International Journal of Electrical and Computer Engineering (IJECE)
Vol. 13, No. 6, December 2023, pp. 7027~7036
ISSN: 2088-8708, DOI: 10.11591/ijece.v13i6.pp7027-7036

A simplified predictive framework for cost evaluation to fault assessment using machine learning

Deepti Rai, Jyothi Arcot Prashant
Department of Computer Science and Engineering, Faculty of Engineering and Technology, Ramaiah University of Applied Sciences, Bengaluru, India

Article Info

Article history:

Received Feb 27, 2023
Revised Jul 11, 2023
Accepted Jul 18, 2023

Keywords:

Cost evaluation
Machine learning
Prediction
Software engineering
Software fault

ABSTRACT

Software engineering is an integral part of any software development scheme which frequently encounters bugs, errors, and faults. Predictive evaluation of software fault contributes towards mitigating this challenge to a large extent; however, there is no benchmarked framework being reported in this case yet. Therefore, this paper introduces a computational framework of the cost evaluation method to facilitate a better form of predictive assessment of software faults. Based on lines of code, the proposed scheme deploys adopts a machine-learning approach to address the perform predictive analysis of faults. The proposed scheme presents an analytical framework of the correlation-based cost model integrated with multiple standards machine learning (ML) models, e.g., linear regression, support vector regression, and artificial neural networks (ANN). These learning models are executed and trained to predict software faults with higher accuracy. The study considers assessing the outcomes based on error-based performance metrics in detail to determine how well each learning model performs and how accurate it is at learning. It also looked at the factors contributing to the training loss of neural networks. The validation result demonstrates that, compared to logistic regression and support vector regression, neural network achieves a significantly lower error score for software fault prediction.

Corresponding Author:

Deepti Rai
Department of Computer Science and Engineering, Faculty of Engineering and Technology, Ramaiah University of Applied Sciences
Bengaluru, India
Email: deeraisecond@gmail.com

1. INTRODUCTION

The advancements in computing methods, embedded system, communication standards, and artificial intelligence (AI) has disrupted the whole ecosystem of the conventional approaches of the software scope [1], [2]. However, the software development process always aims to meet the quality and safety of all software vulnerabilities [3], [4]. The faults that may cause a functional error or vulnerabilities must be detected and removed. However, it can happen only when the faults are localized. Localizing faults in any system, including software, is the most expensive and challenging requirement before handling the faults [5]. Recently, many novel and fundamental approaches have been proposed for fault localization, providing new dimensions and scope [6]–[13]. The broader classification of the fault localization techniques (FLT) is given into two classes: i) static-FLT and ii) dynamic FLT [6]. However, static FLT in the context of software programs checks for bugs or defects in the source code concerning the standard templates of the program models [14]. On the other hand, dynamic-FLTs work on the principle that it does not have prior knowledge of the program and tag only the correctness based on its run-time behavior [15]. The data-driven approach of the fault locations is significant for achieving fault localization and hence fault prediction.

Existing studies carried out in this direction are briefed as follows: The work of McCabe [15] considers a program: "P" having a complexity metric: {v} that tests a number of the path: {ac: actual complexity}. The correlation between the 'v' and 'ac' defines whether the program is more error-prone or less. Though this technique is quite fundamental in terms of a mathematical model, it lacks scalability [16]. Bal and Kumar [17] discusses various machine learning techniques for software fault prediction and their strengths and limiting factors. The prime interest lies in reviewing extreme learning machines (ELM) for predicting several software faults. Another recent study by Kumar *et al.* [18] introduced an unsupervised approach considering the threshold derivation technique. The study findings exhibit that most recently introduced fault prediction approaches consider labeled datasets. There are relevant studies by Arshad *et al.* [19], Aziz *et al.* [20], and Singh *et al.* [21], which have also focused on the software quality assurance aspect through software fault prediction. Liu *et al.* [22] have presented an empirical study for two-stage data processing in the context of software fault prediction along with pre-processing based on a threshold-based clustering method. Arshad *et al.* [23] also introduced another fuzzy-based data-driven approach to deal with software quality assurance considering semi-supervised Fuzzy-C mean clustering for software fault prediction analysis. On the other hand, the approach of Riaz *et al.* [24] addresses the class imbalance problem and presents an approach based on under-sampling. The experimental outcome shows this approach has a high classification rate and addresses the biases toward majority class samples. The two-stage data pre-processing includes feature selection and rough set-based k-nearest neighbor (k-NN) ruleset-based noise filter and executes easy ensemble approach. Aziz *et al.* [25] addressed the complex methodologies involved in object-oriented designs in software fault prediction. It advocates and encourages the importance of the inheritance factor in predicting software fault proneness. The study also evaluates and explores how much inheritance metrics are required to predict software fault proneness. The outcome of the study based on the evaluation of artificial neural network (ANN) also suggested that a high inheritance metric is not required as it can potentially lead to software faults. The imbalance data distribution problem is also extensively studied by Hassouneh *et al.* [26] to Rathore and Kumar [27] to strengthen the feature selection approach in software fault prediction. The most commonly used classifiers are k-NN, decision trees (DT), and linear discriminant analysis (LDA).

