IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 October 2023, accepted 25 October 2023, date of publication 1 November 2023, date of current version 29 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3329051

RESEARCH ARTICLE

Software Defects Identification: Results Using Machine Learning and Explainable Artificial Intelligence Techniques

MOMOTAZ BEGUM[1], MEHEDI HASAN SHUVO[1], IMRAN ASHRAF[2], ABDULLAH AL MAMUN[1], JIA UDDIN[3], AND MD ABDUS SAMAD[2], (Member, IEEE)

[1]Department of Computer Science and Engineering, Dhaka University of Engineering and Technology (DUET), Gazipur 1707, Bangladesh
[2]Department of Information and Communication Engineering, Yeungnam University, Gyeongsangbuk-do, Gyeongsan-si 38541, South Korea
[3]AI and Big Data Department, Endicott College, Woosong University, Daejeon 34606, South Korea

Corresponding authors: Md Abdus Samad (masamad@yu.ac.kr) and Jia Uddin (jia.uddin@wsu.ac.kr)

This work was supported by Woosong University Academic Research 2023.

ABSTRACT The rising deployment of software in automation and the cognitive skills of machines indicate a machine revolution in modern human civilization. Thus, diagnosing and predicting software faults is crucial to software reliability. In this paper, we first preprocessed four real datasets offered by National Aeronautics and Space Administration with twenty-one features using the Synthetic Minority Oversampling Technique and Label Encoding techniques. Subsequently, we experimented with thirteen software fault diagnosis Machine Learning (ML) models, *i.e.,* Random Forest Regression, Linear Regression, Naïve Bayes, Decision Tree Classifier, Logistic Regression, KNeighbors Classifier, AdaBoost, Gradient Boosting Classifier, Gradient Boosting Regression, XGBR Regressor, XGBoost Classifier, Extra Trees Classifier and Support Vectors Machine after that, we compared each ML Model to select the best diagnostic model. Among them, XGBR outperformed, considering the accuracy, mean square error, and R^2 score. We also used Explainable Artificial Intelligence (XAI), Local Interpretable Model (LIME), and SHapley Additive exPlanations (SHAP) to determine software fault features. We observed that Number of static invocations (*nosi*), Depth Inheritance Tree (*dit*), and Coupling Between Objects (*cbo*) features are the most affected software faults feature from datasets. For LIME, the average True positive of *nosi* is 40%, *dit* is 15%, and *cbo* is 20%; on the other hand, the SHAP average true positive value of *nosi* is 36%, *cbo* is 15%, and the norm true negative value of *dit* is 5%. Thus, LIME can afford the greatest impact on the model outcomes to identify features that are the most significant reasons for software defects.

INDEX TERMS Software defect prediction, features selection, software reliability, software fault diagnosis, explainable AI, SHAP, LIME.

I. INTRODUCTION

As people rely a growing amount on software in their daily lives, such as education, environment, and economic sectors, the complexity of software is increasing [1]. On the other hand, as the use of the internet in all domains has expanded, existing methods to accurately detect software defects have become more challenging. Relevant researchers proposed a software defect prediction system to ensure software quality and quickly identify as many defect modules as possible. In [2], they used the different types of neural network architectures to find the prediction interval for the total number of software defects found during the testing phase. They also looked at how the number of input and hidden nodes affected the reliability of the prediction using the principles of thumb.

Identifying the root causes of software defects are crucial intervention in software development and maintenance.

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Elish.

It can enable allocated resources effectively, prioritize testing efforts, and improve software quality. By determining essential causes, researchers can address underlying problems, thereby preventing the recurrence of defects and enhancing the long-term reliability of software. Therefore, the timely acceptance of modules with an elevated probability of defects during the initial phases of the development process has become more significant. ML-based strategies [3] have been the most well-known. Despite their visible success, they become black boxes, making it difficult for people to understand the reasoning behind their judgments fully. Due to the ''black box'' of ML-based strategies, the XAI approaches like LIME [4], and SHAP [5] have recently gained popular opinion. These strategies explain any decisions made by a model.

Recent [6] research has demonstrated that the LIME and SHAP methodologies can provide insight by ranking features according to their significance in black box decisions. Due to the lack of reality, researchers typically utilize multiple post hoc explanations to comprehend a single model decision. Such an atmosphere generates disagreement between methods of explainability and negatively impacts conclusion drawing. They proposed a method for merging LIME and SHAP explanations that places fewer sights on disagreements while emphasising areas of agreement. Consequently, they are a versatile tool to complement the usual ML-based methodology for software defect prediction. It determines whether or not software modules have faults based on codes or software characteristics. Therefore, ensuring the software quality is a core issue by identifying the features of software defects [7].

A Multilayer Perceptron (MLP) is a type of feedforward artificial neural network that consists of multiple layers of interconnected neurons arranged in a sequence. The authors [8], described an MLP with a one-stage look-ahead prediction as an experiment in solving and predicting software faults using Box-Cox Power Transformation. In [9], a combination of machine learning and evolutionary computing techniques was used to enhance the predictive performance of traditional ANN models.

On the other hand, software release time heavily depends on the appropriate diagnosis of a software fault, which has become researchers' primary concern. Begum and Dohi [10], [11] presented various ANN strategies for identifying optimal software release time. The limitation of employing a multi-stage look-ahead prediction method in which the optimal number of hidden neurons and transformation values were determined.

In the last four decades, most of the researchers have worked on software defect prediction. On the other hand, predicting software defects is an important subject for ensuring software reliability, maintenance, and cost minimization. They did not clear the root of the issue of software faults. So, in this concern we showed the prediction of software faults then we chose the best model that performed a good score overall after that we showed which features are important for software faults. The contributions of this study are the following.

- In our systems here, we have examined the features of software fault diagnosis based on the accuracy, mean square error, and R^2 score with thirteen machine-learning algorithms Random Forest Regression (RF), Linear Regression (LR1), Naïve Bayes (NB), Decision Tree Classifier (DTC), Logistic Regression (LR2), KNeighbors Classifier (KNC), AdaBoost, Gradient Boosting Classifier (GBC), Gradient Boosting Regression (GBR), XGBR Regressor (XGBR), XGBoost Classifier (XGBC), Extra Trees Classifier (ETC) and Support Vectors Machine (SVM) while choosing the best algorithm. After that, the selected algorithms used in LIME and SHAP techniques of XAI are illustrated. Because it provides transparent and understandable explanations for complex ML models, enabling better decision-making, trust, and accountability in AI systems.
- In addition, it makes a difference in the relative importance of the main k characteristics that occur more frequently than disagreements about their importance's direction. Based on this realization, we obtained the LIME explanations as the novel way for finding the top k features.

It is important to note that the effectiveness of any specific method in software defect identification depends on various factors, including the quality and quantity of the data used for training, and the choice of ML algorithms. Additionally, the integration of XAI techniques ensures that the results are not only accurate but also interpretable, which is crucial for gaining trust and insights from stakeholders in the software development process.

The rest of the paper is organized as follows: Section II discusses related works, the gap in the field, and the papers for the motivation behind our study. The proposed methodology is explained in Section III for software fault diagnosis by applying various statistical and machine-learning methods with suitable block diagrams. In Section IV, the theoretical formulation of all machine learning algorithms is explained in detail, and the numerical illustration of our proposed systems is briefly described in Section V, which includes the analysis of the results by different performance metrics.

II. LITERATURE REVIEW

In contemporary society, the software is a vital part of our everyday existence. Under the circumstances, various works regarding software fault prediction with different methods have been proposed, which involve using specific statistical techniques such as Multivariate Regression (MR), LR, and Univariate Regression (UR). These techniques are effective in identifying faults in researchers. However, they may not be helpful for novel research; as a result, scholars have devised the implementation of ML and its methodologies.

Additionally, academics look into many facets of ML to address software fault diagnosis. They came up with

a high-level design class cohesion measure that is based on reasonable assumptions, works with expected math, and can be used to automatically check the quality of designs early on using UML models. They proved that similarity-based class cohesion is based on well-thought-out theoretical principles, works with reasonable assumptions, and can rapidly identify faults [12]. On the other hand, the aim of software fault diagnosis is to identify and rectify errors in computer programs as well as ensure optimal software release time. Regarding this, they compared their method to non-homogeneous Poisson process (NHPP) based software reliability models (SRMs) and demonstrated that their method provides more precise and adaptable decision-making. Using ANNs and software defect count data, the authors proposed an original method to estimate the optimal software release time and find software fault prediction [13].

In [14], we discussed the models and metrics of software, using classification, regression, and other significant techniques for predicting software fault issues. However, DT calculations may become substantially more complex and unstable due to minor modifications to the data; therefore, the training process takes longer time. Recently, in [15], they introduced long short-term memory (LSTM) networks as a novel approach for predicting software errors using Min-Max Scalar and Box-Cox Transformation to normalize software fault count data and showed the dispersion of data. However, for simplicity, they have worked on a small number of epochs and neurons of the LSTM network for simplicity.

