

Review of Computer Engineering Research

2024 Vol. 11, No. 1, pp. 1-15

ISSN(e): 2410-9142


ISSN(p): 2412-4281

DOI: 10.18488/76.v11i1.3597


© 2024 Conscientia Beam. All Rights Reserved.



Software reliability prediction using ensemble learning with random hyperparameter optimization

 **Getachew Mekuria Habtemariam¹**

 **Sudhir Kumar Mohapatra²⁺**

 **Hussien Worku Seid³**

^{1,2}Department of Software Engineering, Addis Ababa Science and Technology University, Addis Ababa, Ethiopia.

¹Email: getachewmekuria19@gmail.com

²Email: hussien.seid@aastu.edu.et

²Faculty of Emerging Technologies, Sri Sri University, Cuttack, India.

³Email: sudhir.mohapatra@srisriuniversity.edu.in



(+ Corresponding author)

ABSTRACT

Article History

Received: 18 September 2023

Revised: 22 November 2023

Accepted: 27 December 2023

Published: 10 January 2024

Keywords

Ensemble learning
Hyperparameter
Machine learning
Random hyperparameter
Optimization
Scaling
Software reliability prediction.

The paper investigates software reliability prediction by using ensemble learning with random hyperparameter optimization. Software reliability is a significant problem with software quality that developers face. It involves accurately predicting the next failure. In recent years, machine learning techniques and ensemble learning approaches have been applied to improve software reliability prediction. These approaches aim to analyze historical data and develop models that can accurately forecast when failures are likely to occur. The article proposes an ensemble learning regression model using Ridge, Bayesian Ridge, Support Vector Regressor (SVR), K-Nearest Neighbors Algorithm (KNN), Regression tree, Random Forest, Neural network, and Decision Tree as base learners. Ridge is used as a combiner model. Each base learner hyperparameter is tuned using a random search algorithm automatically. A random hyperparameter search optimization algorithm selects the hyperparameter and adjusts it for overfitting and underfitting. The base models are tuned to minimize bias and variance. The performances of the models are evaluated using standard error measures such as Mean Squared Error (MSE), Sum of Squared Error (SSE), and Normalized Root Mean Square Error (NRMSE). The proposed ensemble model is compared with existing models using a benchmark dataset. The Iyer, and Lee, and Musa datasets are used for the experiment. The dataset is scaled using standard methods like logarithmic scaling, lagging, and linear interpolation. The results of the statistical comparison show better performance by our proposed model as compared to existing models.

Contribution/Originality: This research developed an ensemble learning model to predict software reliability. The use of a random hyperparameter algorithm for automatic parameter selection adds more value to the novel model. This model fine-tuned the machine learning parameters automatically from a given range.

1. INTRODUCTION

The concept of software reliability is defined as the “ability of the software to perform its required function under stated conditions for a stated period of time.” Based on the definition provided by the American National Standards Institute (ANSI), software reliability is “the likelihood of a failure-free process of a software program for a stated period of time in a stated environment” [1]. Determining the probability of a software failure within a specified time frame or the expected time interval between successive failures is the main goal of software reliability. In this work, a model has been developed for predicting and estimating the quantity of defects in the system.

Machine Learning (ML) approaches are found to be more effective in predicting than statistical or classical approaches and could possibly be applied for predicting software reliability. ML is a technique that concentrates on training instantaneously and permits systems to participate and forecast the software behaviour formulated on previous and current failure data as well. Therefore, it is fairly common to recognize which technique happens to be best for an assumed malfunction dataset [2-4].

In this work, an ensemble learning method is applied to predict and evaluate a software model based on its performance. Researchers use various ML algorithms to forecast the time between consecutive software failures. In software reliability prediction, different models are trained on the input data, and the trained model predicts the values based on the test data. The main challenge in the prediction problem is to use different error-measuring techniques to determine the performance of the model. Using a single model for predicting the data may not result in better predictions all the time. So different regression models are used to determine the next failure time, and the error measures of different models are compared. Furthermore, the objective is to propose an ensemble model by combining various models to improve performance over multiple datasets. In this work, several ML methods are applied for reliability prediction, i.e., Random Forest, Decision Tree, Ridge Regression, Bayesian Ridge Regression, K-Neighbours Regression, Lasso Regression, ElasticNet Regression, Artificial Neural Network (ANN), Radial Basis Function Network (RBFN), and Support Vector Regressor (SVR). These machine learning methodologies are employed on different datasets.

To forecast software reliability, a dataset of consecutive failures of the software is used, and the above-mentioned ML methods are applied for the prediction of the failure time of the software reliability dataset. The value is predicted based on the Time Between Successive Failures (TBSF) using a single feature dataset representing the time between successive failures in chronological order. Based on lag length, a data window of various lengths in the dataset is created. Various ML methods and algorithms are used on the lagged dataset to get various statistical error measures and the next failure time by reverse logarithmic scaling. In prediction, a set of hypotheses is combined to generate better accuracy and improved results.

2. LITERATURE SURVEY

In software reliability prediction research, various statistical as well as machine learning models were previously employed by researchers. Researchers used several ML models for determining software reliability. Malhotra, et al. [5]; Li, et al. [6]; Lo [7]; Karunanithi, et al. [8]; Singh and Kumar [9] and Costa, et al. [10]. Ho, et al. [11] conducted a widerange of investigations on connectionist models as well as their relevancy to software reliability prediction and suggested that these were more effective than conventional type models. Su and Huang [12] used an artificial neural network (ANN) in their model. The model predicts the software's reliability with less accuracy. A comparatively new procedure in software reliability prediction development [8] has revealed that ANN models executed a head of parametric models as well as the kinds of network structural design are able to remarkably affect predictive efficiency. In the last few years, few works have been reported on assessing the application of neural network models for software soundness as well as reliability. Time series method is commonly applied for forecasting the upcoming period between consecutive software failures [13]. Overall, ANN balances its input in the range $[0, 1]$ by applying a linear interval, like allocating the values through the highest anticipated value of the correlated input variable. ANN revealed good learning as well as forecasting competence once data points are equally disseminated within an interval of $[0,1]$ [8]. However, in the early stages of testing, the time between successive failures is anticipated to be insignificant. As testing proceeds in advance, software becomes more trustworthy, and the time between successive failures is extended. A predictive capability of ANN for a software failures data has been examined to be improved after the data are evaluated by using an appropriate logarithmic function [14] which evaluates the input data in the interval $[0,1]$. To validate the proposed reliability prediction model, which has applied several competence processes [15] as well to accomplish this implementation,

the important requirement is the selection of independent and dependent variables. The dependent variable that has been applied in this research paper [15] was the failure amount, and the independent variable used was a time interval in a week. Through interrelated failures, testing time in weeks is designated possibly an independent variable. Pai and Hong [16] perform research utilizing support vector machines (SVM) for prognosis software reliability. Aljahdali and Buragga [17] as well as Kumar and Singh [18] stated machine learning methods like cascade correlation neural networks, decision trees, and fuzzy inference systems to predict the soundness of software output [17, 19]. A study by Sabnis, et al. [20] and Ho, et al. [11] compared different technologies and found that ML methods have been used to estimate the defect level of the software. They have utilized various methods like SVM, ANN, Naive Bayes (NB) and RF, where ANN shows good results as compared to others. ANN classifiers have the best accuracy, about 65.5%, among all other ML technologies.