The identified research problems are as follows: i) very less emphasis on Imbalanced data: It is observed that very less focus inclines towards handling the problem of imbalanced software fault data. The class imbalance problem is also less likely to be explored in the existing system. This could affect the learning accuracy performance for machine learning (ML)-based prediction approaches; ii) lack of nonlinear attribute relations: fewer studies measure the utility of features and exploit the nonlinear interaction among attributes. Also, very few studies have emphasized optimizing feature selection performance by incorporating better transfer functions (TF) [28], [29]; iii) less consideration of software metrics in the dataset: in the existing studies, the data set considered majorly does not indicate the presence of software metrics and inheritance factors along with fault values which also restricted their scope from attaining desirable performance towards software fault prediction; and iv) complexity in deep learning (DL) models: ML-based approaches have been widely adopted for software fault prediction and produce varied results regarding software faults. Despite of popularity DL based approaches, specifically multi-layer perception (MLPs) and convolutional neural networks (CNN) might result in better performance of fault prediction concerning the accuracy of fault detection rate measurement but when it comes to computational performance, then both the techniques lack efficiency.

The research work's motivation is as follows: McCabe's work argued that the codes having higher complexities would have more errors. The line of code (LOC) plays an important role in both efforts, and the project schedule and fault correlate with both factors. However, there are many more metrics in the context of fault prediction that typically includes: i) cyclomatic complexity (McCabe's complexity), ii) Halstead metrics, iii) McCabe essential complexity (MEC) metric, iv) the McCabe module design complexity (MMDC) metric, and v) object-oriented metrics [30]. However, a systematic study by Radjenović *et al.* [31] concludes that the use of data set is 58% and 33% for private and small datasets, respectively. Therefore, more research shall be carried out on the publicly available dataset to meet the benchmark requirements and its real-time relevancy.

The prime goal of the proposed scheme is to address the problem mentioned above by introducing a novel analytical model for fault prediction. The contributions in this paper are as follows: i) the study introduces an optimized correlative cost model framework that incorporates a program to evaluate three popular ML approaches towards software fault prediction. The approach of supervised learning initially exploits the fault data from the statistical point of view and extracts the valuable features for training; ii) the study also addresses the problem of imbalanced data and considers balanced data of software metric and fault for the construction of data frame and also the statistical correlation metric provide better insights of the metric software features which could be useful during the learning process where the nonlinear relationship among data attributes are addressed; iii) the approach of supervised learning for logistic regression (LR), support vector regression

(SVR), and ANN considers optimization for the learning process, which targets reducing the training error to a greater extent which has not been studied much in the past; iv) the response variable of thousand (k) lines of codes (KLOC) indicates the possibility of faults and having a relationship with the cost modeling, which is also formulated in this proposed study; v) study also constructed the model of ANN with optimized execution flow of the Adam optimizer, which has resulted in smoother execution of the ANN learning with considerable errors. It also shows that the training loss performance has been significantly improved and gradually decreases in the case of ANN; and vi) the study provides a detailed experimental outcome that justifies the suitability of the three different learning models for predicting software fault and shows how accurately a model can learn.

2. METHOD

The proposed system adopts an analytical research method where the very fundamental input for the learning model is the historic dataset. Let us consider dataset D, which is the pair value of $\{x, y\}$. As per the basic fundamental principle of learning a model, the dataset is subjected to be divided into two segments represented by two variables. Here x represents the predictor variables, whereas y corresponds to the response class. Figure 1 clearly shows that the $\{x, y\}$ value pairs are further considered in the learning model where trainable parameters are m and c, which refer to the slope and y-intercept, respectively. The optimizer is basically incorporated to optimize the learning performance and aims to reduce the empirical error corresponding to the predicted response. The loss function incorporated in Figure 1 shows how 'Yp' and 'Ya' variables are used to compute the training loss, which is also referred to in the proposed correlative cost modeling of software fault prediction. The study considers this baseline strategy of learning model as a foundation to develop this framework where three ML-based approaches of LR, SVR, and ANN are evaluated. The system model corresponding to the proposed framework is further illustrated below.

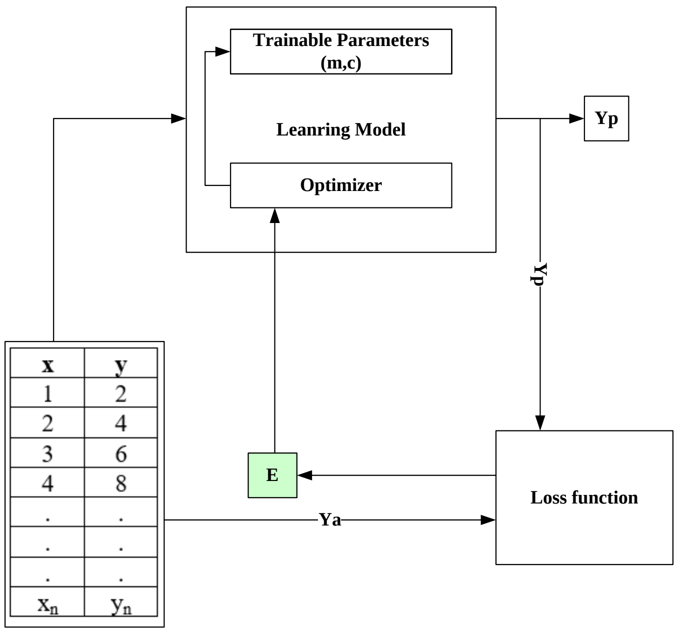


Figure 1. Architectural model for the learning mechanism from the dataset

The study considers a training set for the class of software fault data (F). Each data frame (d) corresponds to F in the dataset. The study also aims to evaluate the function of the line of code, which indicates the cost measure for software metrics. The study considers the reference F dataset, which contains a training set for class software faults.