The ML technique has been used for discovering faults by collecting information from software archives and comprising messages and source code proposed in [16]. Many assessment metrics were used for finding the accuracy of different techniques, such as the probability of detecting False Positive (FP), True Negative (TN), and True Positive (TP) costs, precision, G measure, and F measure, and *so on*. They did not conduct a comprehensive study of issues of software test cases within publicly available source code. The authors [17] used the optimized artificial immune networks (Opt-aiNet) approach to choose the features for forecasting faults. Finally, the results are evaluated with ML classifiers using the AUC and accuracy metrics. The individuals in question did not demonstrate care in identifying the underlying causes of software issues.

Additionally, a software failure prediction model was recently provided with a variety of data transformation strategies from Poisson data to Gaussian data. Then, in order to predict long-term bugs in the software, LR1 was used and compared with Nave Gauss and exponential smoothing time series forecasting models in terms of average relative error. Data transformation methods were employed to convert Poisson data into Gaussian data. However, the ideal value of λ for the Box-Cox and Yeo-Johnson power transformation was not determined presented by Begum, M. et. al. [18].

The purpose of research is to improve the accuracy of literature studies. Our main objective is to identify software defects' root causes, and is the process of pinpointing the underlying reasons behind defects in software systems. Researchers are very concerned about finding the problem definition of this domain. Most of them are focused on investigating methodologies, tools, or approaches that allow efficient and accurate identification of root causes. This research aims to reduce software development costs, improve software quality, and enhance the overall software development process by addressing the challenges of identifying and mitigating defects effectively.

Firstly, we preprocessed our datasets using Synthetic Minority Oversampling Technique (SMOTTech) and Label Encoding (LE) techniques after examining the processed data through several ML algorithms. Then, we evaluated all algorithms using different simulation criteria. XGBR Regression ML Model goes to a good score. Next, the best model was used in XAI for specific main finding features for software faults. Here, we work with SHAP and LIME in XAI techniques. Finally, compared with both techniques, LIME provides a more accurate diagnosis of software faults.

III. PROPOSED METHODOLOGY

Software developers focus on the accuracy of software functions and reliability, as individuals cannot survive without it in a single day. Software diagnosis has become more challenging over the last several decades as the complexity and volume of software have increased. In this paper, we have found appropriate features for causing software faults via the best architecture of ML with XAI.

Figure 1 explains the proposed structure for identifying top k features for software faults diagnosis. At first, we collected datasets from different resources. After that, we used SMOTTech and LE techniques for data preprocessing. There are two distinct groups in our data. Thirteen ML algorithms are used for analysis, and we ''trained'' our models using 80% training data and then ''tested'' using 20% testing data from datasets. Consequently, we have considered many metrics while assessing our models, such as Mean Absolute Percentage Error (MAPE) and Explained Variance. After exhaustively comparing all available options, we exported the most optimal model and obtained XGBR as the best model. Then, we apply LIME and SHAP techniques to analyze software faults. Afterward, we compare which technique is most acceptable. Lastly, we decide on finding the top k features for software faults based on the accuracy of techniques, where LIME provides the best accuracy.

A. DATASETS

In this investigation, we have used four open sources of real project datasets PC1, Baseline, CM1, and JM1 from the PROMISE software engineering database [26], where twenty-two features have been considered for diagnosing software faults. Here, twenty-one independent features; the

TABLE 1. Contributions and limitations of different studies in the literature.

Ref.	Contributions	Limitations
[19]	Using a combination of expert knowledge and machine learning, the authors performed anomaly detection and root cause analysis in sensor data streams. They ultimately reduced defects and anomalies downtime and provided greater insight into frequently occurring issues.	They did not assess multiple cases regarding the reduction of human involvement and how to make the rule-mining component more dynamic.
[20]	ML-based classifications such as ANNs classifier, NB classifier, and DTC were utilized for fault production they used five classes for experimentation, and NB and SVM performed the best for specific software faults compared to other algorithms.	The paper in question looked at how ML techniques can be used to predict software bugs. But they did not work on fixing the bugs that caused them.
[21]	They applied LSTM for multiple days of software faults using a recursive approach with different data transformation techniques. After preprocessing the datasets, they compared their model with the performance of conversational software reliability growth models (SRGMs).	Computation time for the recursive approach with LSTM took a long time.
[22]	Analyzing the DT model in the context of cyber-security applications, the researchers have used the XAI idea to improve trust management.	Not having category variables with a different number of levels and having issues with their datasets being too overfitted were problems that they had. This issue makes it work better on the training dataset but not so well on the test dataset.
[23]	The authors have presented a framework for developing software prediction datasets with adequate characteristics and statistical data validation methods.	They have only considered a binary prediction for Defective instead of Non-Defective code, without taking into account the prediction of the severity of the defect.
[24]	Six open-source projects were used to train on the just-in-time software fault prediction model. This helped them make more accurate predictions for interpretability applications.	They have struggled to explain intuitively complex, non-linear relationships between software metrics and defects.
[25]	They refined the artificial neural network (RANN) method used to predict long-term software faults based on the number of software faults and suggested a simulation-based method (PI simulation) to make prediction intervals (PIs). Finally, they compared with the conventional delta method from the point of view of several assessment criteria, such as the mean prediction interval width and PI coverage rate.	The software metrics (such as McCabe, Halstead, and OO metrics) have not been verified using various prediction interval techniques.

TABLE 2. Details of datasets.

No	Dataset	Instance	Class Distribution	
			Faults	No Faults
1	PC1	4714	2360	2354
2	Baseline	6052	3026	3026
3	CM1	6992	3455	3537
4	JM1	6029	3018	3011

remaining one depends on which reveals whether there is a software fault or not. Table 2 presents the details of all datasets and their properties.

In PC1, the total number of instances is 4714, and the class distribution is composed of such a dataset that the dependent feature has 2360 faults and 2354 no faults. In addition, Baseline data sets consist of 6052 faults, and no faults are the same, 3026. As well as CM1 and JM1 data sets are given as 6992 and 6029 instances, respectively. It should be noted that all data sets are McCabe, Halstead, Miscellaneous, True or False type.

B. PREPROCESSING

In this paper, we have preprocessed our datasets using LE and SMOTTech techniques for data normalization and balancing. Chawla et. al. emerged with the SMOTTech regular approach [27] in 2002. In this method, the minority class is made up of fake samples of the minority class that have equal distribution around the originally identified positive cases. SMOTTech makes fake samples from underserved groups by running its business in the feature space. First, SMOTTech finds the data points that are closest to the ones in the minority

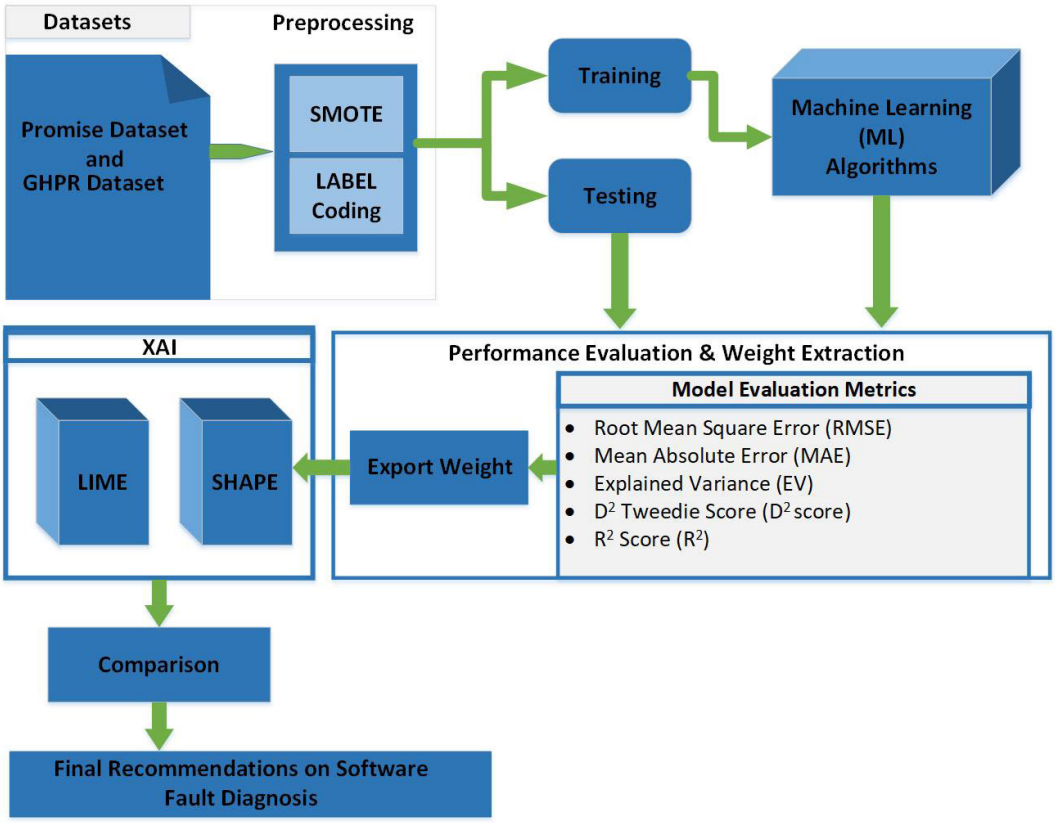


FIGURE 1. Proposed framework for finding top k features for software faults diagnosis.

group. Then, it makes a new point that is totally random in relation to the other points. The marginalized group is made up of numbers that are represented by these new dots. Lastly, it will keep making new data until the problem with the mismatch of data is fixed. f_i stands for the feature vector of the under-investigated minority class sample, and f_{near} is one of f_i's is K closest neighbors. Equation (1) is a way to show what the newly made fake sample f_{new} looks like.

$$f_{new} = f_i + (f_i - f_{near}) \times R \tag{1}$$

Here, $i = 1, 2, 3, \ldots, n$, n is the total number of data points, and R is a random number that ranges from 0 to 1. Additionally, we used the LE technique to convert categorical data into numerical values. It assigns a unique integer to each category or label in a categorical variable. The basic idea is to represent categories with integers so that ML algorithms can work with the data [28].