Another study has been done by Jindal and Gupta [21] and Su and Huang [12], in which a heuristics test of different ML and deep learning methods on univariate software failure time stamp data is used to find the best approach for software reliability. ANN has been taken as the baseline model, and it has been found that the Long Short-Term Memory (LSTM) model performs well. Banga, et al. [22] and Bisi and Goyal [13] introduced an approach that is used to find the most relevant parameter affecting software reliability. In this research, a hybrid approach is used to predict the fault of software with the help of machine learning. Vishwanath [14] proposed a method to detect the quality of software with the help of matrices. The information provided by the matrices is important to detect the failure earlier, which is very important in the field of software. In the experiment part, they utilized eight different types of classifiers using metrics, which are collected from the PROMISE data repository.

Yaghoobi [23] and Jaiswal and Malhotra [15] proposed a Software Reliability Growth Model (SRGM). However, the model is for the specific data set. Sudharson and Prabha [24] and Pai and Hong [16] stated that to get dependability in software results by assessing faults during an examination, software reliability was a crucial quantitative attribute. To find product faults, time-dependent software reliability models are used, but they are useless in environments that are constantly changing. Other researchers used individual ML techniques for reliability prediction and fault detection [17, 18]. A detailed literature review was carried out by Habtemariam and Mohapatra [25] and they elaborately discussed the kinds of literature available and their limitations Habtemariam, et al. [26]. Habtemariam and Mohapatra [25] and Getachew, et al. [27] use different machine learning and soft computing techniques in software testing for ensuring the reliability of the software [4, 27-29].

Table 1 summarises the performance achievement of selective models from the literature review.

Table 1. Literature review summary.

Studies	Performance	Dataset	Findings
Sabnis, et al. [20]	65.5% accuracy	NA	ANN classifiers have the best accuracy among all machine-learning technology
Jindal and Gupta [21]	Mean absolute error is 1.5639	Software failures dataset[*]	LSTM model performs well.
Banga, et al. [22]	78% accuracy	NA	A hybrid new approach to fault prediction based on a machine learning algorithm.
Yaghoobi [23]	85% accuracy	NA	Statistical models are used.
Sudharson and Prabha [24]	85% accuracy	NA	Soft computing methods are used.
Reddivari and Raman [30]	AUC of 0.75	UIMS and QUES [**]	Decision tree-based prediction techniques perform well.

Note: User Interface Management System dataset (UIMS), Quality Evaluation System dataset (QUES), Area under curve (AUC).

* is used in place of exactly *one* qualifier.

** indicates either the nonexistence of leading, trailing, or middle qualifiers or the fact that they play no role in the selection process.

The following are the two major motivations that were inferred from our literature review:

1. It has been proven that ML-based prediction is better than that of statistical technique'. Statistical techniques are better for correlating variables. ML, on the other hand, is good at prediction.
2. In the case of machine learning, combination or ensembles are proven to be more effective than a single base model. The prediction accuracy and performance of ensemble models are much higher than those of a single model.

2.1. Dataset

In this section, we describe the two benchmark datasets that have been used in this research work. Several conventional printed works have been published on these datasets, and numerical fault calculations are also accessible. The time series datasets used for this model are Iyer and Lee's (1996) software reliability dataset and Musa, J.D: software reliability data¹. These datasets are basically a time of failure for the software. The pattern of the dataset is depicted in Figure 1.

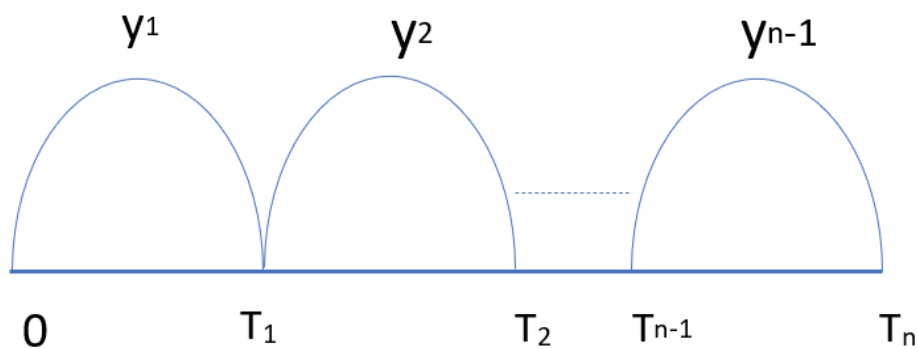


Figure 1. Software failure process.

Figure 1 depicts a typical software failure procedure, where T_i is the period instant of the i^{th} failure. The datasets provide the intervals between each of the total failures that are documented. The dataset's individual inspection is indicated by the notation (i, y_i) , where i represents the failure number and y_i denotes the interval between the $(i - 1)^{\text{th}}$ and i^{th} failures. The formula 1 that was produced due to the software malfunction is shown below.

$$y_i = T_i - T_{i-1} \quad (1)$$

Dataset was collected from Bhuyan, et al. [31] in March 2023.

2.2. Proposed Model

The proposed model of software reliability prediction involves learning from a training algorithm and then integrating the predictions of various learning algorithms. At first, individual algorithms learn by utilizing the accessible data; subsequently, assembler algorithms learn to create the last prediction by employing every prediction of the distinct algorithms as further input. When a random assembler algorithm is applied, the model perhaps hypothetically denotes all the ensemble learning methods. The proposed model is exhibited in Figure 2.

2.2.1. Random Search Optimization Algorithm

The working principle of the random search algorithm is very simple. Firstly, it searches from the list of possible values of the hyperparameter. This tuning of the hyperparameter continues with several random searches. It stops when the desired result is achieved. One of the drawbacks of this algorithm is its computational complexity. The hyperparameter value and range of the individual models are represented in Table 2.

