Journal of Information Systems and Telecommunication
Vol.11, No.3, July-September 2023, 222-231
http://jist.acecr.org
ISSN 2322-1437 / EISSN:2345-2773

Long-Term Software Fault Prediction Model with Linear Regression and Data Transformation

Momotaz Begum[1], Jahid Hasan Rony[1], Md. Rashedul Islam[1,2], Jia Uddin[3*]

[1].Department of Computer Science and Engineering, Dhaka University of Engineering & Technology, Gazipur-1707, Dhaka, Bangladesh;
[2].Department of Computer Science and Engineering, International University of Business Agriculture and Technology
[3].AI and Big Data Department, Endicott College, Woosong University, Daejeon, South Korea

Received: 12 Apr 2022/ Revised: 04 Sep 2022/ Accepted: 27 Oct 2022

Abstract

The validation performance is obligatory to ensure the software reliability by determining the characteristics of an implemented software system. To ensure the reliability of software, not only detecting and solving occurred faults but also predicting the future fault is required. It is performed before any actual testing phase initiates. As a result, various works on software fault prediction have been done. In this paper presents, we present a software fault prediction model where different data transformation methods are applied with Poisson fault count data. For data pre-processing from Poisson data to Gaussian data, Box-Cox power transformation (Box-Cox_T), Yeo-Johnson power transformation (Yeo-Johnson_T), and Anscombe transformation (Anscombe_T) are used here. And then, to predict long-term software fault prediction, linear regression is applied. Linear regression shows the linear relationship between the dependent and independent variable correspondingly relative error and testing days. For synthesis analysis, three real software fault count datasets are used, where we compare the proposed approach with Naïve gauss, exponential smoothing time series forecasting model, and conventional method software reliability growth models (SRGMs) in terms of data transformation (With_T) and non-data transformation (Non_T). Our datasets contain days and cumulative software faults represented in (62, 133), (181, 225), and (114, 189) formats, respectively. Box-Cox power transformation with linear regression (L_Box-Cox_T) method, has outperformed all other methods with regard to average relative error from the short to long term.

Keywords: Software Reliability; Software Faults; Forecasting; Long Term Prediction; Relative Error.

1- Introduction

In the modern era, the software acts as an essential part of our life. The software ensures the performance of our digital devices and helps to maintain our lifestyle, manage businesses, and so on. It has become impossible to pass even a single day without software usage in our daily life. When software is responsible for a massive operation, making a minor software fault, the entire system can collapse. For example, in 2018, a fully autonomous uber test car hit a pedestrian and accidentally killed her [1]. Because of the object detection software fault, the system failed to detect the human who was crossing the road with her bike. In addition, the Hawaii missile false alarm is another example of major suffering due to software failure [2]. Such incidents could have been avoided by reliable software with a software failure prediction system which is a popular approach in software engineering.

The engineering approach of systematic application development can be defined as the term, software engineering. The test effort, optimal cost analysis, correction prediction, security, effort, reusability, and quality-related prediction are a few vital parts issues of software engineering. To find a versatile method of prediction analysis further research is still going on in this area. On the other hand, to ensure software reliability, software fault prediction performance has to make sure. Software reliability is the probability of failure-free software. Long-term software failure can detect the possibility of a software failure so that impact of the failure can be minimized by taking necessary steps and precautions [3].

The development of software is expected to be perfect. However, it is impracticable to design and develop

✉ **Jia Uddin**
jia.uddin@wsu.ac.kr

software with 100% accuracy and dependability. From a previous study, already it has been established that proficiency of fault prediction is caused due to the lack of proper evaluation criteria of performance and different fault distribution in a fault dataset [4]. But day by day the importance of software fault prediction has gained lot of attention because of the capability of providing faults number as well as the occurrence pattern of a certain system. Subsequently, it is also helpful for the quality assurance team as it can reduce testing time and cost.

The purpose of the software fault prediction is to identify the fault before sending it to the testing phase in the basis of software structural characteristics. In addition, to ensure software quality, professional stakeholders use prediction systems for optimal cost and effort during the operational phases. In this regard, we focused on the long-term software fault prediction using a linear regression method. Besides that, we have compared the model with Naïve gauss, exponential smoothing time series forecasting model, and two existing software reliability growth models name log extreme minimum (SRGM_LEM) [5] and pareto (SRGM_Pareto) [6]. Besides that, we have used software fault count data instead of fault detection time data because of the availability and the usefulness. Furthermore, three real data sets have been used for this study and the three most popular data transformations Box-Cox_T, Yeo-Johnson_T as well as Anscombe_T methods have been applied for the Poisson data into Gaussian data.

The organization of the rest of the paper is as follows: the related study is explained with conventional NHPP-based SRGMs in Section 2. Then in Section 3, the system architecture is described with suitable figures associated with data pre-processing techniques and forecasting models. The fault prediction with the proposed methodology is presented in Section 4. After that, Section 5 represents the experimental illustration assist with system setup, performance measurement, and result analysis. Finally, the paper concludes with future direction in Section 6.

2- Related Works

Various works have been done and are still going on in this field to predict faults of software to ensure reliability. Software reliability growth models (SRGMs) are one of the oldest with some limitations, such as the maximum likelihood estimation requires high computation power, and from a large number of SRGMs, researchers get confused to select the suitable model for every software data [3]. Nowadays, another popular classification method is an artificial neural network (ANN) with backpropagation (BP) learning algorithms used in software fault prediction [5], [67], [8].