C. ML MODELS

1) LINEAR REGRESSION (LR1) AND LOGISTIC REGRESSION (LR2)

LR1 describes a specific modeling scenario when there is a singular independent variable. LR1 [29] can distinguish between how dependent factors influence one other and how independent variables influence each other. On the other hand, LR2, also known as the logistic or logit model, is a statistical methodology investigating the statistically significant relationship between multiple independent variables and a categorical dependent variable. The methodology seeks to determine the probability of an event's occurrence by fitting the available data to a logistic curve.

2) SUPPORT VECTOR MACHINE REGRESSION (SVM)

An SVM is a powerful ML algorithm used for classification and regression tasks. The primary objective of the SVM algorithm is to ascertain mathematical functions that exhibit a high degree of regularity with respect to the predictor variables. Concurrently, it reduces the difference between the predicted and actual values by a fixed threshold for all training data points. The time complexity of SVM increases at a rate greater than quadratic as the number of samples increases [30].

3) NAIVE BAYERS (NB)

The NB classifier is a supervised machine learning algorithm applied for classification tasks such as text classification. In addition, it has several advantages, including a straightforward methodology and high reliability.

4) RANDOM FOREST REGRESSION (RF) AND DECISION TREE CLASSIFIER (DTC)

The RF algorithm is an ensemble technique that practices randomized decision trees. The outcome is a collective methodology derived from several DTC scores produced using techniques such as bagging or bootstrapping, subsampling, or RF. Regarding regression, the unexpected outcome of the forest is determined by the average scores obtained from the randomized decision trees.

In contrast, the DTC methodology involves dividing the predictor variable field into distinct subsets based on the similarity of their corresponding target variables. It creates a tree-like structure where each internal node represents a feature, and each leaf node corresponds to a class label. It is popular for its simplicity and ability to handle both numerical and categorical data. On the other hand, a random variable determines the optimal partitioning strategy at each node in a randomized decision tree [31].

5) ADAPTIVE BOOSTING (ADABOOST)

AdaBoost, also known as Adaptive Boosting, is a very successful technique utilized for enhancing the performance of learners who are struggling [32]. This technique can be utilized with various learning algorithms, such as DT, neural networks, and SVM. The strategy is based on training weak learners repeatedly and assigning more significance to incorrectly classified samples.

The process mentioned above is iterated until the necessary degree of precision has been attained or until the maximum threshold of struggling learners has been reached. The ultimate result is obtained by calculating the weighted total of the predictions made by the learners that have been weakened. This methodology has notable advantages when the dataset consists of a considerable amount of heterogeneous examples through a limited amount of factors, and the division of the dataset is either ineffective or non-uniform.

6) GRADIENT BOOSTING CLASSIFIER (GBC) AND GRADIENT BOOSTING REGRESSION (GBR)

GBC, or Gradient Boosting, is an additional ensemble machine learning algorithm that is effective for regression and characterization problems. It implements the boosting procedure, combining several weak learners to form one strong learner. Calculation of this algorithm can rapidly overfit a preparing dataset. It may gain an advantage from regularization techniques that execute various parts of the calculation. Finally, enhance the calculation's presentation by reducing overfitting [33].

On the other hand, a GBR is an ensemble model consisting of several tree models placed sequentially. Each subsequent model in the sequence learns from the errors made by the previous model. It uses ''boosting'' to generate predictions and combines multiple weak prediction models. At last, it creates decision trees, for a more resilient and accurate model. The output of $f_n(x)$ for input x can be written of n trees as the sum:

$$f_n(x) = \sum_{i}^{n} \gamma_i h_i(x) \tag{2}$$

where h_i represents an average learner that exhibits an individual performance, and γ_i is a scaling factor that accounts for the effect of a tree on the overall model. GBR employs the gradient descent loss function to minimize errors through the iterative process of updating the original estimation with the new measurement [34].

7) EXTREME GRADIENT BOOSTING (XGBOOST)

XGBoost implements gradient boosting machines (GBM), a popular supervised learning technique. This technique can be employed in both regression and classification scenarios. Data scientists favor XGBoost due to its remarkable capacity for high-speed execution in out-of-core computation [35]. It is extremely fast and may be executed on distribution platforms. In addition, the library of XGBoost is faster than other gradient-boosting libraries, which is a significant advantage. The machine learning system, XGBoost, is robust and flexible, making it suitable for various contexts. XGBoost is widely utilized in both industrial and research settings.

8) EXTRA TREE CLASSIFIER (ETC)

The ETC is an ensemble learning technique based on decision trees. In the ETC, the source of irregularity does not stem from data scaling but rather from the randomization component in selecting all predictors [36]. It operates by reducing variance and increasing perception simultaneously. One significant advantage of ETC compared to conventional DT is to enhance computational efficiency and reduce variance.

9) KNEIGHBOARS CLASSIFIER (KNN)

The supervised algorithm KNN is used for classification purposes. It has been utilized extensively for disease prediction, data classification, *etc*. The KNN algorithm can generally classify datasets using a training model similar to the testing query by considering the K training data points (neighbors) closest to the query [37].

IV. EXPERIMENTAL RESULT AND ANALYSIS

We used various evaluation techniques during the experimental result analysis, including MAE, RMSE, R^2 score, EV, maximum error, and accuracy. This section has been separated into two subsections. In the first section, the criteria employed for performance measurement were analyzed. Subsequently, the ML algorithms were evaluated based on these criteria, leading to the identification of the most optimal method.

A. PERFORMANCE METRICS

1) MEAN ABSOLUTE ERROR (MAE)

MAE measures errors between matched observations representing the same phenomena. This value is calculated by averaging the absolute mistakes. The fact that the MAE units are the same as the predicted goal makes determining if the error level justifies concern easier. By studying the MAE, the sum of these mistakes, we can better analyze the model's performance throughout the entire dataset. Error is one way to compare forecasts with actual results. The MAE is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{3}$$

where, $i = 1, 2, 3, \ldots, n$, y_i actual value, $\hat{y}_i$ predicted value, and n is the total number of data points.

2) ROOT MEAN SQUARE ERROR (RMSE)

RMSE, similar to mean square error (MSE), desires square rooting the MSE. RMSE behaves similarly to the standard deviation of the residuals, which indicates how evenly distributed the residuals are [38]. It is represented as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2} \tag{4}$$

3) R^2 SCORE (R^2)

In a regression model, the R^2 statistic represents the variance difference that can be attributed to the independent variables. The number falls between zero and one, where one means the model thoroughly explains the variance, and zero means it does not explain any variance.

$$R^2 = \frac{SSR}{SST} = \frac{\sum\left(\hat{y}_i - \bar{y}\right)^2}{\sum\left(y_i - \bar{y}\right)^2} \tag{5}$$

where $\bar{y}$ is the mean of actual value. The SSR measures how far the actual values deviate from the projected values. The SST is calculated as the summation of the average squared deviations between the predicted and actual values. The coefficient of R^2 is widely employed as a metric to assess the degree of adequacy in linear regression models, although its applicability extends beyond such models. It measures the goodness of fit of a model to the data. A higher R^2 value indicates a better match between the model and the data, whereas a lower value shows a poorer score between the model and the data.

4) EXPLAINED VARIANCE (EV)

The concept of EV is used in statistical analysis to quantify the proportion of the overall variability in the dependent variable of a regression model. It can be accounted for by the variations in the independent variables. Typically, it is represented as a numerical value ranging from zero to one. A score of one indicates that the model accounts for all variations in the dependent variable, while a zero value suggests that the model does not account for any variations. The R^2 statistic is generally used to calculate the explained variance, as described in [39].

$$\text{EV}(y_i, \hat{y}_i) = 1 - \frac{\text{Var}(y_i - \hat{y}_i)}{\text{Var}(y_i)} \tag{6}$$

where, $\text{Var}(y_i - \hat{y}_i)$ and $\text{Var}(y_i)$ are the standard deviations of prediction errors and actual values, respectively. The EV score close to 1.0 is a positive indicator of a well-fitting and reliable model.