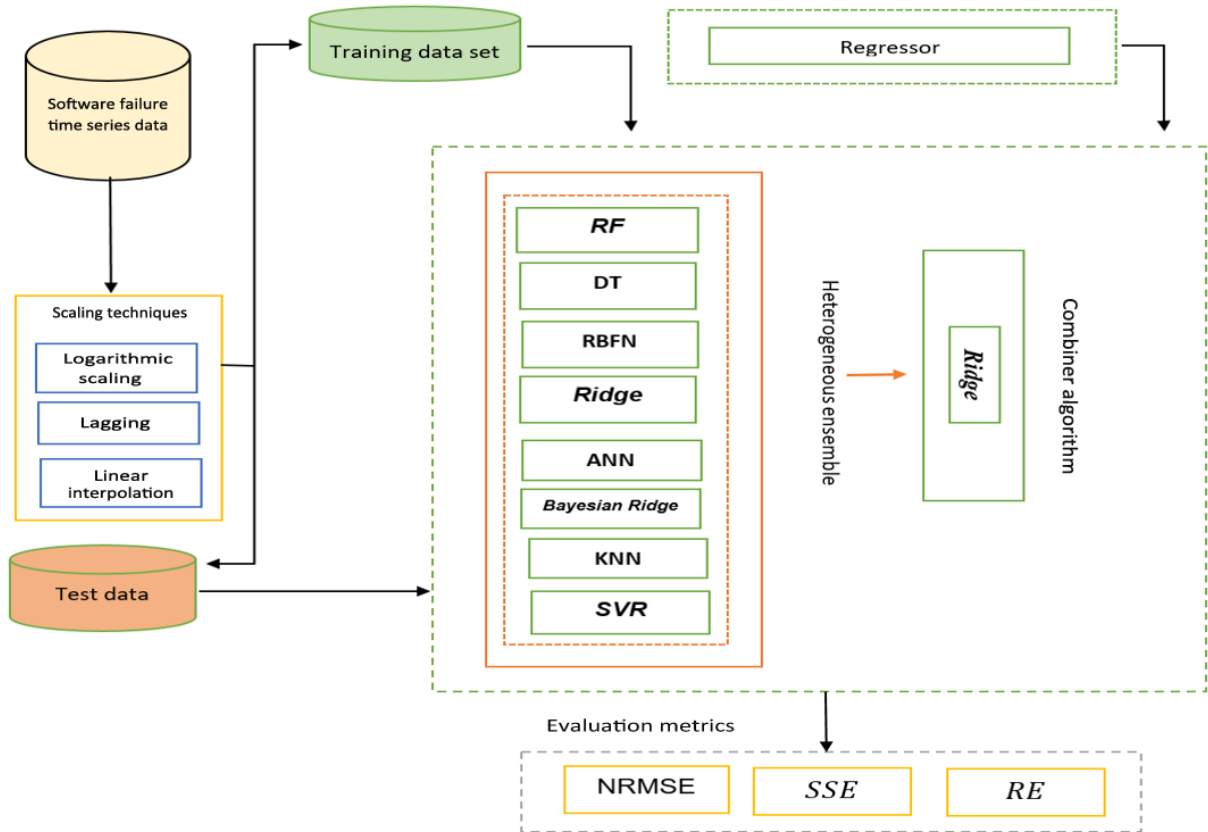


Figure 2. The proposed ensemble model with automatic hyperparameter selection.

Note: Random forest (RF), decision tree, radial basis function network (RBFN), Ridge, artificial neural network (ANN), Bayesian ridge regression, K-Neighbours regression, Support vector regressor (SVR).

Table 2. Hyperparameter of individual models.

Base model	Values(In range)	Hyperparameter
Ridge	$10^{\text{range}(-5, 0)}$	Alpha
Bayesian ridge	$10^{\text{range}(-5, 0)}$	alpha_1
	$10^{\text{range}(-5, 0)}$	alpha_2
Regression tree	Range(4, 23)	max_depth
SVR	Linspace(0.01, 5, 20)	C
	Range(0.01, 0.5, 0.05)	Gamma
	{Linear, poly, rbf}	Kernel
KNN	Range(2, 11)	n_neighbors
Random forest	{100, 200, 500}	n_estimators
	Range(4, 10)	max_depth
Neural network	Linspace(0.0001, 0.5, 20)	Alpha
	Linspace(0.0001, 0.5, 20)	Learning_rate_init
	{Identity, logistic, tanh, relu}	Activation
Decision tree	Range(4, 51)	Max_depth
	Range(4, 30)	Max leaf nodes

Figure 2 shows how various based learner algorithms are integrated to provide the result with the best precision. The designed model typically yields superior performance than every individual trained model. It has been efficiently utilized in supervised learning methods as well as in unsupervised learning methods. The proposed algorithm for our model is given below:

Algorithm 1 proposed model

Input: D = Dataset

Method:

1. *BaseLearnersModel* = [Bayesian Ridge, SVR, ANN, KNN, DT.....]
 2. *CombinerModel* = [Ridge]
 3. $x_trainD, x_testD, y_trainD, y_testD = split(D)$
 4. *PModel* = []
 5. For function in *BaseLearnersModel*:
 6. *PModel.append* (function. *train* (x_trainD, y_trainD))
 7. *Train_setD* = []
 8. For i in $range(length(x_trainD))$:
 9. *PTrain* = []
 10. For m in *PModel*:
 11. *PTrain.append* ($m.predict(x_trainD[i])$)
 12. *CombinerModel.train* (*Train_setD, y_trainD*)
- *Prediction:*
1. *Train_setD_x* = []
 2. For m in *pmodel*:
 3. *Train_setD_x.append* ($m.predict(x)$)
 4. *Output* = *CombinerModel.predict* (*Train_setD_x*)
- *Output: Array of Individual model Accuracy and Stacking accuracy*

Note: Decision tree (DT).

The above ensemble algorithm that we have applied in our work is intended to enhance the consistency of software by integrating diverse other algorithms.

2.2.2. Machine Learning and Scaling Techniques

Various base models are used, and the Ridge model is used as the combiner model. For scaling and pre-processing the data, the logarithmic scaling approach is used.