2.1. LR learning model

In the first predictive model, fault prediction is realized as a regression problem. In this model design, it is assumed that the numeric output r is the summation of the deterministic function of the input and additional random noise($\in$). It is an extra hidden variable that cannot be observed. This can be realized with the following mathematical as (1),

$$r = f(x) + \in \quad (1)$$

here the unknown function $f(x)$ is to be approximated considering the estimator $g(\cdot)$. Here the values of x indicate the training data of software fault, and this supervised approach of ML fits a function *f*(.). To this, x is to learn y as a function of x. Let us assume the input attributes of the predictor variable for the software fault data is x and can be represented using (2). Each training sample for software fault data can also be represented as an ordered pair of (x, y). If the training set consists of the total S number of samples, (3) shows the representation of the ordered pair.

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} \quad (2)$$

$$X = \{x^t, r^t\}_{t=1}^{S} \quad (3)$$

here $r^t \in R$ and also $f(x)$ is unknown with random noise $\in$ from (1). The software fault dataset contains an x_n predictor variable where $n = 25$. The fitted function can also be realized using as (4) and (5),

$$y = wx + w_0 \quad (4)$$

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n + \in \quad (5)$$

here w, and w_0 consist of suitable numerical values representing slope and intercept, respectively, for (4). In (5), the software fault prediction problem is further generalized as a multiple linear regression problem for different regression coefficients and independent variables. As shown in (2) and (3) highlight that the LR learning model is considered a supervised learning problem. The function approximation also takes place considering the numerical adjustment of weight factors w. In the context of ML for predicting a software fault, the task for the LR model is to formulate a mapping between $x \rightarrow y$. Machine learning in this software fault prediction context is that the model is defined concerning a set of parameters which can be expressed with the (6).

$$y = g(x|\theta) \quad (6)$$

In (6), y produces a number in regression outcome from evaluating the model $g(\cdot)$. Here θ indicates the model parameters. The regression function $g(\cdot)$ is modeled in such a way that the ML program aims to optimize the parameters θ in the given function. The prime motive is to minimize the approximation error so that the estimated outcome would become closer to the actual values given in the software fault data training set. The prime aim is to construct the $g(\cdot)$. That can reduce the empirical error. If $g(x)$ is linear, then it can be generalized using as (7).

$$g(x) = \sum_{j=1}^{S} w_jx_j + w_0 \quad (7)$$

The linear model constructed for the prediction of response in r in reference to y shows that it estimates the response for software fault prediction, which can also be evaluated in the cost measure (C). Here the function is approximated by learning from the data where the parameters w_j and w_0 learns from the data x. The values of w_j and w_0 minimize the empirical error in the training set concerning the following as (8).

$$E(g|x) = \frac{1}{S}\sum_{t=1}^{S}[r^t - g(x^t)]^2 \quad (8)$$

The design model of the presented LR is customized by applying the estimator $g(x|\theta)$ on $f(x)$. The estimator approximates the $f(x)$ response for software fault prediction in the measure of cost. Here the set of parameters θ is also defined for the measure of learning. It is assumed that the value of $\in$ is zero-mean Gaussian with the constant variance of σ^2. Then the normalized value approximation for random noise can be represented with $\in \sim N(0, \sigma^2)$ for normal distribution. The normal distribution here basically approximates the discrete distribution. The substitution of the estimator $g(\cdot)$ in the place of $f(x)$ provides the probability of the output given in input using as (9).

$$p(r|x) \sim N(g(x|\theta), \sigma^2) \quad (9)$$

As shown in (9) depicts the normal distribution computation of the x values of software fault data considering joint probability density. This indicates in this linear model of software fault detection; 0 mean Gaussian noise is added. The LR model in this proposed framework inherits normal distribution's standard and potential properties to study the software fault data [24]. The study also considers maximum likelihood computation ($\mathcal{L}$) to learn the parameters defined in θ. The pair $\{x^t, r^t\}_{t=1}^{S}$ in training, sets are drawn considering the unknown probability density measure of $p(x, r)$, which can be mathematically expressed as (10),

$$p(x, r) = p(r|x)p(x) \tag{10}$$

here, the $p(x, r)$ represents the unknown joint probability density estimation, measured as a product of the output given the input. Also, $p(x)$ indicates the probability of the input density. The input of x and the parameters θ in the model $g(x|\theta)$ also formulates the problem of multiple linear regression. The learning algorithm is further designed considering the training set of fault data in $p(x, r)$. The approximation error is further computed considering a loss function of r and g (x|θ). The approximation error for the loss function (E) is designed using as (11),

$$E(\theta|x) = \sum_t L(r^t, g(x^t|\theta)) \tag{11}$$

When learning the class corresponding to a software fault, the LR model uses the square of a difference considering (9). The LR model in this proposed study also applies an optimization procedure to extract θ^* This can minimize the error corresponding to predicted or approximated response of software fault, which can be represented using as (12).