5) D^2 TWEEDIE SCORE (D^2 SCORE)

The D^2 score consists of a range of probability distributions, such as the continuous normal, gamma, inverse Gaussian distributions and the discrete scaled Poisson distribution. Additionally, this family includes the compound Poisson-gamma distributions, which demonstrate a positive mass at zero but are otherwise indefinite. Tweedie distributions are a subclass of exponential dispersion models commonly employed as probability distributions within the framework of generalized linear models.

B. PERFORMANCE ANALYSIS

We performed an investigation using Python code for every dataset separately and thirteen ML algorithms. Listed below are our findings for each of the datasets.

KNC and NB have the most significant mean and root mean square Errors in Table 3. However, RFR and XGBC performed moderately well. This table showed that LR1, DTC, LR2, AdaBoost, and GBC achieved the most significant mean square error. However, its RMSE and R2 values are marginally higher. Concerning the concepts of EV, maximum error, and D^2 score, the obtained score can be classified as moderate. Nevertheless, the XGBR model demonstrates superior performance across all evaluated criteria. Based on a comprehensive analysis of many parameters, our research findings suggest that the XGBR model exhibits the highest level of performance among all the models shown in the table.

Our insights are tabulated in Table 4, XGBR, despite having a lower RMSE score with the top max error. Among the other ML models, ADC and LR1 performance was notable. However, XGBR's performance was the most superior.

The CM1 dataset is shown in Table 5. Though the XGBoost Classifier was the superior-performing model in the previous datasets, its performance decayed here. Except for GBR, the remaining ML algorithms performed much better. Furthermore, the DTC approach exhibits the lowest values for mean squared and absolute percentage errors. Based on our research, it has been determined that the XGBR model has more significant advantages in this particular subject area.

Table 6 shows what we discovered. Even though RFR did the best in the software fault predictions, it has the most significant mean squared error in this stream, even though it has an average RMSE score. AdaBoost Classifier and LR1

TABLE 3. Performance evaluation for PC1.

Sl No	Model	Accuracy	MSE	RMSE	R^2	EV	Max Error	D^2 Score
0	RFR	0.8674	0.3606	0.4745	0.0991	0.0993	1	0.0991
1	LR1	0.0863	0.4839	0.5026	-0.0108	-0.0107	2.2497	-0.0108
2	NB	0.5523	0.4687	0.6846	-0.8751	-0.2323	1	-0.8751
3	DTC	0.9840	0.3637	0.6031	-0.4551	-0.4550	1	-0.4551
4	LR2	0.7114	0.3372	0.5807	-0.3490	-0.3378	1	-0.3490
5	KNC	0.6117	0.6680	0.8173	-1.6726	-1.6726	1	-1.6726
6	AdaBoost	0.7960	0.2460	0.4960	0.0157	0.0460	1	0.0157
7	GBC	0.8164	0.2513	0.5013	-0.0054	0.0036	1	-0.0054
8	GBR	0.4075	0.3589	0.4193	0.2966	0.2966	0.8639	0.2966
9	**XGBR**	**0.4886**	**0.3284**	**0.4239**	**0.2811**	**0.2811**	**0.9489**	**0.2811**
10	XGBC	0.9596	0.3191	0.5649	-0.2769	-0.2753	1	-0.2769
11	ETC	0.9840	0.3732	0.6109	-0.4933	-0.4921	1	-0.4933
12	SVM	-0.3397	0.5015	0.5840	-0.3647	-0.1195	0.9044	-0.3647

TABLE 4. Performance evaluation for baseline.

Sl No	Model	Accuracy	MSE	RMSE	R^2	EV	Max Error	D^2 Score
0	RFR	0.7634	0.3807	0.4865	0.0526	0.0538	1	0.0526
1	LR1	0.0634	0.4729	0.4841	0.0621	0.0627	0.8842	0.0621
2	NB	0.5435	0.4931	0.7022	-0.9736	-0.3613	1	-0.9736
3	DTC	0.9180	0.3945	0.6281	-0.5788	-0.5759	1	-0.5788
4	LR2	0.6834	0.3469	0.5890	-0.3883	-0.3883	1	-0.3883
5	KNC	0.6151	0.7278	0.8531	-1.9127	-1.8977	1	-1.9127
6	AdaBoost	0.7798	0.2653	0.5150	-0.0616	-0.0200	1	-0.0616
7	GBC	0.8216	0.2959	0.5439	-0.1841	-0.1804	1	-0.1841
8	GBR	0.3581	0.3835	0.4294	0.2620	0.2636	0.8435	0.2620
9	**XGBR**	**0.5088**	**0.3633**	**0.4389**	**0.2289**	**0.2305**	**0.9349**	**0.2289**
10	XGBC	0.9163	0.3707	0.6088	-0.4836	-0.4798	1	-0.4836
11	ETC	0.9180	0.3843	0.6199	-0.5380	-0.5324	1	-0.5380
12	SVM	-0.4768	0.5120	0.6212	-0.5442	-0.0738	0.9003	-0.5442

results stood out among the other ML models. On the other hand, XGBR did the best job.

C. EXPLAINABLE ARTIFICIAL INTELLIGENCE (XAI)

XAI aims to improve the performance of individuals applying AI-driven systems by enhancing their comprehension of the underlying reasoning processes employed by these systems. It can potentially serve as a practical appearance of the social superiority to explanation. Although the absence of an obligatory right, the integration of XAI can potentially enhance the user experience of a given service. It provides transparency and comprehension of the decision-making processes of the underlying AI system. XAI endeavors to comprehensively explain past, ongoing, and future actions and the underlying information upon which these actions are predicated. Additionally, it can help understand a model's behavior and create faith in the model itself. Several well-known strategies in the field of XAI include SHAP, activation maximization, saliency map visualization, layer-wise relevance back-propagation (LRP), and LIME. This article provides a comprehensive analysis of the LIME and SHAP methodologies.

1) LOCAL INTERPRETABLE MODEL AGNOSTIC EXPLANATIONS (LIME)

LIME is model-independent which is applicable for explaining the expected results of ML models of their architecture or underlying assumptions [40]. Due to the method's model

TABLE 5. Performance evaluation for CM1.

Sl No	Model	Accuracy	MSE	RMSE	R^2	EV	Max Error	D^2 Score
0	RFR	0.8792	0.3058	0.4175	0.3021	0.3035	1	0.3021
1	LR1	0.0612	0.4745	0.4840	0.0621	0.0632	1.3928	0.0621
2	NB	0.5240	0.4631	0.6805	-0.8536	-0.3546	1	-0.8536
3	DTC	0.9822	0.2859	0.5347	-0.1442	-0.1399	1	-0.1442
4	LR2	0.7119	0.2830	0.5320	-0.1327	-0.1154	1	-0.1327
5	KNC	0.6061	0.6497	0.8060	-1.6002	-1.5961	1	-1.6002
6	AdaBoost	0.7822	0.2087	0.4568	0.1647	0.1791	1	0.1647
7	GBC	0.8022	0.2065	0.4545	0.1732	0.1786	1	0.1732
8	GBR	0.3688	0.3467	0.3980	0.3658	0.3667	0.8357	0.3658
9	**XGBR**	**0.4407**	**0.3082**	**0.3923**	**0.3838**	**0.3845**	**0.9374**	**0.3838**
10	XGBC	0.9404	0.2466	0.4965	0.0130	0.0139	1	0.0130
11	ETC	0.9822	0.2909	0.5393	-0.1642	-0.1627	1	-0.1642
12	SVM	-0.5640	0.5086	0.6396	-0.6376	-0.0047	0.9072	-0.6376

TABLE 6. Performance evaluation for JM1.

Sl No	Model	Accuracy	MSE	RMSE	R^2	EV	Max Error	D^2 Score
0	RFR	0.8639	0.3263	0.4282	0.2662	0.2664	1	0.2662
1	LR1	0.0637	0.4775	0.4856	0.0560	0.0562	1.4092	0.0560
2	NB	0.5335	0.4568	0.6759	-0.8282	-0.1431	1	-0.8282
3	DTC	0.9813	0.3242	0.5693	-0.2973	-0.2927	1	-0.2973
4	LR2	0.6954	0.3026	0.5501	-0.2110	-0.1516	1	-0.2110
5	KNC	0.6189	0.6923	0.8320	-1.7705	-1.7705	1	-1.7705
6	AdaBoost	0.7831	0.2031	0.4507	0.1870	0.2300	1	0.1870
7	GBC	0.8048	0.1965	0.4433	0.2136	0.2282	1	0.2136
8	GBR	0.3766	0.3450	0.3942	0.3779	0.3779	0.8474	0.3779
9	**XGBR**	**0.4472**	**0.3077**	**0.3871**	**0.4001**	**0.4001**	**0.9333**	**0.4001**
10	XGBC	0.9427	0.2620	0.5118	-0.0485	-0.0484	1	-0.0485
11	ETC	0.9813	0.3208	0.5664	-0.2840	-0.2830	1	-0.2840
12	SVM	-0.5139	0.5099	0.6284	-0.5805	-0.0564	0.9001	-0.5805

independence, it can clarify any ML strategy's predictions. It accomplishes this by providing a reduced and interpretable model only valid in the area surrounding the instance under consideration. This functionality enables users to comprehensively comprehend the rationale behind the model's predictions for a given instance. Therefore, the model's overall behavior is intricate as well as hard to comprehend. The approach operates by altering the provided input sample and producing a new dataset that is contextually equivalent to the original instance.