2.2.3. Logarithmic Scaling

Linear scaling of data points decreases the correlation between the points and, hence, the performance of the evaluation. So, to enhance the performance of the model, the logarithmic scaling technique is utilized between successive failures, as given by Equation 2.

$$y^* = \ln(1 + By) \quad (2)$$

In which, y^* is the measured value of time among consecutive failures, y is the period among consecutive failures and B is the measuring constant which is given in Equation 3.

$$B = \frac{e^{y^*} - 1}{y_{max}} \quad (3)$$

After prediction, reverse scaling is performed to get the real-time among consecutive failures and is shown in Equation 4.

$$y = \frac{e^{y^*} - 1}{B} \quad (4)$$

Where y^* is the imitation result of prediction and B is the measuring constant.

2.2.4. Lagging

The benchmark datasets that have been used for this proposed work are single-featured datasets. So, for a single instance of the prediction of error, the Lag method is used to generate a window of values by combining the set of values for the machine learning prediction problem. The lagging algorithm in this work is as follows:

Algorithm 2 lagging

- *Input: Dataset, lag sequence (k, e.g.: k=3 or 4 or 5 etc)*
 - *➤ Algorithm:*
 1. $L = \text{dataset length}$
 2. *for i in 0 to L-k+1:*
 3. *Store data from i to i + k index of the dataset in the list m*
 4. *return m*
 - *Output: lagged value array*
-

The lag length in this proposed work varies from 2 to 25, depending on the model used for prediction. After lagging, the single-featured dataset is segmented into different lag sequences according to the defined lag length.

2.2.5. Linear Interpolation

Linear interpolation is a technique for building virtual data points in a dataset. However, in the case of a small dataset quantity, it is difficult to acquire the best consistency result after the train and split test. Therefore, to augment the soundness of the model, these methods are applied to optimize the level of the training data. After optimizing the existing dataset level, it is more efficient in the training dataset and model assessment. As the benchmark dataset used in this work is distinct and features failure time data, linear interpolation is simple to use. The algorithm of linear interpolation is exhibited as follows:

Algorithm 3 linear interpolation

- *Input: D=Dataset*
 - *➤ Algorithm:*
 1. $t = \text{train_set}(D)$
 2. $\text{dataset} = []$
 3. $\text{dataset.append}(\text{train}[0])$
 4. *for index in range (1, length(t))*
 5. $\text{dataset.append}((t[\text{index}-1]+t[\text{index}])/2)$
 6. $\text{dataset.append}(t[\text{index}])$
 - *Output: Linearly interpolated*
-

This algorithm helps to create virtual point in the train data by using the linear interpolation method.

2.2.6. Error Measures

Various statistical error measures are used in this work, to evaluate the regression models and also to compare the performance of the benchmark dataset. They are listed below.

- **NRMSE:** Normalized root mean squared error
The Normalized Root Mean Square Error (NRMSE) is used to facilitate the comparison between models with different scales. The Normalized RMSE (NRMSE) relates the RMSE to the observed

range of the variable. Therefore, it is possible to interpret the NRMSE as the portion of overall range that the model typically resolves.

$$NRMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - y'_i)^2}{\sum_{i=1}^n y'_i}} \quad (5)$$

Where n is the number of data points used to train the model, y_i is the actual value, and y'_i is the predicted value obtained from the algorithm for the i th data point in the software failure dataset. NRMSE is used to minimize the error generated during the learning of the model with the given algorithm.

- SSE: Sum of squared error

Sum Squared Error (SSE) is an accuracy measure where the errors are squared and then added. It is used to determine the accuracy of the prediction model when the data points are similar in magnitude. The lower the SSE, the more accurate the prediction. The sum of squared errors is usually used as a criterion for the comparison of goodness-of-fit and predictive power. SSE can be calculated as follows:

$$SSE = \sum_{i=1}^n (y_i - y'_i)^2 \quad (6)$$

Where:

$\{y_i\}$ is the actual observations time series value and $\{y'_i\}$ is the estimated or forecasted time series value

- RE: Relative error

The relative error is defined as the ratio of the absolute error of the measurement to the actual measurement. Using this method, we can determine the magnitude of the absolute error in terms of the actual size of the measurement. If the true measurement of the object is not known, then the relative error can be found using the measured value. The relative error indicates how good the measurement is relative to the size of the object being measured. If y_i is the actual value of a quantity, y'_i is the measured value of the quantity is the absolute error, then the relative error can be measured using the below formula.

$$RE = \frac{|y'_i - y_i|}{y_i} * 100 \quad (7)$$

An important note is that relative errors are dimensionless. When writing relative errors, it is usual to multiply the fractional error by 100 and express it as a percentage.

Overall, n represents the number of prediction data points; y_i represents the real-time among failures and y'_i represents the predictable time among i^{th} and $(i-1)^{\text{th}}$ failures data. These are standard error measurements which are utilized to evaluate the performance of the model. The flow of the process of the model is presented in [Figure 3](#).

3. RESULTS

The time between consecutive failures is being experimented with using several algorithms, and the reliability is assessed by statistical evaluation methods. The gliding window size is set to 2-25 for better observation. In addition, we discovered that by increasing the distance of the sliding window, the size of the training set is reduced. Thus, the establishment of virtual data points appeared as a way of growing the dataset. During a time series, data acquired positive and negative maximum points; linear interpolation was used to sort out the training dataset. Throughout the investigation, a few models improved their capability by interpolating the outcome since the relationship among data points was developed and the variance among them dwindled. But, for a few models like Ridge and Bayesian Ridge regressors, there is a reverse effect. Concurrently, the unfairness of the non-CART (Classification and Regression Trees) models increases as we downturn the variance through the unfair variance trade-off; the achievement of the model downturn is notable. The model is implemented using Python 3.6 with keras, Tensorflow, Pandas, Sklearn, and other required packages. Model execution is done on a hardware platform

with an Intel (R) 11th Gen Intel (R) Core (TM) i7-1165G7 @ 2.80GHz 2.80 GHz machine with 16.0 GB of memory capacity. Python is installed in the Windows 11 Home single-user operating system with a visual code editor.