$$\theta^* = \text{ArgMin } E(\theta|x) \tag{12}$$

The depreciation of error E also shows how accurately the model of the approximating function of $g(x|\theta)$ is learned concerning software fault's intercept and slope data. The response class of KLOC is also considered in the dataset. Here KLOC: the indicator (L), i.e., thousands of lines of code, refers to how large a computer program is. More line of code indicates the possibilities associated with more fault occurrence. The presented approach to learning considers $y = g(x|\theta)$ and divides the training dataset of software faults accordingly. The customized function of the model of LR fits the training data considering $y = g(x|\theta)$, here the training of the model takes place concerning x, y. A closer observation of the predictor variable xn shows that during the splitting of x and r from x_r of software fault data x_1:x_{24} becomes the predictors where ($y \leftarrow x_{25}$) becomes the response variable (r) from the data frame dϵF (which indicates the KLOC value). The study invokes a customized functional module of $\Theta_{split}(x)$, which considers the input of $x = x_1 \rightarrow x_{24}$ and $y = x_{25}$ with test size ($t_{size} = 0.2$) and random state (rstate) and computes the data of $[x_{train}, y_{train}]$. The fitness of the model $g(x|\theta)$ is further evaluated concerning $[x_{train}, y_{train}]$, this ensures the learning from the data. Finally, the software fault prediction model outcome generates the response of y considering the test data (x_{test}). Estimating LR model learning and prediction accuracy is further performed considering a set of performance metrics such as mean squared error (MSE), root mean square error (RMSE), mean absolute error (MAE), and mean magnitude of relative error (MMRE).

2.2. SVR learning model

The core idea of the study also incorporates an SVR-based learning model for predicting software fault data concerning the cost measure of KLOC. The model of the SV algorithm follows a nonlinear generalization metric. The idea of SVR is to minimize the generalization error bound instead of observed training error to achieve generalized performance in the context of software fault prediction. It is functionally modeled based on the computing procedure of LR in a high-dimensional feature space. It has to be noted that in that feature space, the input data are mapped via a nonlinear function. The core design of SVR in the proposed framework of the correlative cost model of software fault prediction considers input training data in the form of $\{(x_1, y_1), \ldots\ldots, (x_n, y_n)\} \subset \mathcal{X} \times \mathbb{R}$. Here $\mathcal{X}$ represents the space of fault data pattern for the input instance $\mathcal{X} = \mathbb{R}^d$. Regarding $\mathcal{E}$-SVR, the prime target is to compute the function $f(x)$ That yields a maximum $\mathcal{E}$ for deviation concerning the gradually obtained response of software fault in y. The y is obtained for all the training data (d) related to software faults. The analysis in the context of SVR does not care much about the errors as long as the $\text{error} < \mathcal{E}$. However, it does not permit any deviation larger than $\mathcal{E}$. The linear function of $f(x)$ parameterization in SVR can be described using (13). The SVR constructs this software fault prediction problem as a convex optimization problem. It aims to minimize the$||w||^2 = \langle w, w \rangle$. The convex optimization problem in the proposed software fault prediction context can be formulated using (14).

$$f(x, w) = \langle x \cdot w \rangle + b;\; \text{Here } w \in \mathcal{X}.\, b \in \mathbb{R} \tag{13}$$

$$\text{minimize } \frac{1}{2}\|w\|^2 \;;\; \text{Subject to } \begin{cases} y_i - \langle w, x_i \rangle - b \le \varepsilon \\ \langle w, x_i \rangle + b - y_i \le \varepsilon \end{cases} \tag{14}$$

The presented study evaluates the model of SVR on the learning process from the software fault dataset of $\langle x \cdot w \rangle$ Where x represents the predictor variables. The core objective of the function $f(\cdot)$ is to approximate the value pairs of $\{x: y\}$ with a considerable outcome of ε. The model also checks whether the convex optimization problem is feasible for all the inputs of $\{x: y\}$ from the software fault dataset. To deal with the other infeasible constraints of optimization in SVR slack variables ξ, ξ^* Are also introduced. This approximates the trade-off between the flatness of $f(\cdot)$ and the amount up to which the range of deviations > ε are tolerated.