Then, it applies a simple and readily interpretable model, such as a linear model or a decision tree, to the new dataset. The coefficients of the LIME may be used to ascertain the significance of each attribute in predicting the original instance. Furthermore, it gives human-readable explanations of a model's behaviour. Additionally, it demonstrates efficacy in evaluating and interpreting predictions generated by ML models and facilitating the development of more transparent and dependable models. While working with LIME, we utilized the model that provides the best fit.

2) PSEDOCODE FOR LIME

In Algorithm [1] we showed model construction for the proposed LIME in XAI techniques. At first, we selected our best model M compared with thirteen ML techniques. After training the selected M Model, created a LineTabular_Explainer object called *explainer*. When created an object passes 3 parameters, firstly X is an independent feature then passes the feature name, and lastly passes ML type regression or classification that assigns to the variable *predictFnName*. Then call the *exp* (explain_instance) method on the explainer object. This object has information about the data features' importance. When calling this method then pass another 3 parameters, X, predict function *predictFnName*, and finally how many features want to show declare it starts from 1 to N, where N is the length of independent features size. The 'show_in_notebook()' method on the Explanation Object should be called. This will make a graph that shows which characteristics helped with the prediction. Other retrieval techniques are available for obtaining feature importances in various formats 'as_list()', 'as_map()', 'as_html()', and 'as_pyplot_figure()'.

Algorithm 1 Model Construction for Proposed LIME in XAI

Require: M best ML-MODEL with fit by XTrain, YTrain data and X independent features

1: $M \leftarrow$ best ML-Model compare other algorithms
2: **if** M is regression **then**
3: $mode$ = regression, $predictFnName = M$.predict
4: **else**
5: $mode$ = classification, $predictFnName = M$.predict_fn
6: **end if**
7: $explainer \leftarrow$ lime.lime_tabular.LineTabular_Explainer (X, features_names = X.columns, mode = $mode$)
8: $exp \leftarrow$ explainer.explain_instance(X, $predictFnName$, num_features = $1, 2, 3, \ldots, n$)
9: exp.show_in_notebook()
10: **Return:** visualizes feature importance by highlighting the most influential input features for an ML model's predictions.

3) RESULT ANALYSIS OF LIME

Table 7 and 8, represent the discovery of the features of software faults using LIME, where, Figure a.1 and a.2 (in Table 7) show the structure of dataset PC1. The visualization of the true value is 0.7035, whereas our model predicted 0.7468 presented in Figure a.1. LIME has successfully outlined the most essential characteristics in producing these expected results. Among 22 features in the dataset, the *dit* feature showed the highest impact on the model's estimation ability. However, our approach adequately anticipated its outcome, as seen in Figure a.2. The actual value was 0.2486, while the value predicted by the model was 0.1373. It can be found that *cbo*, *mathOperationsQty*, and the *LOC* features have the most favorable values. We can realize that by using LIME, the above three attributes have become essential reasons for diagnosing software faults.

Figure b.1 and b.2 (in Table 7), we depict the major features for reasoning the software faults of the dataset Baseline. In Figure b.1, our model predicted 0.6449, but the actual value is 0.7397. This modest misconception of the prediction is due to the *variablesQty*, the *number of mathOperationsQty*, and *stringLiteralsQty* features. In addition, Figure b.2, features *nosi*, *cbo*, *LOC*, *variableQty*, and *assignmentQty* of this dataset's software defects have been positively impacted, which results in its model predicting 0.7082 when the actual value is 0.9146.

Additionally, in Figure c.1 and c.2 (in Table 8) shown by CM1, and here in Figure c.1, the predicted value is 0.1827, and the true value is 0.1342 on the other side, Figure c.2, the predicted value is 0.6828, and the true value is 0.7065. Finally, in JM1, as in Figure d.1 and d.2 (in Table 8), Figure d.1, depicts that predicting the outcome local value is 0.2112, whereas our model predicted 0.1371. Also, the most negative features are *nosi*, *dit*, and *stringLiteralsQty*.

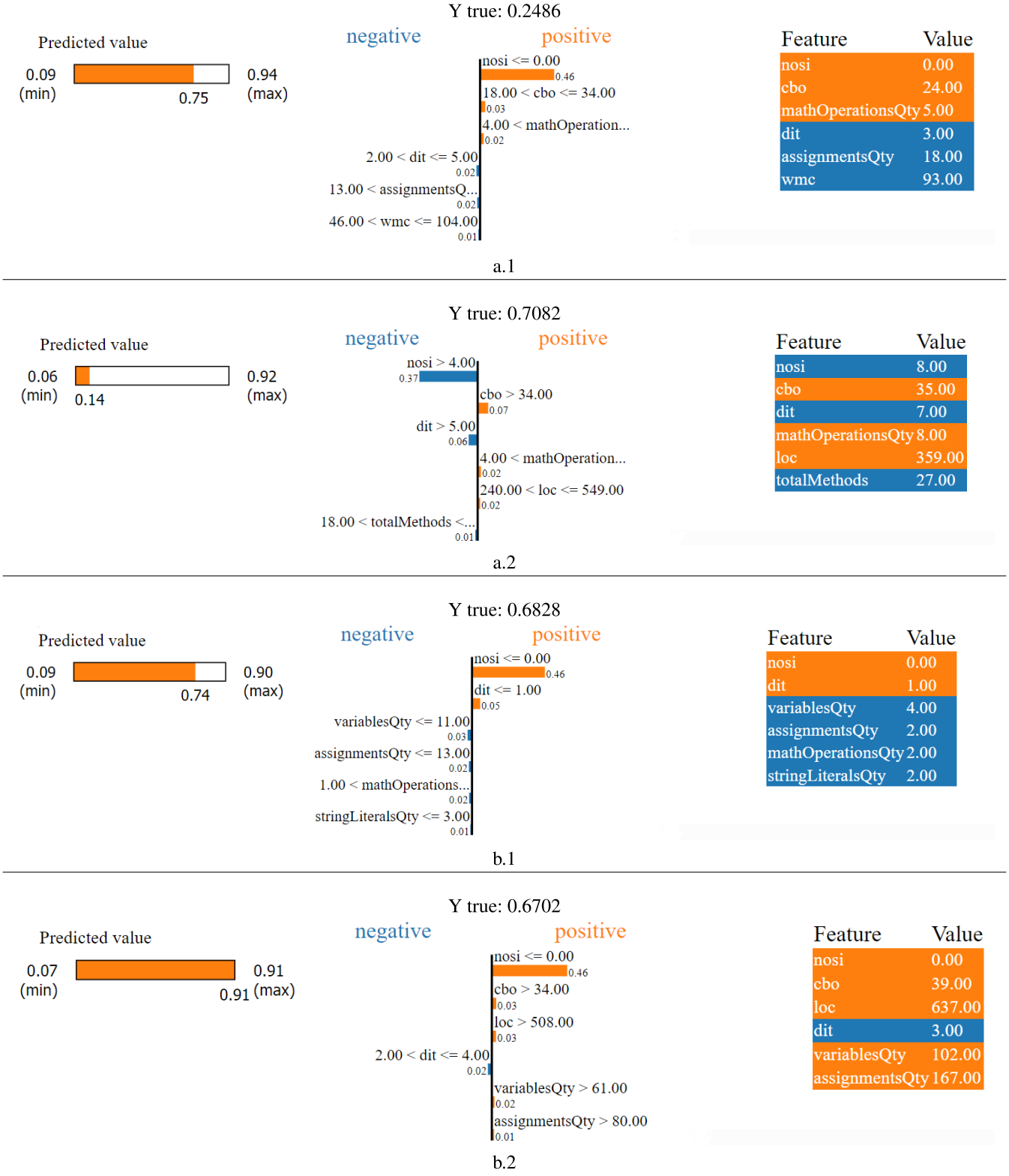
4) SHAPLEY ADDITIVE EXPLANATIONS (SHAP)

The SHAP approach is applied to explain individual forecasts and make a prediction approach using the model for a specific instance. After that, the overall prediction is achieved by measuring the individual features. It computes shapley values derived from coalitional game theory, providing a method for equally distributing the predicted outcome among the various attributes. One advantage of SHAP is the representation of the shapley value explanation as an additional feature attribution technique, which can be conceptualized as a linear model.