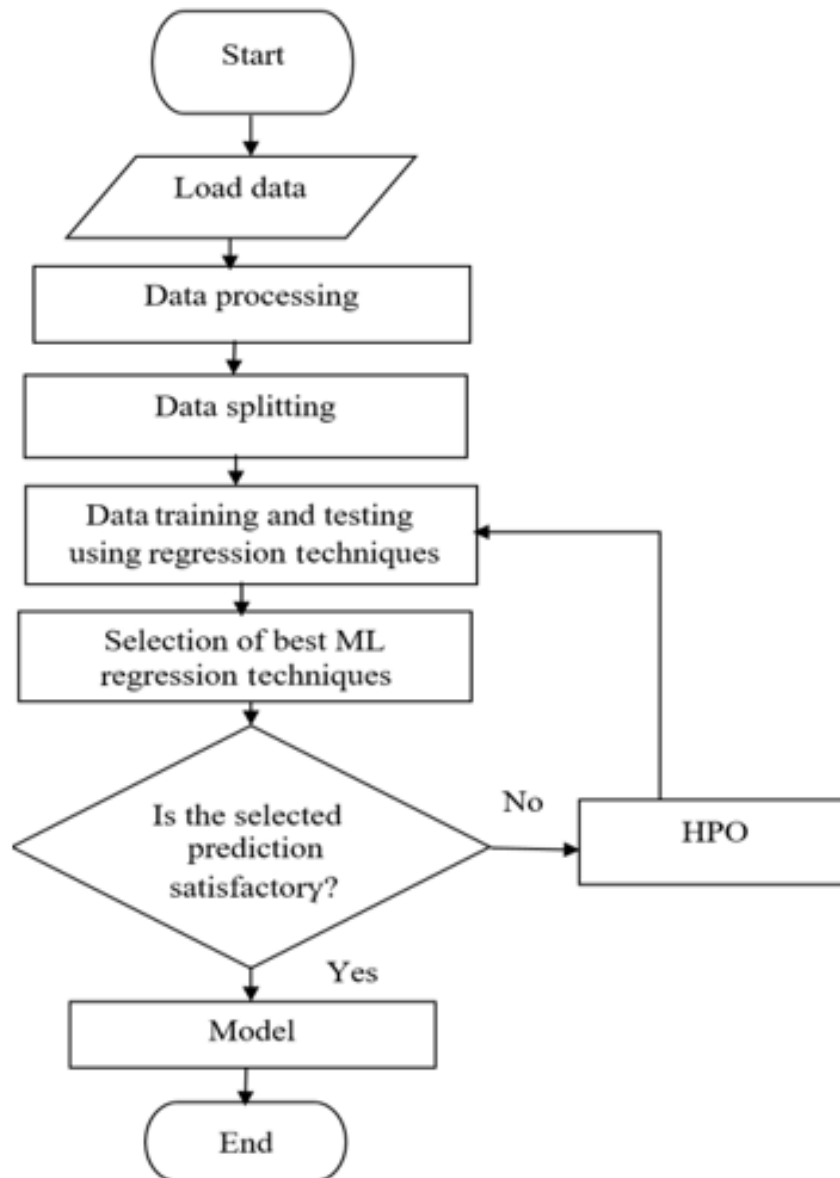


Figure 3. The flowchart of the process.

3.1. Musa Dataset Results

The visualization of NRMSE values for all the base models on the Musa dataset is discussed in this section. Ridge, Bayesian Ridge, SVR, and Random Forest performance is better. Other regressors, like ElasticNet as well as Lasso regression, have poor results. The introduction of the interpolation technique makes KNN and ANN achieve better results. Figure 4 exhibits the NRMSE performances of the algorithms used in our experiment.

The introduction of linear interpolation of the train data set creates a virtual dependency between NRMSE values and the lag distance of datasets for the regressors. Linear interpolation doesn't always increase efficiency; sometimes it reduces the efficiency of the model.

3.2. Iyer Lee Dataset Results

Next, the result of the Iyer and Lee dataset is elaborately discussed in this section. The visual representation is in two categories, i.e., with interpolation and without interpolation. Regressors RBFN, ANN, and SVR achievement

is better, whereas ElasticNet and Lasso's results are not satisfactory. In interpolated values KNN, ANN, random forest, and decision tree performance is good. Figure 5 represents the NRMSE performances of the algorithms used in our experiment.