Recently, Begum et al. proposed a robust prediction interval method using a refined artificial intelligence approach, where 5 data transformation methods are used for pre-processing and compared with the traditional method SRGMs [9]. They have constructed prediction intervals using their proposed method, and performance analysis is conducted by coverage rate and mean prediction interval width as well as compared with the existing delta method. However, the architecture of neural networks is complex; as a result, computation time is very high. The same author has related works [10–13] based on multilayer perceptron to address optimal software release problems.

Furthermore, the paper [14] presented a neuro base software fault prediction method using the Box-Cox transformation scheme. They have also investigated the optimal value of transformation parameter λ in case of average relative errors. Subsequently, they compared their result with traditional SRGMs and showed that their method outperformed in the early testing phase. On the other hand, multiplayer neural network architecture is used for identifying optimal software testing time [15]. For underlying software fault count, they have also pre-processed the data using a well-known data transformation technique. Where experimental result was conducted from four (4) actual software fault count data.

In [16] a study showed a study about software fault prediction and the different components and parameters of software fault prediction. Then the paper focused on the accomplishment in this area as well as recent research trends. In addition, they have discussed major future challenges of software fault prediction. But the advantage and disadvantage of recent studies has not prevailed. Different software fault prediction technique [17]–[20] have been proposed previously but none of those fully fills the long-term software fault prediction criteria as well as could not provide enough quality assurance resources and logistics.

Recently semantic long short-term memory (LSTM) network is used to train a model that can be self-directed to identify fault prediction and performed on real projects where the proposed model outperformed state-of-the-art approaches of fault prediction [21], [22]. After that, for finding fault in real life, a deep learning-based fault prediction model is presented in [23]. Which matches the abstract of source code syntax tree representation based on tree structure LSTM. They have evaluated their model using Samsung and a public PROMISE repo dataset. Then in a study [34], author estimate the reliability improvement that the recently suggested SDAFlex&Rel software development methodology, which aims to create reliable but flexible software, promises. By laying the groundwork for formal modelling, refinement, and verification which in turn avoid and eliminate possible faults, that method increases the dependability of software.

On the other hand, Deepak and Pravin introduced object-oriented metrics that are used for main factor findings [24]. In, Diego J. Pedregal [25] presents a paper where a few time series forecasting methods such as regression, Naïve methods, ARIMA, Transfer Functions, VAR(X), exponential smoothing (ETS), unobserved components (UC) models are used to develop a user-friendly graphical interface tool based on MATLAB for automatic outlier identification and detection. Also, a regression technique is applied for prediction analysis which provides a different approach for the software prediction process. For example, the linear regression method is used in optical network fault tracing to reduce high cost and fault detection time cited in [26].

In this paper, we presented different transformation methods as pre-processing for the real dataset and applied linear regression, Naïve Gauss, and exponential smoothing time series forecasting methods to predict software faults. Finally compared the outcome from various perspectives with existing popular SRGMs.

2-1- Non-Homogeneous Poisson Process-based SRGM

Let the group number of software fault represented by $G(t)$ at time t. Suppose (i) $G(0) = 0$, (ii) $G(t)$ has independent increments, (iii) $Pro[G(t+q)- G(t) \geq 2] = O(q)$, (iv) $Pro[G(t+q)- G(t) =1] = \lambda(t; \Theta)q+O(q)$, Here, non-homogeneous Poisson process(NHPP) intensity function is $\lambda(t; \Theta)$, Θ is the parameter of the model and infinitesimal time q higher term is $O(q)$. So that, probability of $G(t) = y$ is calculated by,

$$Pro[G(t) = y] = \frac{\Lambda(t;\Theta)^y}{y!} * \exp[-\Lambda(t;\Theta)] \quad (1)$$

where mean value function is,

$$\Lambda(t;\Theta) = \int_0^t \lambda(y;\Theta)dy \quad (2)$$

By estimating model parameter Θ, mean value function λ(t;Θ) or Λ(t;Θ) can be specified the NHPP, regarded as parametric SRGMs. In [32], using SRATS model parameter can be estimated. After finding model parameter, the mean value function of random time t_(n+1) presented below:

$$\Lambda(t_{n+1};\hat{\Theta}) = \int_0^{t_{n+1}} \lambda(y;\hat{\Theta})dy \quad (3)$$

$$\Lambda(t_{n+1} \mid G(t_n) = y_n;\hat{\Theta}) = y_n + \int_{t_n}^{t_{n+1}} \lambda(y;\hat{\Theta})dy = y_n + \Lambda(t_{n+l};\hat{\Theta}) - \Lambda(t_n;\hat{\Theta}) \quad (4)$$

here, l represents the length of prediction and y_n denotes cumulative fault for n-th testing days. For log extreme minimum (SRGM_LEM) [5] and pareto (SRGM_Pareto) [6], the equation are:

$$\Lambda(t;\Theta) = a\big(1 - F(-\log t)\big), \theta \in (a,b,c)$$
$$F(t) = exp(-\exp[\left(-\frac{t-c}{b}\right)]) \quad (5)$$

$$\Lambda(t;\Theta) = aF(t), \theta \in (a,b,c),$$
$$F(t) = 1 - (\frac{c}{t+c})^b \quad (6)$$

3- System Architecture of Proposed Model

The system architecture of the proposed model is shown in Figure 1, where the best method is highlighted in bold. As we know the fault count data is Poisson count data and used as an integer value in the software fault prediction so the underlying data requires to be transformed from Poisson to the Gaussian data in advance. The real data noted by Data Set are passed into the pre-processing state where categorized; With_T and Non_T. With_T represents pre-processed data by the well-known transformation methods; Box-Cox_T, Anscombe_T and Yeo-Johnson_T. Non_T define the non-transformed data. The methodology has two categories; linear regression and time series forecasting. Where Naïve Gauss, and exponential smoothing method of time series forecasting is used. Furthermore, two popular SRGMs; SRGM_LEM and SRGM_Pareto is applied to justify the accuracy. After that, the result is sent to Inverse_Trans state for inverse transformation. For determining better accuracy of long-term software fault prediction, we have compared linear regression with Naïve gauss, exponential smoothing methods, and SRGMs in terms of average and relative error.