5) PSEDOCODE FOR SHAP

In Algorithm [2] we showed model construction for the proposed SHAP in XAI techniques. At first, we selected our best M model compared with thirteen ML techniques. After training the Selected M Model, create a TreeExplainer object called *explainer* and pass in parameters independent feature of M. Then create the shap_values object as *shapValues* pass value of X independent features for data examples using the explainer object. It have been built to produce charts using both matplotlib and JavaScript. Using the JavaScript backend, we'll produce a chart. We must initialize shap by using the initjs() function in order to do so. The force plot illustrates the contributions of the shap values used in an additive force layout to get the overall prediction. When calling this method pass 3 parameters firstly pass expected_value which takes the base value, to which the shape values are appended. Then pass *shapeValues* and X for an individual sample of data. Other retrieval techniques are available for obtaining feature importance in various formats 'Bar Plot', 'Waterfall Plot', 'Decision Plot', 'Dependence Plot', 'Embedding Plot', 'Summary Plot', and 'Partial Dependence Plot'.

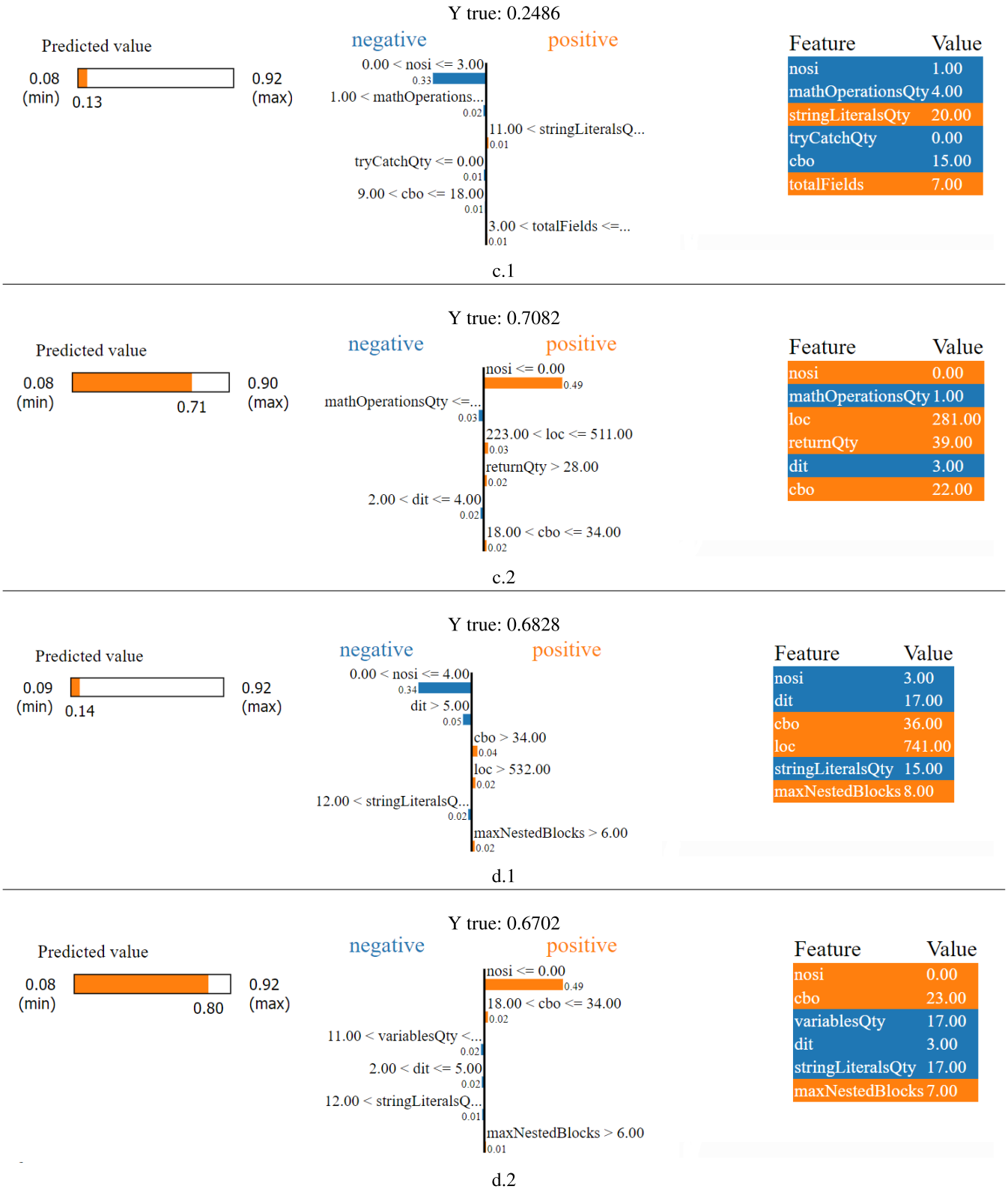
TABLE 7. LIME Agnostic Explanations Prediction. Here, PC1 by (a.1) and (a.2), Baseline by (b.1) and (b.2).



6) RESULT ANALYSIS OF SHAP

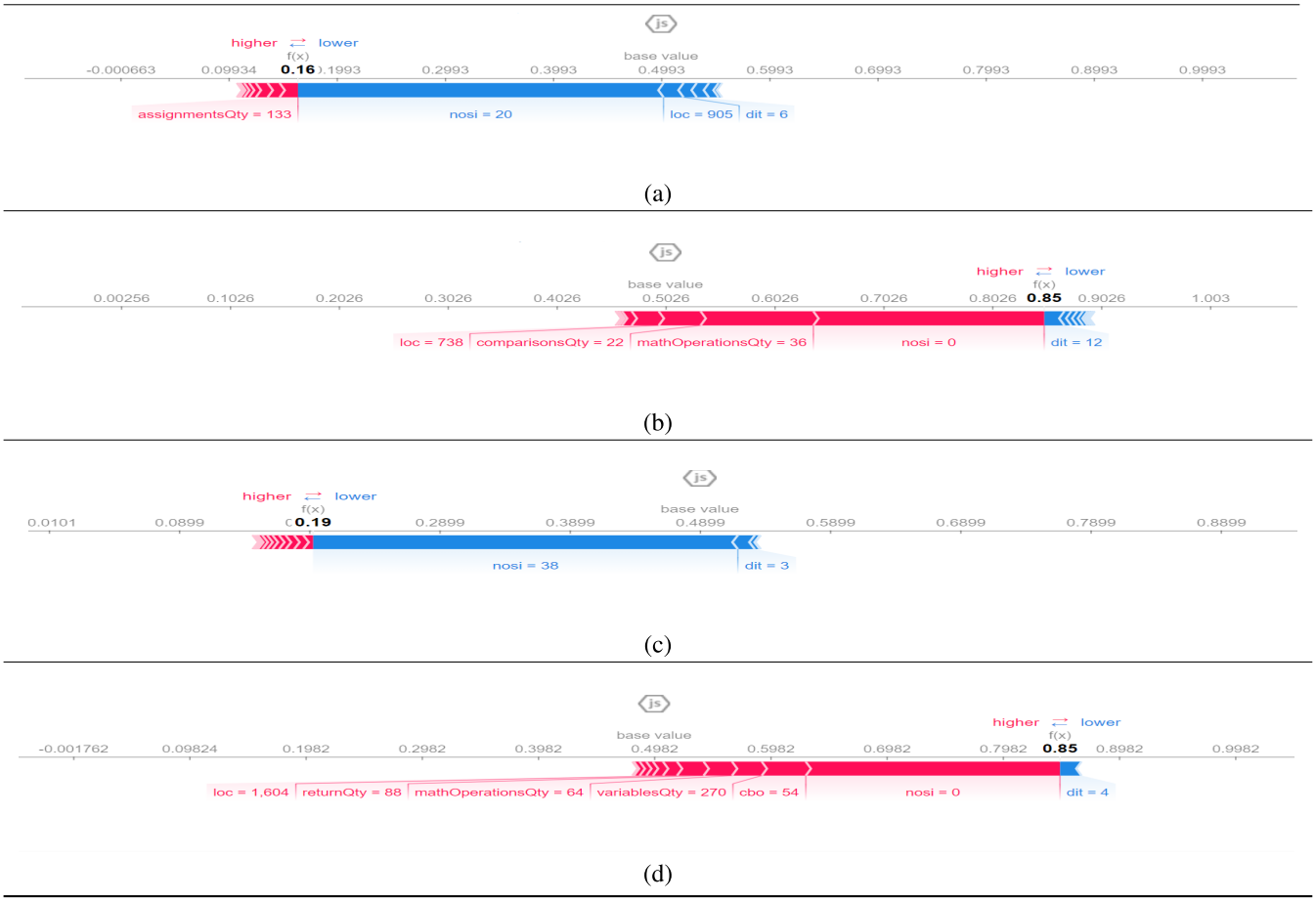
In 3. (a) (in Table 9) base value is 0.4493, the average of all predictions made by the model on the training dataset is also known as null model prediction (model without any features). Here, the output value is 0.16, called a prediction value. Features in red color influence positively,

TABLE 8. LIME Agnostic Explanations Prediction. Here, CM1 by (c.1) and (c.2), and finally, JM1 by (d.1) and (d.2).



dragging the prediction value closer to 0. Assuming that the *assignmentsQty* feature is detached, the prediction will reduce from 0.16 to 0.088. On the other hand, features in blue color influence negatively, *i.e.* drag the prediction value closer

TABLE 9. SHapley Additive exPlanations Prediction. Here, PC1 by (a), Baseline by (b), CM1 by (c), and finally, JM1 by (d).