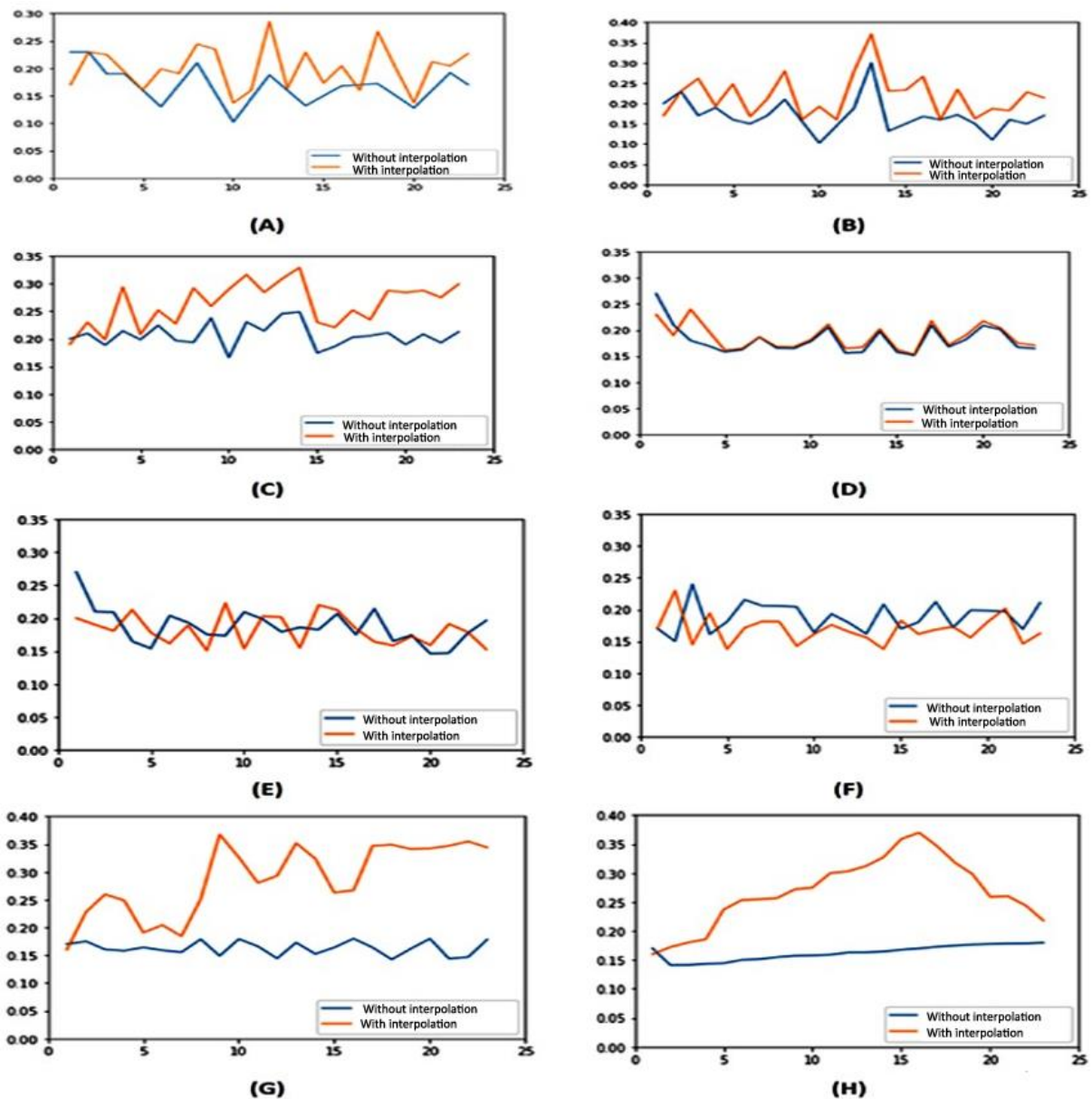


Figure 4. Base model's NRMSE performance for varied lag length.

The results show that the impact of interpolation has a negative impact on NRMSME for some of the regressor links in SVR. An increase in lag length converges on the result.

3.3. Overall Performance

The comparison of the NRMSE values of all the regressed models with a variation in lag length is shown in Figures 6 and 7 for the Musa dataset and Iyer and Lee dataset, respectively.

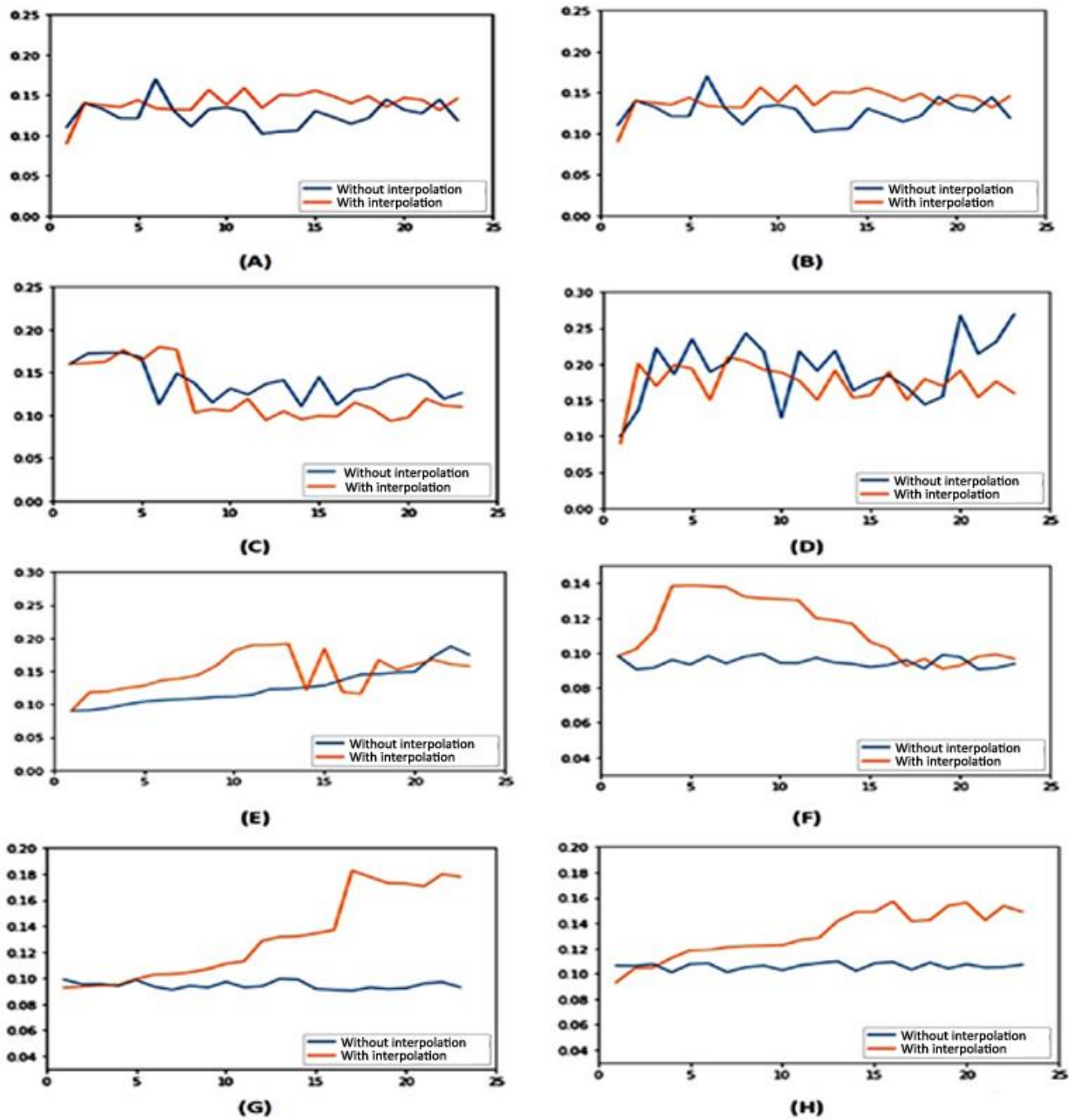


Figure 5. Base model's NRMSE performance for varied lag length.

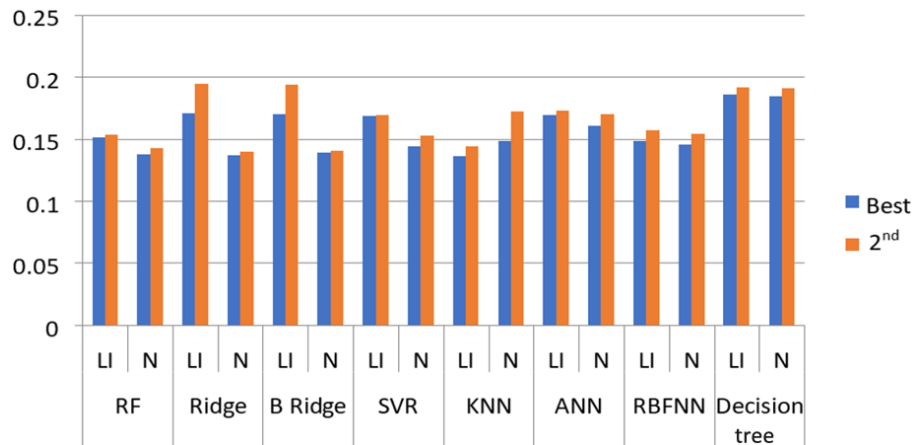


Figure 6. Top two individual models (Musa data).