2.3. Learning model for ANN

The study also considered the significant aspects of ANN and their AI origins to address this software fault prediction problem. In the ANN model, information input is propagated among a network of nodes. The nodes in this model mathematically interact with each other, which the user does not know. Eventually, the model computes an output for the value pair of $\{x: y\}$, formulating a relationship between x and y. This further map the expected macroscopic input-output pattern of the relationship. The interaction between nodes is adjusted until the model finds desired input-output outcome. The prime construct of ANN considers three distinct layers with interconnection among nodes. The ANN constructs three prime layers where the input layer receives information from external sources and further relays the information to the ANN for model processing. The hidden layer processes the information received from the input layer and further passes it to the output layer. The output layer receives the processed information and sends it to the external receptor. In the proposed study, the ANN constructs the model to retain information by connecting nodes with neighbors and the magnitudes of the signals. The ANN in the presented study addresses the problem of imbalanced data and considers processing noisy, incomplete, and inconsistent information. Here each node encodes a micro-feature of the input-output pattern. One novelty aspect of this model is that, unlike other computational techniques, this approach considers micro-feature computation. Here each node acts as a processing element to deal with the software fault data of $\{x: y\}$. In each PE of ANN, most of the calculations are performed. The *jth* node of the processing element considers an input vector of x with the components of $x_1 \rightarrow x_n$ and perform processing. This yields the output of y as a function $f(x, w)$. The ANN model in the proposed system of software fault prediction considers the (15) mathematical expressions.

$$y_j = f\left(\sum x_j w_{ij} - T_j\right) \tag{15}$$

Here every input is multiplied with its corresponding weight factor, and the weighted inputs are considered in each node for further calculations. Here the threshold T_j for jth node control the activation of the node. Similarly, for all n number of nodes, the total activation can be computed using (16).

$$\text{Total Activation} = \sum_{i=1}^{n} (x_j w_{ij} - T_j) \tag{16}$$

In the proposed system of software fault prediction, the study considers a sequential model of ANN and further construct x_{train} and prepares the input layers for x_{train}. The proposed execution modeling of ANN incorporates adam optimizer to optimize the execution flow of software fault prediction and also aims to reduce the training error for software fault detection. The study has found that training loss has significantly dropped over incremental epochs, ensuring that the optimized model has attained better learning accuracy for software fault prediction. The models mentioned above of LR, SVR, and ANN consider the training and testing data of $[x_{train}, y_{train}]$ to fit the models and further with x_{test}, the models perform prediction of the software fault data. The next section further discusses the experimental outcome obtained from simulating these ML-based software fault prediction approaches.

3. RESULTS

This section illustrates the experimental outcome obtained after simulating the proposed correlative cost framework for software fault prediction in a numerical simulation environment. The study considers the interactive scientific computing tool Jupyter Notebook to realize these models where initially the framework extracts the data frame (d) from the fault data F. The system configuration considers 64-bit windows operating system supported by an Intel i7-processor with CPU@260 GHz, 2.59 GHz, and 16.0 GB internal memory. The

Jupyter notebook application uses the local host: the 8888 servers, and enables the kernel processes. The original format of the dataset comes as "attribute relation file format (Arff)," which is suitable for a machine learning application, namely "WEKA," and this dataset is quite suitable for the regression task. It consists of '10885' different projects with the varied line of codes (LoC) ranging between [Max, Min]. The computing of the data frame $d\epsilon F$ can be visualized in Table 1.

Table 1. Computation of data frame d for $\{x: y\}$

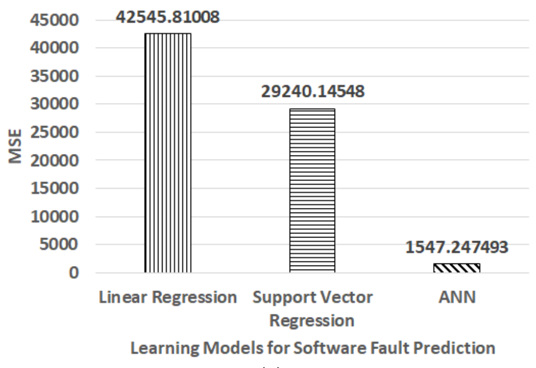
No.	Record Number	ACT_EFFORT	Model	….	tool	sced	site	Docu	Physical Delivered KLOC
0	1	117.6	cocomoII	….	1.17	1.14	0.93	1	33.0
1	2	117.6	cocomoII	….	1.17	1.14	0.93	1	31.1
2	3	31.2	cocomoII	….	1.17	1.14	0.93	1	9.5
3	4	36	cocomoII	….	1.17	1.14	0.93	1	8.4

Table 1 shows that the system initially computes the data frame considering the input dataset [32] by reading and importing the dataset into the system's scope of execution. The table is constructed as a matrix of 5 rows × 26 columns. Further, the study also considers computing the definition of the data frame where most of the predictor variables are found of type float64. In contrast, the variable record number and model are considered objects. The model considered in the dataset is referred to as 'cococmoII.' Further, the framework also performs statistical computation to visualize the descriptive statistics from the data. The study considers the described count measure for the software fault data, which are further considered as training samples in the form of variables followed by mean computation for software fault training data from the statistical point of view. The preliminary statistical evaluation shows that the mean values are quite higher in the case of docu and ACT_EFFORT. A similar computation is also performed for standard deviation computation. The outcome of the standard deviation variables for the training samples exhibited higher scores for the predictor variables docu and ACT_EFFORT. Further, min and max computation is carried out for the software fault data training variables and it is tabulated in Table 2.