Algorithm 2 Model Construction for Proposed SHAP in XAI

Require: M best ML-MODEL with fit by XTrain, YTrain data and X independent features

1: $M \leftarrow$ best ML-Model compare other algorithms
2: *explainer* $\leftarrow$ shap.TreeExplainer (M)
3: *shapValues* $\leftarrow$ *explainer*.shap_values (X)
4: shap.initjs()
5: shap.force_plot(*explainer*.expected_value, *shapValues* [0,:], X.iloc[0,:])
6: **Return:** visualization of the impact of individual features on a model's output, aiding feature importance analysis.

to 1. If the feature of *nosi* is removed, the prediction rate will rise from 0.16 to 0.4993. Therefore, software faults will not occur; thus, practitioners and stakeholders will benefit.

In 3. (b), the base value is 0.5026, and the output value is 0.85. Whether the *nosi* feature is unfastened, then the prediction will reduce from 0.85 to 0.6026, and if the *dit* feature is separated, then the prediction will increase from 0.85 to 0.9026. In 3. (c), the base value is 0.4899, and the output value is 0.19. If the *nosi* feature is removed, the prediction will expand from 0.19 to 0.5219. Furthermore, the last one, 3. (d) base value is 0.4982, and the output value is 0.85. When the *nosi* feature is extracted, the prediction will lessen from 0.85 to 0.6243, and if the *dit* feature is eliminated, the prediction will be improved from 0.85 to 0.8680.

7) COMPARATIVE RESULT OF LIME AND SHAP

In our investigation, it has been observed that LIME exhibits greater efficacy in detecting software defects within our datasets, as compared to SHAP. LIME and SHAP can be used to explain how any model works efficiently. On the one hand, LIME builds a local explanation by fitting a simple model to a prediction, which is lower in precision than the complex model but simpler to understand. However, SHAP measures the importance of every feature of a model by using game theory. Finally, both methods are used to figure out how different factors affect an expected result depending on specific use cases and priorities.

V. CONCLUSION

This study aims to develop an ML model that helps software engineers, analysts, and stakeholders find the best features of software faults.The main advantages of finding and addressing the root causes of software faults are

essential for building high-quality, reliable, and maintainable software. Moreover, it improves user satisfaction, optimal cost savings and does not encounter frequent errors or crashes. In this paper, we use SMOTTech and LE techniques for the normalization and preprocessing of missing feature values. Twenty-one characteristics were considered during the prediction analysis.

Among the thirteen machine learning methods, EGBR has a high level of accuracy in predicting the outcomes of datasets. Here, we explored why XAI is essential, and several facts about XAI were analyzed for diagnosing faults in software. Then we categorized it regarding scope and usage in the context of identifying reasons for software faults. In XAI, we performed on LIME and SHAP techniques, and LIME is shown as the best valuable feature for reasoning software faults or not for our datasets. The limitation of our research is that a small number of features for software defects have been selected. In subsequent investigations, the research scope can be expanded by adding an essential amount of data and utilizing current regression and classification techniques.

APPENDIX A
LIST OF ABBREVIATIONS

AdaBoost	AdaBoost Classifier.
AUC	Area Under the ROC Curve.
cbo	Coupling between objects.
cbo	Coupling Between Objects.
dit	Depth Inheritance Tree.
DTC	Decision Tree Classifier.
EGBR	Extreme Gradient Boosting Regression.
ETC	Extra Trees Classifier.
EV	Explained Variance.
GBC	Gradient Boosting Classifier.
GBR	Gradient Boosting Regression.
KNC	KNeighbors Classifier.
LE	Label Encoding.
LIME	Local Interpretable Model.
LOC	Lines of Code.
LR1	Linear Regression.
LR2	Logistic Regression.
MAE	Mean Absolute Error.
MAPE	Mean Absolute Percentage Error.
mathOperationsQty	Number of Math Operations.
ML	Machine Learning.
NASA	National Aeronautics and Space Administration.
NB	Naïve Bayes.
nosi	Number of static invocations.
RF	Random Forest Regression.
RMSE	Root Mean Square Error.
SHAP	SHapley Additive exPlanations.
SMOTTech	Synthetic Minority Oversampling Technique.
SSR	Sum of Squared Residuals.
SST	Total Sum of Squares.
stringLiteralsQty	String Literals.
SVM	Support Vectors Machine.
variablesQty	Number of Variables.
XAI	Explainable AI.
XGBC	XGBoost Classifier.
XGBR	XGBR Regressor.

APPENDIX B
LIST OF USED SYMBOLS

R	A random number that ranges from 0 to 1.
n	Total number of data points.
i	1, 2, 3, . . . , n Number of iteration.
f_i	Feature vector of the under-investigated minority class sample.
f_{near}	f_i's is K closest neighbors.
h_i	Average learner that exhibits an individual performance.
γ_{i}	Scaling factor that accounts for the effect of a tree on the overall model.
y_i	Actual value of Y.
$\hat{y}_i$	Predicted value of Y.
$\bar{y}$	Mean of actual value.

DATA AVAILABILITY

The processed data, trained model, and associated codes pertaining to this work may be accessed at https://github.com/duetianmehedishuvo/Software-Defects-Identification/ (accessed on 09 October 2023).

ACKNOWLEDGMENT

(Momotaz Begum, Mehedi Hasan Shuvo, and Imran Ashraf contributed equally to this work.)

REFERENCES

[1] T. Mens, "On the complexity of software systems," *Computer*, vol. 45, no. 8, pp. 79–81, Aug. 2012.
[2] M. Begum and T. Dohi, "Estimating prediction interval of cumulative number of software faults using back propagation algorithm," *ACIS Int. J. Comput. Inf. Sci.*, vol. 17, no. 2, pp. 25–34, May 2016.
[3] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *Int. J. Appl. Sci. Eng.*, vol. 17, no. 4, pp. 331–344, 2020. [Online]. Available: https://gigvvy.com/journals/ijase/articles/ijase-202012-17-4-331
[4] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," 2016, *arXiv:1602.04938*.
[5] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," 2017, *arXiv:1705.07874*.
[6] S. Roy, G. Laberge, B. Roy, F. Khomh, A. Nikanjam, and S. Mondal, "Why don't XAI techniques agree? Characterizing the disagreements between post-hoc explanations of defect predictions," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2022, pp. 444–448.
[7] 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.
[8] M. Begum and T. Dohi, "A neuro-based software fault prediction with Box-Cox power transformation," *J. Softw. Eng. Appl.*, vol. 10, no. 3, pp. 288–309, 2017. [Online]. Available: https://www.scirp.org/journal/paperinformation.aspx?paperid=74993
[9] S. Noekhah, A. A. Hozhabri, and H. S. Rizi, "Software reliability prediction model based on ICA algorithm and MLP neural network," in *Proc. 7th Int. Conf. E-Commerce Developing Countries, Focus E-Secur.*, Apr. 2013, pp. 1–15.