3-1- Pre-Processing

In the pre-processing stage, we have tested software fault data for both cases, with and without data transformation. In the case of transformation, Box-Cox [27], Anscombe [28], and Yeo-Johnson [29] methods are used in both linear regression and time series forecasting. In case of linear regression, L_Box-Cox_T, L_Anscombe_T, and L_Yeo-Johnson_T represent Box-Cox, Anscomb and Yeo-Johnson respectively. Besides that, two time series forecasting Naïve gauss and exponential smoothing are represented as NG_Box-Cox_T, NG_Anscomb_T, NG_Yeo-Johnson_T, ES_Box-Cox_T, ES_Anscomb_T and ES_Yeo-Johnson_T sequentially.

Researcher Box and Cox develop Box-Cox power transformation (Box-Cox_T) to transform data into normal shapes for determining the appropriate value of a variable λ. It is used for transforming from arbitrary random data to Gaussian data. But the value of variable λ is difficult to find using log transformation. Additionally, when the data is reverted from transformed to the original state, it always provides a median prediction.

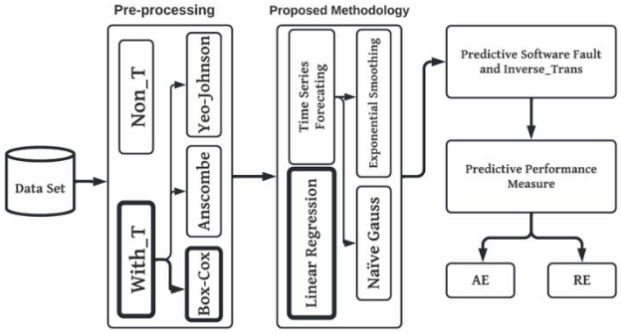


Fig. 1 System architecture of the proposed method.

Another power transformation operator, which is an extension of Box-Cox_T, Yeo-Johnson transformation (Yeo-Johnson_T), is applied to transform the original data into normally distributed numbers. and useful method. Where the distribution is far from Gaussian, distribution is skewed more often. Yeo-Johnson_T is another useful method to handle those data and make them normal (Gaussian).

Table 1: Data Transformation Formulae.

SI No	Name of method		Transformation Formula	Inverse Transformation Formula
1	Box-Cox_T	$\lambda=0$	$y_i = log(x_i)$	$exp(y_{n+l})$
		$\lambda \neq 0$	$y_i = \frac{x_i^{\lambda}-1}{\lambda}$	$(\lambda y_{n+l}+1)^{1/\lambda}$
2	Anscombe_T		$y_i = 2\sqrt{x_i+\frac{3}{8}}$	$\left(\frac{y_{n+l}}{2}\right)^2-\frac{3}{8}$
3	Yeo Johnson	If $\lambda\neq0, y\geq0$	$((x_i+1)^{\lambda}-1)/\lambda$	$(y_{n+l}\lambda+1)^{\frac{1}{\lambda}}-1$
		If $\lambda=0, y\geq0$	$log(x_i+1)$	$exp(y_{n+l})$-1
		If $\lambda\neq2, y<0$	$-((-x_i+1)^{(2-\lambda)}-1)/(2-\lambda)$	$1-(-y_{n+l}(2-\lambda)+1)^{\frac{1}{2-\lambda}}$
		If $\lambda=2, y<0$	$-log(-x_i+1)$	$1-exp(-y_{n+l})$

The Anscombe transform (Anscombe_T) is a variance stabilizing transformation in statistics that converts a random variable with a Poisson distribution into one with a Gaussian distribution that is approximately standard. In photon-limited imaging (astronomy, X-ray), where images naturally obey the Poisson law, it is also commonly employed. In Table 1, different transformation and inverse transformation equations for Box-Cox_T, Anscombe_T, and Yeo-Johnson_T methods are shown. Where input *i*, denotes the cumulative number of software faults detected at *i* (1, 2, 3, … n)-th testing day, output [*l* =1,2,3,…], *l* is the length of prediction. After that, the data inverse transformation formula is used for the final predictive output. When λ=2, Yeo Johnson_T showed ambiguous results so we have to exclude that.

3-2- Forecasting Models

In machine learning, we are required to form predictive outcomes based on different rules, attributes, and patterns from the provided data set. In addition, regression is a well-known method for generating predictive outcomes as a numeric value from given datasets between multiple independent and one targeted single dependent variable. In this paper, we have tested both the transformed and non-transformed dataset in our system using linear regression and time series forecasting methods, such as Naïve gauss and exponential smoothing [30].

Naïve gauss is a sophisticated forecasting model where the last outcome is used as the current predictive value and is widely used for economic and financial time series-related analysis [30] Additionally, another commonly used predictor model is Exponential Smoothing, where prediction is made via an exponentially weighted average of previous observations.