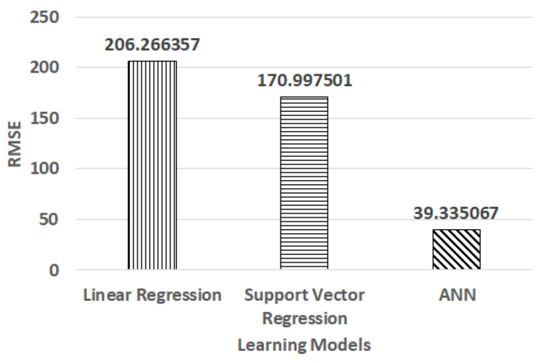
Table 2. Descriptive statistics of $\{x: y\}$

	count	mean	std	min	25%	50%	75%	max
ACT_EFFORT	124	563.33468	1029.2279	6	71.5	239.5	581.75	8211
prec	124	3.11	1.292409	0	2.48	2.48	4.96	4.96
flex	124	2.618952	1.041618	0	2.03	2.03	4.05	5.07
resl	124	3.688871	1.403707	0	2.83	2.83	5.65	6.01
team	124	1.837097	1.094185	0	1.1	1.1	3.29	4.66
pmat	124	5.602984	1.288265	2.84	4.68	4.68	6.24	7.8
rely	124	1.078522	0.103427	0.85	1	1.1	1.1	1.74
cplx	124	1.189892	0.163256	0.87	1.17	1.17	1.2125	1.74
data	124	1.014919	0.117179	0.9	0.9	1	1.14	1.28
ruse	124	0.996935	0.014605	0.95	1	1	1	1.07
time	124	1.124516	0.184476	1	1	1	1.29	1.63
stor	124	1.107097	0.163149	1	1	1	1.17	1.46
pvol	124	0.927406	0.095456	0.87	0.87	0.87	1	1.15
acap	124	0.880276	0.101079	0.71	0.85	0.85	1	1.016667
pcap	124	0.918817	0.085625	0.76	0.88	0.895	1	1
pcon	124	1.000544	0.035766	0.81	1	1	1	1.205
apex	124	0.925712	0.083496	0.81	0.88	0.88	1	1.22
plex	124	1.00459	0.080974	0.91	0.91	1	1	1.19
ltex	124	0.966781	0.089415	0.91	0.91	0.91	1	1.2
tool	124	1.115847	0.078542	0.83	1.09	1.17	1.17	1.17
sced	124	1.043065	0.06376	1	1	1	1.14	1.14
site	124	0.92504	0.017623	0.86	0.93	0.93	0.93	0.9475
docu	124	1.02494	0.05783	0.91	1	1	1.11	1.23
Physical delivered KLOC	124	103.4439	141.45589	0	20	51.9	131.75	980

The study further compares the training outcome for LR, SVR, and ANN concerning the standard performance metrics of MAE, MSE, RMSE, and MMRE. The computational analysis of MSE and RMSE is illustrated in Figure 2. Figure 2(a) clearly shows the comparison of model validation outcomes for the measure of MSE. It exhibits that the MSE score of ANN is much lesser, approximately 1548, and comparatively better than LR and SVR. On the other hand, SVR also attains considerable outcome of MSE during the learning of the model, which is approximately 29240.1. In the case of RMSE Figure 2(b) also, ANN produces better outcomes considering the efficient performance of training loss. The prime reason behind the suitable learning performance of ANN is that it incorporates Adam optimizer, and the parameter settings for processing elements also resulted in better training performance towards software fault prediction. The computational analysis of MAE and MMRE is illustrated in Figure 3. The outcome of both MAE in Figure 3(a) and MMRE in Figure 3(b) also shows the effectiveness of the ANN in predicting the software fault through the estimation of KLOC, which also ensures productivity in software industries can be managed accordingly.

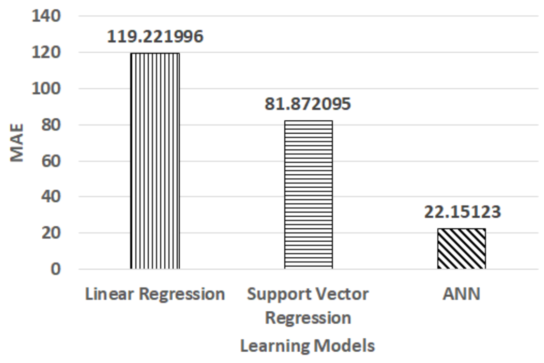


(a)

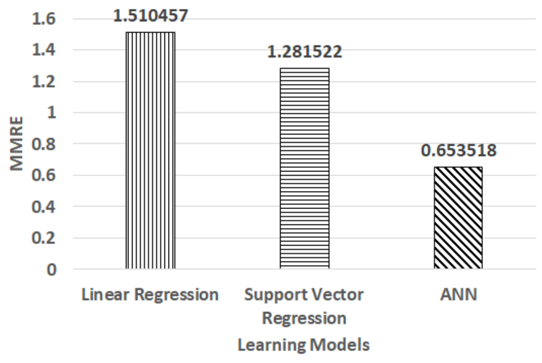


(b)

Figure 2. Computational analysis of (a) MSE and (b) RMSE



(a)



(b)

Figure 3. Computational analysis of (a) MAE and (b) MMRE

The prime novelty of the proposed outcome is manifold: i) the ANN model exhibits approximately 98.97% reduced MSE, 97% of reduced RMSE, 98.78% of reduced MAE, and 54.6% of reduced MMSE as compared to conventional LR and SVR model and ii) it was also found that overall processing time of ANN is 0.4337 s while that of LR and SVM scheme is approximately 1.107 s and 0.8336 s respectively. On the basis of this outcome, it can be eventually stated that ANN offers better predictive performance with higher accuracy towards fault prediction in software design. Hence, better form of cost-effective predictive modelling is presented in proposed scheme.