For the Musa dataset, RandomForest, Ridge, and Bayesian Ridge are the leading performers. In interpolated data, KNN is the highest performer.

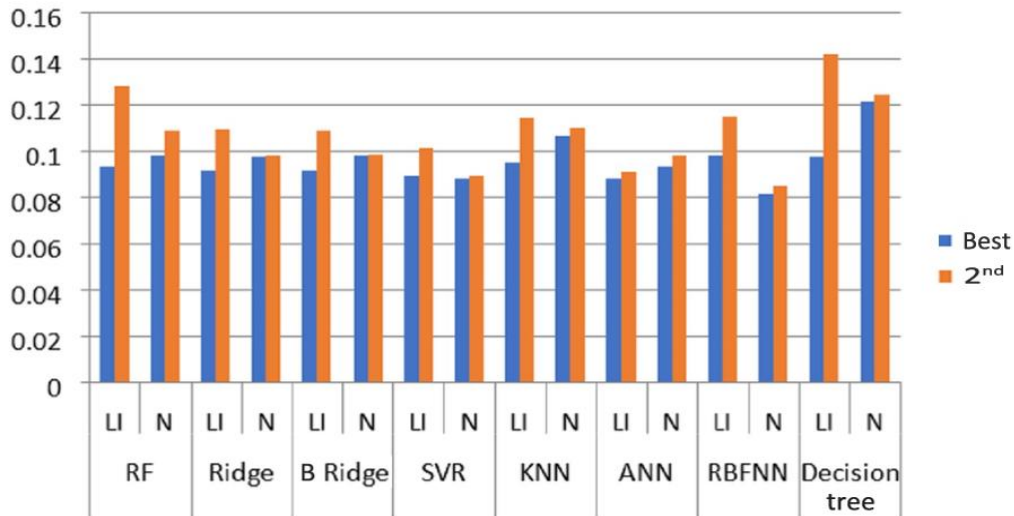


Figure 7. Top two individual models (Iyer Lee Data).

In the Iyer and Lee data set, the best performers are RBFN and SVR on the real dataset. Except for DT, others' performance is average. The worst performer is DT. NRMSE performance of all the methods over both datasets is tabulated in Table 3.

Table 3. The NRMSE performance of different benchmark models.

Lag length		2	3	4	5	6	7
ANN	Musa	0.159	0.142	0.053	0.153	0.163	0.168
	Iyer Lee	0.513	0.494	0.541	0.532	0.564	0.587
ANN PSO	Musa	0.169	0.163	0.145	0.128	0.153	0.178
	Iyer Lee	0.391	0.377	0.325	0.310	0.347	0.398
Ridge	Musa	0.1769	0.171	0.160	0.156	0.151	0.143
	Iyer Lee	0.092	0.109	0.124	0.141	0.147	0.155

Table 4. Comparison with existing models.

Statistical models	Jelinski Moranda	Geometric	Musa basic	Musa Okumoto
Musa	0.144	0.142	0.141	0.141
Iyer Lee	0.342	0.538	0.211	0.326
ML models	ANN	ANN-PSO	KNN	Ridge
Musa	0.142	0.128	0.136	0.157
Iyer Lee	0.494	0.310	0.095	0.492

Note: Artificial neural network-particle swarm optimization (ANN-PSO).

The proposed model is compared with other existing statistical and machine-learning models. The comparison is represented in Table 4. The proposed model's performance is better than that of other existing techniques. In some cases, the ANN-PSO model performs on par with our model.

4. CONCLUSION

The proposed ensemble learning model consists of many weak regressors. Experimental results reveal that Ridge and Bayesian Ridge regressors produce excellent performance for both datasets. Certain models perform better on non-interpolated data, whereas other models are executed on interpolated data. With the presence of

interpolating points, the performance of KNN and ANN is remarkable. Because of scaling, the overall performance of the model has improved. This is because the time-series data are unevenly distributed, and the presence of a pick is found at certain points. The lag length is also importance for the model's performance. RBFN, Ridge, and Bayesian Ridge bring about better results when the lag length is between 7 and 11. ANN and RF work more accurately on the actual dataset with a lag length greater than 15. In interpolated data, KNN and SVR perform well for certain lag lengths.

Funding: This study received no specific financial support.

Institutional Review Board Statement: Not applicable.

Transparency: The authors state that the manuscript is honest, truthful, and transparent, that no key aspects of the investigation have been omitted, and that any differences from the study as planned have been clarified. This study followed all writing ethics.

Competing Interests: The authors declare that they have no competing interests.

Authors' Contributions: Conceptualization, G.M.H. and S.K.M.; methodology, S.K.M. and S.P.; formal analysis, G.M.H., B.T. and S.K.M.; investigation, G.M.H. and B.T.; resources, S.K.M., P.S. and H.W.S.; data curation, P.S. and K.S.C.; writing, G.M.H. and S.K.M.; writing, review and editing, K.S.C., B.T.; visualization, G.M.H. and H.S.W.; supervision, S.K.M. and S.P.; funding acquisition, S.K.M. All authors have read and agreed to the published version of the manuscript.