Among different regression models, linear regression is one of the easiest [31], most widely used methods for predictive analysis and checks whether the set of variables generates a better outcome with dependent variables or not. As a supervised method, here, an independent variable (assume *x*) is given, and the linear regression predicts a suitable dependent variable (assume *y*) and finds out linear relations between those variables. Let the specific set of input values x_i (i =1, 2, 3,…n) where n denotes the number of input and the predicted output for the set of input variables by y_i.

In our software fault data, t_i, x_i denotes the number of testing days and a cumulative number of software faults, respectively. So, without loss of generality, the linear regression equation combines a specific set (t_i, x_i) represented by (y_i, x_i). Hence, the input and predicted output are in numeric value. Suppose that the input variable is (x_i= x_1, x_2, x_3 ,...x_n) and the parameters of the model are β_i = [β_0, β_1, β_2,... β_n]. After that, the prediction model would be:

$$y_i \approx \beta_0 + \sum_{i=1}^{n} \beta_i \times x_i \qquad (7)$$

If x_i increases to (x_i= *1, x_1, x_2, x_3 ,...x_n)* after that, y_i would be a parameter of dot product and an independent variable that is

$$y_i \approx \beta_0 + \sum_{i=1}^{n} \beta_i \times x_i \ = \vec{\beta}.\vec{x}_i \qquad (8)$$

Apart from the algebra equation of the straight line, let the slope be "*m*" then the equation become

$$y = mx + b \qquad (9)$$

Then if the slope is "*a*" for the linear regression, the equation become

$$y = ax + b \qquad (10)$$

Where *b* is the *y*-axis intercept and *a* is the slope, then we can easily find the value of the coefficient of *a*, *b*,

$$a = \frac{(\sum_{i=1}^{n} y)(\sum_{i=1}^{n} {y_i}^2) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} x_i y_i)}{n(\sum_{i=1}^{n} {x_i}^2) - (\sum_{i=1}^{n} x_i)^2} \qquad (11)$$

$$b = \frac{n(\sum_{i=1}^{n} x_i y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n(\sum_{i=1}^{n} {x_i}^2) - (\sum_{i=1}^{n} x_i)^2} \qquad (12)$$

In the cases, where the coefficient is zero, the effect of input variable x_i eliminates the model's effectiveness; therefore, the predicted output will be (0 * x = 0). It has a consequence with the regularization method so that learning algorithms of regression models decrease the complexity of the algorithm by putting pressure on the size of the coefficient.

4- Fault Prediction through Proposed Methodology

For software fault prediction, we have to use the previous fault data to predict next faults. In our proposed system, we have applied linear regression, naïve gauss and exponential smoothing time series forecasting model. For theorical better understanding we have illustrated the configuration of our training and prediction scheme in Figure 2 where the x-axis shows the time and y-axis shows the number of faults In the training phase (0, *n*), after pre-processing, processed input data x_i, [i=1,2,3,…, n] and output $y_{n+l,}$ [l =1,2,3…] where prediction length l=5, 10, 15.To predict the cumulative number of software faults for l testing days from the training point t_n, the prediction has to be made for the $n+l$ days. In this scheme, with given λ the Box-Cox, Yeo-Johnson and Anscombe transformations transformed data (x_1, x_2, …, x_n) are used for the input, and the predicted fault (y_{n+1}, y_{n+2}, … , y_{n+l}) are used for the evaluation in the prediction phase.

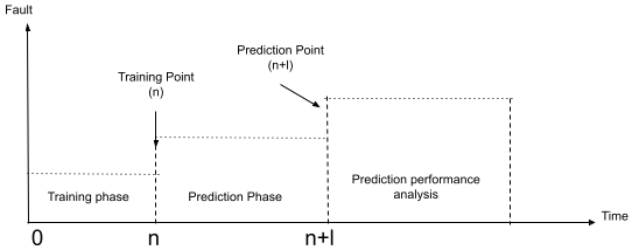


Fig. 2 Training and prediction scheme.

$$y_{n+l} = ax_i + b \qquad (13)$$

In the equation of linear regression (13), the coefficient of *a* and *b* are generated from equation (11) and (12). However, it is not necessary to train all the unknown patterns in principle by linear regression.

$$y_{n+l} = x_n \qquad (14)$$

where x_n is processed input at n-th testing days. Equation (13) represents Naïve gauss [31].

$$y_{n+l} = \alpha x_n + (1 - \alpha) y_n \qquad (15)$$

Where, α is the smoothing constant (0.25, 0.5, 0.75, 1) and y_n is the predictive software fault at n-th testing days. Exponential smoothing is presented in equation (15).

5- Experimental Setup and Results Analysis

5-1- Experimental Setup

We have used three real project datasets cited in the reference [33]; DS_1, DS_2, and DS_3, which contain days and cumulative software fault (grouped) shown in Table 2.

DS_1 contains 62 days, cumulative software fault 133 and the project type is "command and control subsystem". Then the "command and data subsystem" type project data is used in DS_2 and DS_3 where the cumulative number of faults is 225 and 189 for 181 days and 114 days respectively. In the datasets, the length of total training and testing of software faults are given by(10X5), (20X10), (30X10), (40X15). Here, (10X5) means ten inputs, five defined as short and in (40X15) architecture presents forty inputs, fifteen predicted outputs which represented as long time prediction, respectively.

Table 2: Structure of the Data Set in Study.

Dataset	Number of Days	Number of Faults	Type of the project
DS_1	62	133	Command and Control subsystem
DS_2	181	225	Command and Data subsystem
DS_3	114	189	Command and Data subsystem

To find out the desired output via the linear regression, naïve model, and exponential smoothing model, we have implemented our proposed system in python and used NumPy's for pre-processing the data. In Figure 3, the X-axis represents testing days, and the Y-axis shows a cumulative number of software faults. Where the data set DS_1, DS_2 and DS_3 is represented using different colours (red, purple and green).