4. CONCLUSION

The study introduces a numerical framework of correlative cost modeling for software fault prediction considering three popular learning models: LR, SVR, and ANN. The proposed system model considers a standard software fault dataset and evaluates these three models for prediction, considering numerical modeling and implementation. This work's novelty is that it addresses the data imbalance problem and optimizes the performance of learning models to minimize the empirical error of the predicted response class for KLOC. The study considers KLOC as a response variable to predict the possibility of faults in the code. The experimental outcome clearly shows that ANN outperforms the other models in learning accuracy among LR, SVR, and ANN. It clearly shows that the Adam optimized in ANN not only exhibits its considerable execution performance but also ensures very negligible training loss and learning errors in the measure of MAE, MSE, RMSE, and MMRE. This indicates that with the training data, the ANN model has been trained effectively, and it maximizes the possibility of accurate fault prediction from the considered dataset. Another interesting point to be noted in this implementation is that in order to carry out analysis of fault tolerance in software engineering, proposed machine learning based approach offers cost effective solution and does not demand any adoption of complex form of new evolving deep learning schemes. This approach can help the companies maximize their profit by minimizing the cost of production and deployment of line of codes in software programs. The future work of the proposed scheme will be further carried out towards optimizing more software metrics considering more number of risk factors and uncertainties.

REFERENCES

[1] H. H. Olsson and J. Bosch, "Going digital: Disruption and transformation in software-intensive embedded systems ecosystems," *Journal of Software: Evolution and Process*, vol. 32, no. 6, Jun. 2020, doi: 10.1002/smr.2249.

[2] N. S. Sewpersadh, "Disruptive business value models in the digital era," *Journal of Innovation and Entrepreneurship*, vol. 12, no. 1, Jan. 2023, doi: 10.1186/s13731-022-00252-1.

[3] R. Widyasari *et al.*, "On the influence of biases in bug localization: evaluation and benchmark," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2022, pp. 128–139, doi: 10.1109/SANER53432.2022.00027.

[4] R. Croft, Y. Xie, and M. A. Babar, "Data preparation for software vulnerability prediction: a systematic literature review," *Prepr. arXiv.2109.05740*, Sep. 2021.

[5] G. Mahajan and N. Chaudhary, "Design and development of novel hybrid optimization-based convolutional neural network for software bug localization," *Soft Computing*, vol. 26, no. 24, pp. 13651–13672, Dec. 2022, doi: 10.1007/s00500-022-07341-z.

[6] X. Xiao, Y. Pan, B. Zhang, G. Hu, Q. Li, and R. Lu, "ALBFL: A novel neural ranking model for software fault localization via combining static and dynamic features," *Information and Software Technology*, vol. 139, Nov. 2021, doi: 10.1016/j.infsof.2021.106653.

[7] A. Maru, A. Dutta, K. V. Kumar, and D. P. Mohapatra, "Software fault localization using BP neural network based on function and branch coverage," *Evolutionary Intelligence*, vol. 14, no. 1, pp. 87–104, Mar. 2021, doi: 10.1007/s12065-019-00318-2.

[8] M. Wardat, W. Le, and H. Rajan, "DeepLocalize: Fault localization for deep neural networks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, pp. 251–262, doi: 10.1109/ICSE43902.2021.00034.

[9] Y. Kucuk, T. A. D. Henderson, and A. Podgurski, "Improving fault localization by integrating value and predicate based causal inference techniques," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, pp. 649–660, doi: 10.1109/ICSE43902.2021.00066.

[10] A. Dutta and S. Godboley, "MSFL: a model for fault localization using mutation-spectra technique," in *LASD 2021: Lean and Agile Software Development*, 2021, pp. 156–173.

[11] Y. Lou *et al.*, "Boosting coverage-based fault localization via graph-based representation learning," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Aug. 2021, pp. 664–676, doi: 10.1145/3468264.3468580.

[12] D. Ghosh and J. Singh, "Spectrum-based multi-fault localization using chaotic genetic algorithm," *Information and Software Technology*, vol. 133, May 2021, doi: 10.1016/j.infsof.2021.106512.

[13] Q. I. Sarhan and A. Beszedes, "A survey of challenges in spectrum-based software fault localization," *IEEE Access*, vol. 10, pp. 10618–10639, 2022, doi: 10.1109/ACCESS.2022.3144079.