REFERENCES

- [1] A. Quyoum, M.-U.-D. Dar, and S. Quadri, "Improving software reliability using software engineering approach-A review," *International Journal of Computer Applications*, vol. 10, no. 5, pp. 41-47, 2010. <https://doi.org/10.5120/1474-1990>
- [2] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Investigating the effect of coupling metrics on fault proneness in object-oriented systems," *Software Quality Professional*, vol. 8, no. 4, p. 4, 2006. <https://doi.org/10.5381/jot.2007.6.10.a5>
- [3] B. Goel and Y. Singh, "An empirical analysis of metrics to predict the maintainability for real-time object-oriented software," *Software Quality Professional*, vol. 11, no. 3, pp. 35-45, 2009.
- [4] S. Ramalingam, "Study and review of classical, machine learning and deep learning methods of software reliability estimation for safety-critical systems," *Indian Journal of Natural Sciences*, vol. 13, no. 76, pp. 1-16, 2023.
- [5] R. Malhotra, A. Kaur, and Y. Singh, "Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines," *International Journal of System Assurance Engineering and Management*, vol. 1, pp. 269-281, 2010. <https://doi.org/10.1007/s13198-011-0048-7>
- [6] X. Li, X. Li, and Y. Shu, "An early prediction method of software reliability based on support vector machine," presented at the 2007 International Conference on Wireless Communications, Networking and Mobile Computing, 2007.
- [7] J.-H. Lo, "Predicting software reliability with support vector machines," presented at the 2010 Second International Conference on Computer Research and Development, 2010.
- [8] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Prediction of software reliability using connectionist models," *IEEE Transactions on Software Engineering*, vol. 18, no. 7, pp. 563-574, 1992. <https://doi.org/10.1109/32.148475>
- [9] Y. Singh and P. Kumar, "Prediction of software reliability using feed forward neural networks," presented at the 2010 International Conference on Computational Intelligence and Software Engineering, 2010.
- [10] E. O. Costa, A. T. R. Pozo, and S. R. Vergilio, "A genetic programming approach for software reliability modeling," *IEEE Transactions on Reliability*, vol. 59, no. 1, pp. 222-230, 2010. <https://doi.org/10.1109/tr.2010.2040759>
- [11] S. L. Ho, M. Xie, and T. Goh, "A study of the connectionist models for software reliability prediction," *Computers & Mathematics with Applications*, vol. 46, no. 7, pp. 1037-1045, 2003. [https://doi.org/10.1016/s0898-1221\(03\)90117-9](https://doi.org/10.1016/s0898-1221(03)90117-9)
- [12] Y.-S. Su and C.-Y. Huang, "Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," *Journal of Systems and Software*, vol. 80, no. 4, pp. 606-615, 2007. <https://doi.org/10.1016/j.jss.2006.06.017>

- [13] M. Bisi and N. K. Goyal, *Artificial neural network applications for software reliability prediction*. Hoboken, N.J: John Wiley & Sons, 2017.
- [14] S. P. Vishwanath, "Software reliability prediction using neural networks," Doctoral Dissertation, IIT Kharagpur, 2006.
- [15] A. Jaiswal and R. Malhotra, "Software reliability prediction using machine learning techniques," *International Journal of System Assurance Engineering and Management*, vol. 9, pp. 230-244, 2018. <https://doi.org/10.1007/s13198-016-0543-y>
- [16] P.-F. Pai and W.-C. Hong, "Software reliability forecasting by support vector machines with simulated annealing algorithms," *Journal of Systems and Software*, vol. 79, no. 6, pp. 747-755, 2006. <https://doi.org/10.1016/j.jss.2005.02.025>
- [17] S. H. Aljahdali and K. A. Buragga, "Employing four ANNs paradigms for software reliability prediction: An analytical study," *ICGST International Journal on Artificial Intelligence and Machine Learning*, vol. 8, no. 2, pp. 1-8, 2008.
- [18] P. Kumar and Y. Singh, "An empirical study of software reliability prediction using machine learning techniques," *International Journal of System Assurance Engineering and Management*, vol. 3, no. 3, pp. 194-208, 2012. <https://doi.org/10.1007/s13198-012-0123-8>
- [19] J. D. Musa, A. Iannino, and K. Okumoto, "Software reliability," *Advances in Computers*, vol. 30, pp. 85-170, 1990. [https://doi.org/10.1016/S0065-2458\(08\)60299-5](https://doi.org/10.1016/S0065-2458(08)60299-5)
- [20] P. S. Sabnis, S. Joshi, and J. Naveenkumar, "A study on machine learning techniques based software reliability assessment," in *2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2022: IEEE, pp. 687-692.
- [21] A. Jindal and A. Gupta, "Comparative analysis of software reliability prediction using machine learning and deep learning," presented at the 2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS), 2022.
- [22] M. Banga, A. Bansal, and A. Singh, "Implementation of machine learning techniques in software reliability: A framework," presented at the 2019 International Conference on Automation, Computational and Technology Management (ICACTM), 2019.
- [23] T. Yaghoobi, "Selection of optimal software reliability growth model using a diversity index," *Soft Computing*, vol. 25, no. 7, pp. 5339-5353, 2021. <https://doi.org/10.1007/s00500-020-05532-0>
- [24] D. Sudharson and D. Prabha, "A novel machine learning approach for software reliability growth modelling with pareto distribution function," *Soft Computing*, vol. 23, no. 18, pp. 8379-8387, 2019. <https://doi.org/10.1007/s00500-019-04047-7>
- [25] G. M. Habtemariam and S. K. Mohapatra, "A genetic algorithm-based approach for test case prioritization. In: Mekuria, F., Nigussie, E., Tegegne, T. (eds) Information and Communication Technology for Development for Africa. ICT4DA 2019. Communications in Computer and Information Science," vol. 1026. Cham: Springer. https://doi.org/10.1007/978-3-030-26630-1_3, 2019.
- [26] G. M. Habtemariam, S. K. Mohapatra, H. W. Seid, and D. B. Mishra, *A systematic literature review of predicting software reliability using machine learning techniques. In: Khari, M., Mishra, D.B., Acharya, B., Gonzalez Crespo, R. (Eds.), Optimization of Automated Software Testing Using Meta-Heuristic Techniques*. Cham: EAI/Springer Innovations in Communication and Computing. Springer, 2022, pp. 77-90.
- [27] D. Getachew, S. K. Mohapatra, and S. Mohanty, *A heuristic-based test case prioritization algorithm using static metrics. In: Khari, M., Mishra, D.B., Acharya, B., Gonzalez Crespo, R. (eds) Optimization of Automated Software Testing Using Meta-Heuristic Techniques. EAI/Springer Innovations in Communication and Computing*. Cham: Springer. https://doi.org/10.1007/978-3-031-07297-0_4, 2022.
- [28] S. K. Mohapatra, A. K. Mishra, and S. Prasad, "Intelligent local search for test case minimization," *Journal of The Institution of Engineers (India): Series B*, vol. 101, pp. 585-595, 2020. <https://doi.org/10.1007/s40031-020-00480-7>

- [29] G. M. Habtemariam, S. K. Mohapatra, and H. W. Seid, "Prediction of software reliability using particle swarm optimization," presented at the International Conference on Innovations in Intelligent Computing and Communications, 2022.
- [30] S. Reddivari and J. Raman, "Software quality prediction: An investigation based on machine learning," in *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, 2019: IEEE, pp. 115-122.
- [31] M. K. Bhuyan, D. P. Mohapatra, and S. Sethi, "Software reliability prediction using fuzzy min-max algorithm and recurrent neural network approach," *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 6, no. 4, pp. 1929-1938, 2016.

Views and opinions expressed in this article are the views and opinions of the author(s), Review of Computer Engineering Research shall not be responsible or answerable for any loss, damage or liability etc. caused in relation to/arising out of the use of the content.