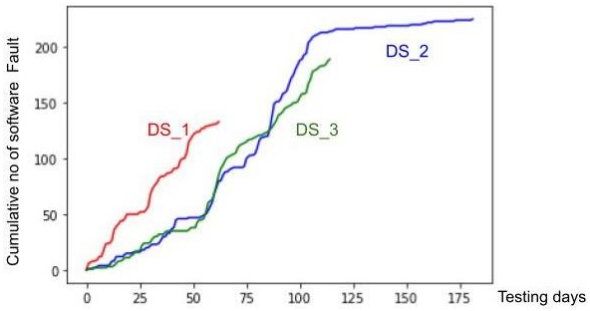


Fig. 3 Dataset Structure

5-2- Performance Measurement Criterion

To measure the performance of the system, we have followed two criteria: average error and relative error. Average error represents a degree to which a group of incorrect values with respect to absolute value. In addition, the relative error is a ratio of absolute error and actual error. For *n*-th testing day, the observation point is given tn. Here (n - l) software fault counts data are used for training in linear regression models. The prediction model is measured by the average error.

$$AE_l = \frac{\sum_{s=1}^{l} RE_s}{l} \qquad (16)$$

where REs is called the relative error for the future time $t = n + s$ and is given by

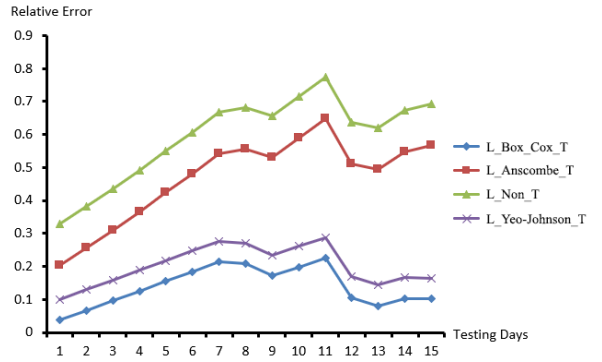
$$RE_s = \left| \frac{\tilde{x}_{n+s} - y_{n+s}}{\tilde{x}_{n+s}} \right| \qquad (17)$$

Here, $\tilde{x}_{n+s}$ is actual software fault and y_{n+s} is predicted software fault at (n+s)-th testing days. So, we regard the prediction model with smaller AE as a better prediction model.

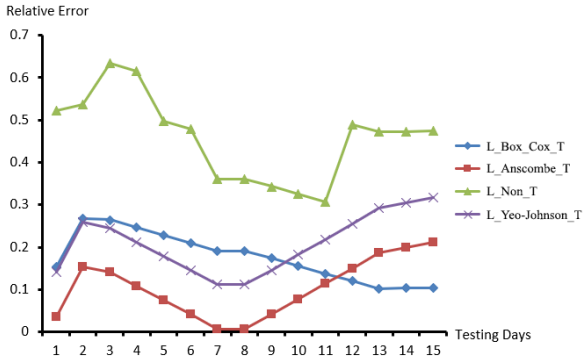
5-3- Result Analysis

For better understanding, we have analysed the proposed result and then compared it with other time series forecasting models (Naïve, Exponential Smoothing) and SRGMs. Figure 4 shows the comparative graphical results where the relative error is shown on the Y-axis with respect to testing days on the X-axis, and the length of the prediction is 15 days based on linear regression. In the datasets DS_1 and DS_3, L_Box-Cox_T (when λ=1) shows less error, whereas in DS_2, L_Anscombe_T is better, and L_Non_T gives the worst condition in all the cases. In the figure, all the behavior of the data transformation in the case of relative error is shown in different colors such as, red with a square representing shows L_Anscombe_T, light green with a triangle representing L_Non_T, L_Box-Cox_T represented in sky blue color with a diamond sign and the L_Yeo -Johnson_T is showed in violet with a cross symbol. In the DS_1, L_Non_T and L_Yeo -Johnson_T continuously changed the states while the L_Box-Cox_T showed stable and comparatively less error. In addition, furthermore, in DS_2, the worst result was provided by the L_Non_T on the other hand other methods showed the average result. Furthermore, L_Box_Cox_T clearly outperformed all other methods in DS_3.

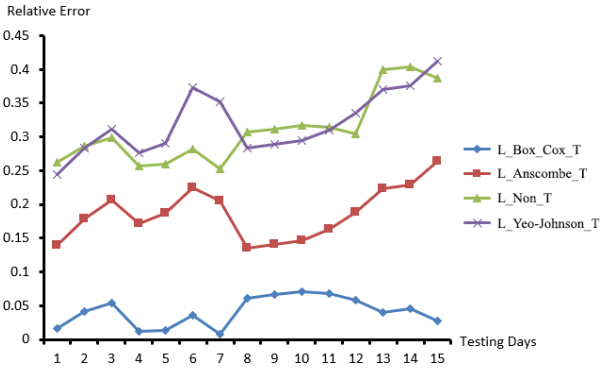
In Table 3, prediction performance results of software fault based on average error (AE) for three datasets denoted as DS_1, DS_2, and DS_3, respectively. Here, the architecture represents the number of input and predicted output. From the data table, we can say that for different values of λ, L_Box-Cox_T showed better results. In the 10X5 architecture, for both DS_2 and DS_3, L_Box-Cox_T was preferable, but in DS_1, L_Yeo-Johnson_T performed better. Besides that, L_Box-Cox_T outperformed other methods in 20X10 architecture for all datasets and in DS_1 and DS_3 for 40X15 architecture. Exceptionally in 30X10 architecture, the L_Box-Cox_T, L_Yeo-Johnson_T as well Non_T give less error in DS_2, but in DS_1 L_Yeo-Johnson_T and in DS_3 L_Anscombe_T was better. The best performance of every architecture is highlighted in bold text. Comparison of AEs for the naïve gauss and exponential smoothing time series forecasting model considering the three datasets is shown Table 4 and Table 5.