[14] Z. Shen and S. Chen, "A survey of automatic software vulnerability detection, program repair, and defect prediction techniques," *Security and Communication Networks*, vol. 2020, pp. 1–16, Sep. 2020, doi: 10.1155/2020/8858010.

[15] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.

[16] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: a survey," *Expert Systems with Applications*, vol. 172, Jun. 2021, doi: 10.1016/j.eswa.2021.114595.

[17] P. R. Bal and S. Kumar, "WR-ELM: Weighted regularization extreme learning machine for imbalance learning in software fault prediction," *IEEE Transactions on Reliability*, vol. 69, no. 4, pp. 1355–1375, Dec. 2020, doi: 10.1109/TR.2020.2996261.

[18] R. Kumar, A. Chaturvedi, and L. Kailasam, "An unsupervised software fault prediction approach using threshold derivation," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 911–932, Jun. 2022, doi: 10.1109/TR.2022.3151125.

[19] A. Arshad, S. Riaz, L. Jiao, and A. Murthy, "The empirical study of semi-supervised deep fuzzy C-mean clustering for software fault prediction," *IEEE Access*, vol. 6, pp. 47047–47061, 2018, doi: 10.1109/ACCESS.2018.2866082.

[20] S. R. Aziz, T. A. Khan, and A. Nadeem, "Efficacy of inheritance aspect in software fault prediction—a survey paper," *IEEE Access*, vol. 8, pp. 170548–170567, 2020, doi: 10.1109/ACCESS.2020.3022087.

[21] P. Singh, N. R. Pal, S. Verma, and O. P. Vyas, "Fuzzy rule-based approach for software fault prediction," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 5, pp. 826–837, May 2017, doi: 10.1109/TSMC.2016.2521840.

[22] W. Liu, S. Liu, Q. Gu, J. Chen, X. Chen, and D. Chen, "Empirical studies of a two-stage data preprocessing approach for software fault prediction," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 38–53, Mar. 2016, doi: 10.1109/TR.2015.2461676.

[23] A. Arshad, S. Riaz, L. Jiao, and A. Murthy, "Semi-supervised deep fuzzy C-mean clustering for software fault prediction," *IEEE Access*, vol. 6, pp. 25675–25685, 2018, doi: 10.1109/ACCESS.2018.2835304.

[24] S. Riaz, A. Arshad, and L. Jiao, "Rough noise-filtered easy ensemble for software fault prediction," *IEEE Access*, vol. 6, pp. 46886–46899, 2018, doi: 10.1109/ACCESS.2018.2865383.

[25] S. R. Aziz, T. A. Khan, and A. Nadeem, "Experimental validation of inheritance metrics' impact on software fault prediction," *IEEE Access*, vol. 7, pp. 85262–85275, 2019, doi: 10.1109/ACCESS.2019.2924040.

[26] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, "Boosted whale optimization algorithm with natural selection operators for software fault prediction," *IEEE Access*, vol. 9, pp. 14239–14258, 2021, doi: 10.1109/ACCESS.2021.3052149.

[27] S. S. Rathore and S. Kumar, "An approach for the prediction of number of software faults based on the dynamic selection of learning techniques," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 216–236, Mar. 2019, doi: 10.1109/TR.2018.2864206.

[28] S. Pal and A. Sillitti, "Cross-project defect prediction: A literature review," *IEEE Access*, vol. 10, pp. 118697–118717, 2022, doi: 10.1109/ACCESS.2022.3221184.

[29] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 287–300, Mar. 2008, doi: 10.1109/TSE.2007.70768.
[30] I. Batool and T. A. Khan, "Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review," *Computers and Electrical Engineering*, vol. 100, May 2022, doi: 10.1016/j.compeleceng.2022.107886.
[31] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, Aug. 2013, doi: 10.1016/j.infsof.2013.02.009.
[32] J. David, "Recommending software artifacts from repository transactions," in *New Frontiers in Applied Artificial Intelligence*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 189–198.

BIOGRAPHIES OF AUTHORS

Deepti Rai completed her Master degree in 2016 and Bachelor degree in 2004 from Visvesvaraya Technological University, India. She is currently pursuing her Doctoral degree in the domain of ML at the Department of Computer Science and Engineering, Ramaiah University of Applied Sciences, Ramaiah Technology Campus, Bengaluru, Karnataka, India. She has 10 years of experience in teaching and 6 years of industry experience. Her research interest is in the field of ML, DL, AI and cloud computing. She can be contacted at email deeraisecond@gmail.com.

Jyothi Arcot Prashant completed her PhD in 2020, Master degree in 2009, Bachelor degree in 2002 from Visvesvaraya Technological University, India. She is currently working as Faculty of Engineering and Technology, Department of Computer Science and Engineering, Ramaiah University of Applied Sciences, Ramaiah Technology Campus, Bengaluru, Karnataka, India. She has 16 years of experience in teaching and has published many research papers in journals indexed in SCI/SCIE, WOS, SCOPUS and presented papers in several National and International conferences. Her research interest is in the field of Wireless sensor network, IOT, Embedded systems, AI, ML and Deep Learning. She can be contacted at email: jyothiarcotprashant@gmail.com.