed C4. 5 model. In: *IEEE 38th annual computer software and applications conference*, p 406–415
- Aman H, Amasaki S, Sasaki T, Kawahara M (2015) Lines of comments as a noteworthy metric for analyzing fault-proneness in methods. *IEICE Trans Inf Syst* 98(12):2218–2228
- Ulan M, Löwe W, Ericsson M, Wingkvist A (2021) Copula-based software metrics aggregation. *Softw Qual J*. <https://doi.org/10.1007/s11219-021-09568-9>
- Biçer MS, Diri B (2016) Defect prediction for cascading style sheets. *Appl Soft Comput* 49:1078–1084

30. Bowes D, Hall T, Harman M, Jia Y, Sarro F, and Wu F, (2016) Mutation-aware fault prediction. In: Proceedings of the 25th international symposium on software testing and analysis, p 330–341
31. Kim S, Whitehead EJ, Zhang Y (2008) Classifying software changes: clean or buggy. *IEEE Trans Softw Eng* 34(2):181–196
32. Zhao Y, Yang Y, Lu H, Liu J, Leung H, Wu Y, Xu B (2017) Understanding the value of considering client usage context in package cohesion for fault-proneness prediction. *Autom Softw Eng* 24(2):393–453
33. Goyal S (2021) Predicting the defects using stacked ensemble learner with filtered dataset. *Autom Softw Eng* 28:14. <https://doi.org/10.1007/s10515-021-00285-y>
34. Yao J, Shepperd M (2021) The impact of using biased performance metrics on software defect prediction research. *Inf Softw Technol* 139:1–14. <https://doi.org/10.1016/j.infsof.2021.106664>
35. Matloob F, Ghazal TM, Taleb N, Aftab S, Ahmad M, Adnan-Khan M, Abbas S, Rahim-Soomro T (2021) Software defect prediction using ensemble learning: a systematic literature review. *Digit Object Identif*. <https://doi.org/10.1109/ACCESS.2021.3095559>
36. Musinat B, Johnson F, Folorunso O, Ezinne I (2021) Genetic algorithm-based multi-objective optimization model for software bugs prediction. *Ann J Tech Univ Varna Bulg* 6(1):34–48
37. Wang Y, Zorzi S and Bittner K (2021) Machine-learned 3D Building Vectorization from Satellite Imagery. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), p 1–14. <https://arxiv.org/pdf/2104.06485.pdf>
38. Balogun AO, Oladele RO, Mojeed HA, Amin-Balogun B, Adeyemo VE, Aro TO (2021) Performance analysis of selected clustering techniques for software defects prediction. *Afr J Comput ICT* 12(2):30–42
39. Alattas K (2021) System error estimate using combination of classification and optimization technique. *J Comput Sci* 17(3):319–329. <https://doi.org/10.3844/jcssp.2021.319.329>
40. Haj-Ali A, Ahmad NK, Willkie T, Soppjia Y, Asanovic K, and Stolica I (2020) Neurovectorizer: end-to-end vectorization with deep learning. *ACM ISBN978-01-4503*, p 242–245
41. Matveiev O, Zubenko A, Yevtushenko D, and Cherednichenko O (2022) Towards classifying HTML embedded product data based on machine learning approach. pp 1–11

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.