(a)



(b)



(c)

Fig. 4 Behavior of relative error with linear regression when l=15 for (a) DS_1, (b) DS_2, (c) DS_3.

By analyzing the data, we got that naïve gauss provides better result than exponential smoothing in every data set. To be mentioned that, when $\alpha = 1$, Box-Cox and the Yeo-Johnson transformed data provides same error. In addition, usually, whenever we have increased the input days, then the result showed an improved outcome (shown in 40X15 architecture). But the linear regression model for all architecture has shown superior predictive results than the time series forecasting models.

Table 3: Comparison of AEs for linear Regression Model with Different Transformation Methods.

Architecture	Transformation Method	Value of λ	DS_1	DS_2	DS_3
10X5	L_Box-Cox_T	0.5	0.6330	0.4296	0.3160
		1	0.7236	0.6142	0.1526
		2	1.6462	0.7844	0.3904
	L_Anscombe_T	-	0.6682	0.4592	0.3256
	L_Yeo-Johnson_T	0	0.3440	0.9122	2.1502
		1	0.9236	0.7646	0.1975
	L_Non_T	-	1.1236	0.6142	0.3526
20X10	L_Box-Cox_T	0.5	2.8586	0.7093	0.3983
		1	1.2853	0.1631	0.6625
		2	0.3592	0.1543	0.8293
	L_Anscombe_T	-	2.8256	0.6618	0.4263
	L_Yeo-Johnson_T	0	5.4305	1.2803	1.6912
		1	1.0853	0.1584	0.6846
	L_Non_T	-	1.2853	0.1631	0.6625
30X10	L_Box-Cox_T	0.5	0.9270	0.3523	0.2743
		1	0.6003	0.3034	0.4853
		2	1.6852	0.6728	0.8468
	L_Anscombe_T	-	0.8992	0.3089	0.2525
	L_Yeo-Johnson_T	0	8.3375	2.3368	2.1974
		1	0.6000	0.3034	0.4853
	L_Non_T	-	0.6867	0.3034	0.4853
40X15	L_Box-Cox_T	0.5	1.4036	0.3351	1.2571
		1	0.0411	0.1764	0.1380
		2	1.1772	0.9081	0.1415
	L_Anscombe_T	-	0.1866	0.1031	1.2107
	L_Yeo-Johnson_T	0	7.4240	2.4961	0.9557
		1	0.3202	0.2077	0.3477
	L_Non_T	-	0.3096	0.4596	0.5941

Table 4: Comparison of AEs for Naïve Gauss Model with Different Transformation Methods.

Architecture	Transformation Method	Value of λ	DS_1	DS_2	DS_3
10X5	NG_Box-Cox_T	0.5	5.1018	5.9957	11.2750
		1	9.7363	11.3213	20.3810
		2	17.8097	20.2702	42.9644
	NG_Anscombe_T	-	4.9690	5.8260	10.3004
	NG_Yeo-Johnson_T	0	2.9077	4.3616	11.4400
		1	9.7363	11.3213	20.3810
20X10	NG_Box-Cox_T	0.5	1.8409	1.6517	3.8768
		1	3.5839	3.2436	7.4225

		2	6.8030	6.2574	26.9249
	NG_Anscombe_T	-	1.4830	1.6263	3.7993
	NG_Yeo-Johnson_T	0	0.8928	0.9966	2.5400
		1	3.5839	3.2436	7.4225
30X10	NG_Box-Cox_T	0.5	1.0929	2.1003	1.1349
		1	2.1679	4.1139	2.2350
		2	4.2653	7.8961	13.0742
	NG_Anscombe_T	-	1.2691	2.0781	1.1212
	NG_Yeo-Johnson_T	0	0.4916	1.1539	0.6425
		1	2.1679	4.1139	2.2350
40X15	NG_Box-Cox_T	0.5	1.1487	0.8004	1.3202
		1	2.2689	1.5627	2.5800
		2	4.4279	2.9826	7.0538
	NG_Anscombe_T	-	1.2033	0.7945	1.3086
	NG_Yeo-Johnson_T	0	0.4886	0.4067	0.6954
		1	2.2689	1.5627	2.5800

Comparison of two SRGMs, SRGM_Pareto and SRGM_LEM for the same three dataset is presented in Table 6. After getting the mean value function using Equation (5) and (6), we have calculated long term prediction by Equation (4). SRGM_Pareto performed well in the 10X5 architecture however the SRGM_Pareto provided superior result for DS_2 and DS_3 in all other architectures.

In Table 7, the best method according to AEs from linear regression (Table 3), time series forecasting (Table 4 and Table 5) and SRGMs (Table 6) is presented. Linear regression with transformation performed comparatively better result than the other models. Then to be more specific, Box-Cox transformation with linear regression (L_Box-Cox_T) outrun all others. Notwithstanding, SRGMs showed good performance in 30X10 architecture. Subsequently, among the three datasets, only in one case, architecture 40X15 L_Anscombe_T offers superior results than other data models. From the result, we can also observe that the traditional method L_Non_T has not shown better results in any cases.