[10] M. Begum and T. Dohi, "Optimal release time estimation of software system using Box-Cox transformation and neural network," *Int. J. Math., Eng. Manage. Sci.*, vol. 3, no. 2, pp. 177–194, Jun. 2018.
[11] M. Begum and T. Dohi, "Optimal stopping time of software system test via artificial neural network with fault count data," *J. Quality Maintenance Eng.*, vol. 24, no. 1, pp. 22–36, Mar. 2018. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/JQME-12-2016-0082/full/html
[12] J. A. Dallal and L. C. Briand, "An object-oriented high-level design-based class cohesion metric," *Inf. Softw. Technol.*, vol. 52, no. 12, pp. 1346–1361, Dec. 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584910001552
[13] M. Begum and T. Dohi, "Optimal software release decision via artificial neural network approach with bug count data," in *Proc. 7th Asia–Pacific Int. Symp. Adv. Rel. Maintenance Modeling (APARM)*, Aug. 2016, pp. 17–24.
[14] D. Sharma and P. Chandra, "Software fault prediction using machine-learning techniques," in *Smart Computing and Informatics*. Singapore: Springer, 2018, pp. 541–549.
[15] Md. R. Islam, Md. N. Akhtar, and M. Begum, "Long short-term memory (LSTM) networks based software fault prediction using data transformation methods," in *Proc. Int. Conf. Advancement Electr. Electron. Eng. (ICAEEE)*, Feb. 2022, pp. 1–6.
[16] G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, "Understanding machine learning software defect predictions," *Automated Softw. Eng.*, vol. 27, nos. 3–4, pp. 369–392, Dec. 2020, doi: 10.1007/s10515-020-00277-4.
[17] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Softw.*, vol. 12, no. 3, pp. 161–175, Jun. 2018, doi: 10.1049/iet-sen.2017.0148.
[18] M. Begum, J. H. Rony, M. R. Islam, and J. Uddin, "Long-term software fault prediction model using Box-Cox and linear regression," *J. Inf. Syst. Telecommun. (JIST)*, vol. 11, no. 3, pp. 1–12, Feb. 2023. [Online]. Available: http://jist.ir/Article/36585
[19] B. Steenwinckel, D. De Paepe, S. Vanden Hautte, P. Heyvaert, M. Bentefrit, P. Moens, A. Dimou, B. Van Den Bossche, F. De Turck, S. Van Hoecke, and F. Ongenae, "FLAGS: A methodology for adaptive anomaly detection and root cause analysis on sensor data streams by fusing expert knowledge with machine learning," *Future Gener. Comput. Syst.*, vol. 116, pp. 30–48, Mar. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739×20329927
[20] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 2, pp. 1–6, 2018, doi: 10.14569/ijacsa.2018.090212.
[21] M. R. Islam, M. Begum, and M. N. Akhtar, "Recursive approach for multiple step-ahead software fault prediction through long short-term memory (LSTM)," *J. Discrete Math. Sci. Cryptogr.*, vol. 25, no. 7, pp. 2129–2138, Oct. 2022, doi: 10.1080/09720529.2022.2133251.
[22] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, "Explainable artificial intelligence (XAI) to enhance trust management in intrusion detection systems using decision tree model," *Complexity*, vol. 2021, Jan. 2021, Art. no. 6634811, doi: 10.1155/2021/6634811.
[23] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools," *Eng. Appl. Artif. Intell.*, vol. 111, May 2022, Art. no. 104773. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197622000616
[24] W. Zheng, T. Shen, X. Chen, and P. Deng, "Interpretability application of the just-in-time software defect prediction model," *J. Syst. Softw.*, vol. 188, Jun. 2022, Art. no. 111245. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121222000218
[25] M. Begum, M. S. Hafiz, M. J. Islam, and M. J. Hossain, "Long-term software fault prediction with robust prediction interval analysis via refined artificial neural network (RANN) approach," *Eng. Lett.*, vol. 29, p. 44, Aug. 2021.
[26] T. Menzies, J. Distefano, A. Orrego, and M. Chapman, "Assessing predictors of software defects," in *Proc. Workshop Predictive Software Models*, Jan. 2004, pp. 1–11.
[27] 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, Jun. 2002.
[28] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *J. Big Data*, vol. 7, no. 1, p. 28, Apr. 2020, doi: 10.1186/s40537-020-00305-w.
[29] D. Maulud and A. M. Abdulazeez, "A review on linear regression comprehensive in machine learning," *J. Appl. Sci. Technol. Trends*, vol. 1, no. 4, pp. 140–147, Dec. 2020. [Online]. Available: https://jastt.org/index.php/jasttpath/article/view/57
[30] R. Costa-Mendes, T. Oliveira, M. Castelli, and F. Cruz-Jesus, "A machine learning approximation of the 2015 Portuguese high school student grades: A hybrid approach," *Educ. Inf. Technol.*, vol. 26, no. 2, pp. 1527–1547, Mar. 2021, doi: 10.1007/s10639-020-10316-y.
[31] A. Abd Elrahman, T. Hassan A Soliman, A. I. Taloba, and M. F. Farghally, "A predictive model for student performance in classrooms using student interactions with an eTextbook," 2022, *arXiv:2203.03713*.
[32] D. P. Solomatine and D. L. Shrestha, "AdaBoost.RT: A boosting algorithm for regression problems," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jul. 2004, pp. 1163–1168.
[33] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers Neurorobotics*, vol. 7, p. 21, Dec. 2013. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021
[34] D. A. Otchere, T. O. A. Ganat, J. O. Ojero, B. N. Tackie-Otoo, and M. Y. Taki, "Application of gradient boosting regression model for the evaluation of feature selection techniques in improving reservoir characterisation predictions," *J. Petroleum Sci. Eng.*, vol. 208, Jan. 2022, Art. no. 109244. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920410521008998
[35] S. Lee, J. Park, N. Kim, T. Lee, and L. Quagliato, "Extreme gradient boosting-inspired process optimization algorithm for manufacturing engineering applications," *Mater. Des.*, vol. 226, Feb. 2023, Art. no. 111625.
[36] A. Sanmorino, L. Marnisah, and H. Sunardi, "Feature selection using extra trees classifier for research productivity framework in Indonesia," in *Proc. 3rd Int. Conf. Electron., Biomed. Eng., Health Inform.*, 2023, pp. 13–21.
[37] J. J. Valero-Mas, A. J. Gallego, P. Alonso-Jiménez, and X. Serra, "Multilabel prototype generation for data reduction in K-nearest neighbour classification," *Pattern Recognit.*, vol. 135, Mar. 2023, Art. no. 109190. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320322006690
[38] K. Tyagi, C. Rane, and M. Manry, "Regression analysis," in *Artificial Intelligence and Machine Learning for EDGE Computing*. Cambridge, MA, USA: Elsevier, Jan. 2022, pp. 53–63. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128240540000071
[39] K. E. O'Grady, "Measures of explained variance: Cautions and limitations," *Psychol. Bull.*, vol. 92, no. 3, pp. 766–777, Nov. 1982.
[40] A. de Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, Jun. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231216003325

MOMOTAZ BEGUM received the M.Sc. degree in CSE from the Dhaka University of Engineering and Technology (DUET), Gazipur, Bangladesh, in 2013, and the Ph.D. degree in information engineering from Hiroshima University, Japan, fully funded by the Japanese government, specifically the Ministry of Education, Culture, Sports, Science and Technology [Monbukagakusho (MEXT)], in September 2017. She is currently a Distinguished Professor with the Department of Computer Science and Engineering, DUET. Her research interests include software reliability, software engineering, artificial neural networks, data mining, data clustering, information systems, and analysis. Her specialization lies in software rejuvenation and software aging and human pose recognition. She has published several journals and conference papers in the world's reputed journals and conferences on the different topics of computer science, such as the IoT, machine learning, and neural network.

MEHEDI HASAN SHUVO received the B.Sc. degree in engineering (CSE) from the Dhaka University of Engineering and Technology (DUET), Gazipur. He is currently an Engineer. With a strong background in technical concepts and problem-solving, he combines his engineering expertise with his love for storytelling to create engaging and informative content. He is an expert in mobile application development and has developed more than 30 professional Android and iOS applications in many fields. His dedication to research and continuous learning enables him to stay up-to-date with the latest advancements in the field, ensuring that his writing reflects cutting-edge knowledge and trends, and his research interests include machine learning, neural networks, conventional neural networks, and explainable artificial intelligence.

IMRAN ASHRAF received the M.S. degree (Hons.) in computer science from the Blekinge Institute of Technology, Karlskrona, Sweden, in 2010, and the Ph.D. degree in information and communication engineering from Yeungnam University, Gyeongsan, South Korea, in 2018. He was a Postdoctoral Fellow with Yeungnam University, where he is currently an Assistant Professor with the Information and Communication Engineering Department. His research interests include positioning using next-generation networks, communication in 5G and beyond, location-based services in wireless communication, smart sensors (LIDAR) for smart cars, and data analytics.

ABDULLAH AL MAMUN received the B.Sc. degree in engineering (CSE) from the Dhaka University of Engineering and Technology (DUET), Gazipur. He is currently an Engineer. He blends his engineering knowledge with his passion for storytelling to generate ideas that are both interesting and educational. He has a solid foundation in technological concepts and problem-solving. He has experience in web development and has created more than five expert websites in various industries. His commitment to study and ongoing education allows him to stay current with the most recent developments and a wealth of expertise in the subject, ensuring that his work represents the most recent trends and information.

JIA UDDIN received the M.Sc. degree in telecommunications from the Blekinge Institute of Technology, Sweden, in 2010, and the Ph.D. degree in computer engineering from the University of Ulsan, South Korea, in January 2015. He is currently an Assistant Professor with the AI and Big Data Department, Endicott College, Woosong University, South Korea. He is also an Associate Professor (currently on leave) with the CSE Department, BRAC University, Bangladesh. He was a member of the Self-Assessment Team (SAC) of CSE, BRACU, in the HEQEP project funded by the World Bank and the University of Grant Commission Bangladesh (2016–2017). His research interests include fault diagnosis using AI, audio, and image processing.

MD ABDUS SAMAD (Member, IEEE) received the Ph.D. degree in information and communication engineering from Chosun University, Gwangju, South Korea. He was an Assistant Professor with the Department of Electronics and Telecommunication Engineering, International Islamic University Chittagong, Chattogram, Bangladesh, from 2013 to 2017. He has been a Research Professor with the Department of Information and Communication Engineering, Yeungnam University, South Korea. His research interests include signal processing, antenna design, electromagnetic wave propagation, the applications of artificial neural networks, deep learning, and millimeter-wave propagation by interference and/or atmospheric causes for 5G and beyond wireless networks. He won the prestigious Korean Government Scholarship for his doctoral study.