Table 5: Comparison of AEs for Exponential Smoothing model with different transformation methods.

Architecture	**Transformation Method**	**Value of λ**	**DS_1**	**DS_2**	**DS_3**
10X5	ES_Box-Cox_T	0.5	6.8332	7.7045	15.0115
		1	12.8170	14.4343	26.2358
		2	22.7536	25.4941	26.2358
	ES_Anscombe_T	-	7.1665	7.4876	13.5335
	ES_Yeo-Johnson_T	0	3.9619	5.6239	15.4431
		1	12.8171	14.4345	20.3810
20X10	ES_Box-Cox_T	0.5	2.2531	2.1116	5.1992
		1	4.3536	4.1366	9.8457
		2	8.1509	7.9421	7.4225
	ES_Anscombe_T	-	1.7020	2.0790	5.0920
	ES_Yeo-Johnson_T	0	1.0996	1.2775	2.5412
		1	4.3536	4.1366	2.3970
30X10	ES_Box-Cox_T	0.5	1.5773	2.7959	1.7096
		1	3.0995	5.4466	3.3406
		2	5.9938	10.3474	7.4225
	ES_Anscombe_T	-	1.9551	2.7662	1.6887
	ES_Yeo-Johnson_T	0	0.7176	1.5450	1.6425
		1	3.0995	5.4466	
40X15	ES_Box-Cox_T	0.5	1.5464	1.0887	1.6128
		1	3.0418	2.1135	3.1425
		2	5.8889	3.9919	2.2350
	ES_Anscombe_T	-	1.6053	1.0805	1.5985
	ES_Yeo-Johnson_T	0	0.6603	0.5561	1.6954
		1	3.0418	2.1135	2.5800

Table 6: Comparison of AEs for SRGMs with Different Transformation Methods.

Architecture	SRGM Model	DS_1	DS_2	DS_3
10X5	SRGM_LEM	1.4259	4.7747	14.4099
	SRGM_Pareto	1.2472	0.9279	1.5445
20X10	SRGM_LEM	0.4354	1.00171	1.2371
	SRGM_Pareto	0.6770	0.3921	0.2609
30X10	SRGM_LEM	0.2005	0.4511	0.4904
	SRGM_Pareto	0.4054	0.2425	0.1474
40X15	SRGM_LEM	0.1128	0.1948	0.3073
	SRGM_Pareto	0.2814	0.1423	0.1242

Table 7: Best Method based on AEs (Among linear Regression and Time Series Forecasting and SRGMs.)

Architecture	**DS_1**	**DS_2**	**DS_3**
10X5	L_Yeo-Johnson_T	L_Box-Cox_T	L_Box-Cox_T
20X10	L_Box-Cox_T	L_Box-Cox_T	SRGM _Pareto
30X10	SRGM_LEM	SRGM_Pareto	SRGM_Pareto
40X15	L_Box-Cox_T	L_Anscombe_T	SRGM_Pareto

6- Conclusion

In this paper, we have presented a long-term software fault prediction model based on linear regression with data transformation. In our model, we have pre-processed three actual software development project datasets with three

different transformation methods. Through a comprehensive analysis with non-transformation, time series forecasting method and conventional SGRMs, it has been shown that linear regression with Box-Cox (L_Box-Cox_T) could work well to predict the software fault in short time prediction. On the other hand, SRGM_Pareto showed better result for 30X10 architecture. Additionally, when we have increased input testing days for long-term prediction the relative errors have decreased. As a result, linear regression-based model was much attractive, though SGRMs are used often for long term software fault prediction. For further development, we will study to find an optimal value of λ for Box-Cox and Yeo-Johnson power transformation. Moreover, we will construct the prediction interval for better accuracy by well-known methods and apply our method in experimental or simulation data using Monte Carlo simulation methods for better prediction.

Acknowledgement.

This research is funded by Woosong University Academic Research in 2023.

References

[1] J. Stilgoe, "Who Killed Elaine Herzberg?," in Who's Driving Innovation? New Technologies and the Collaborative State, J. Stilgoe, Ed. Cham: Springer International Publishing, 2020, pp. 1–6. doi: 10.1007/978-3-030-32320-2_1.

[2] B. P. Murthy, N. Krishna, T. Jones, A. Wolkin, R. N. Avchen, and S. J. Vagi, "Public Health Emergency Risk Communication and Social Media Reactions to an Errant Warning of a Ballistic Missile Threat — Hawaii, January 2018," *Morb. Mortal. Wkly. Rep.*, vol. 68, no. 7, pp. 174–176, Feb. 2019, doi: 10.15585/mmwr.mm6807a2.

[3] H. Pham, *System Software Reliability*. Springer Science & Business Media, 2007.

[4] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Autom. Softw. Eng.*, vol. 17, no. 4, pp. 375–407, Dec. 2010, doi: 10.1007/s10515-010-0069-5.

[5] A. L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 12, pp. 1411–1423, Dec. 1985, doi: 10.1109/TSE.1985.232177.

[6] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of competing software reliability predictions," *IEEE Trans. Softw. Eng.*, vol. SE-12, no. 9, pp. 950–967, Sep. 1986, doi: 10.1109/TSE.1986.6313050.

[7] S. Santosa, R. A. Pramunendar, D. P. Prabowo, and Y. P. Santosa, "Wood Types Classification using Back-Propagation Neural Network based on Genetic Algorithm with Gray Level Co-occurrence Matrix for Features Extraction," 2019.

[8] Y. Wang, D. Niu, and L. Ji, "Short-term power load forecasting based on IVL-BP neural network technology," *Syst. Eng. Procedia*, vol. 4, pp. 168–174, Jan. 2012, doi: 10.1016/j.sepro.2011.11.062.

[9] "Long-term Software Fault Prediction with Robust Prediction Interval Analysi...: EBSCOhost."

[10] M. Begum and T. Dohi, "Optimal Release Time Estimation of Software System using Box-Cox Transformation and Neural Network," *Int. J. Math. Eng. Manag. Sci.*, vol. 3, pp. 177–194, Jun. 2018, doi: 10.33889/IJMEMS.2018.3.2-014.

[11] M. Begum and T. Dohi, "Estimating prediction interval of cumulative number of software faults using back propagation algorithm," May 2016.

[12] M. Begum and T. Dohi, *optimal software release decision via artificial neural network approach with bug count data*. 2016.

[13] M. Begum and T. Dohi, "Prediction Interval of Cumulative Number of Software Faults Using Multilayer Perceptron," vol. 619, pp. 43–58, Jan. 2016, doi: 10.1007/978-3-319-26396-0_4.

[14] M. Begum and T. Dohi, "A Neuro-Based Software Fault Prediction with Box-Cox Power Transformation," *J. Softw. Eng. Appl.*, vol. 10, no. 3, Art. no. 3, Mar. 2017, doi: 10.4236/jsea.2017.103017.

[15] M. Begum and T. Dohi, "Optimal stopping time of software system test via artificial neural network with fault count data," *J. Qual. Maint. Eng.*, vol. 24, pp. 00–00, Jan. 2018, doi: 10.1108/JQME-12-2016-0082.

[16] Y. Kamei and E. Shihab, "Defect Prediction: Accomplishments and Future Challenges," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Mar. 2016, vol. 5, pp. 33–45. doi: 10.1109/SANER.2016.56.

[17] V. R. Basili, "The experimental paradigm in software engineering," in *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, Berlin, Heidelberg, 1993, pp. 1–12. doi: 10.1007/3-540-57092-6_91.

[18] T. M. Khoshgoftaar *et al.*, "Predicting fault-prone modules with case-based reasoning," in *Proceedings The Eighth International Symposium on Software Reliability Engineering*, Nov. 1997, pp. 27–35. doi: 10.1109/ISSRE.1997.630845.

[19] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4626–4636, Apr. 2011, doi: 10.1016/j.eswa.2010.10.024.

[20] K. Thantirige, A. K. Rathore, S. K. Panda, S. Mukherjee, M. A. Zagrodnik, and A. K. Gupta, "An open-switch fault detection method for cascaded H-bridge multilevel inverter fed industrial drives," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2016, pp. 2159–2165. doi: 10.1109/IECON.2016.7794032.

[21] M. Islam, M. Akhtar, and M. Begum, *Long short-term memory (LSTM) networks based software fault prediction using data transformation methods*. 2022, p. 6. doi: 10.1109/ICAEEE54957.2022.9836388.

[22] M. Islam, M. Begum and M. Akhtar, *Recursive Approach for Multiple Step-Ahead Software Fault Prediction through Long Short-Term Memory (LSTM)*. p. 10.

[23] H. K. Dam *et al.*, "Lessons Learned from Using a Deep Tree-Based Model for Software Defect Prediction in Practice," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, May 2019, pp. 46–57. doi: 10.1109/MSR.2019.00017.

[24] D. Sharma and P. Chandra, "Linear regression with factor analysis in fault prediction of software," *J. Interdiscip. Math.*, vol. 23, pp. 11–19, Jan. 2020, doi: 10.1080/09720502.2020.1721641.

[25] D. J. Pedregal, "Time series analysis and forecasting with ECOTOOL," *PLOS ONE*, vol. 14, no. 10, p. e0221238, Oct. 2019, doi: 10.1371/journal.pone.0221238.

[26] O. Nyarko-Boateng, A. F. Adekoya, and B. A. Weyori, "Predicting the actual location of faults in underground optical networks using linear regression," *Eng. Rep.*, vol. 3, no. 3, p. eng212304, 2021, doi: 10.1002/eng2.12304.

[27] G. E. P. Box and D. R. Cox, "An Analysis of Transformations," *J. R. Stat. Soc. Ser. B Methodol.*, vol. 26, no. 2, pp. 211–252, 1964.

[28] F. J. Anscombe, "The Transformation of Poisson, Binomial and Negative-Binomial Data," *Biometrika*, vol. 35, no. 3/4, pp. 246–254, 1948, doi: 10.2307/2332343.

[29] S. Weisberg, "Yeo-Johnson Power Transformations." 2001.

[30] E. S. Gardner, "Exponential smoothing: The state of the art—Part II," *Int. J. Forecast.*, vol. 22, no. 4, pp. 637–666, Oct. 2006, doi: 10.1016/j.ijforecast.2006.03.005.

[31] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," *WIREs Comput. Stat.*, vol. 4, no. 3, pp. 275–294, 2012, doi: 10.1002/wics.1198.

[32] H. Okamura and T. Dohi, "SRATS: Software reliability assessment tool on spreadsheet (Experience report)," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, Nov. 2013, pp. 100–107. doi: 10.1109/ISSRE.2013.6698909.

[33] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*. Los Alamitos, Calif.: New York: McGraw-Hill, 1996.

[34] A. Rasoolzadegan, "A new approach to the quantitative measurement of software reliability," 2015.