



Evaluating the Predicting Capabilities of Software Reliability Model

Shailee Choudhary¹ and Yogita Kansal²

¹Department of Information Technology, Manav Rachna College of Engineering, India

²Department of Computer Science & Engineering, Manav Rachna College of Engineering, India

Abstract: Software Reliability Engineering focuses on dealing with the best quality of software. It covers the maintainability, capability; install ability and much more features of software. The researchers have proved that there are thousands of models for evaluating the reliability of software. But these models lack in prediction as the parameters selection criteria are different in each model. Furthermore, when we talk about prediction the best way to carry this out is via Neural Network model (i.e. Non-Parametric model). The other idea is to unify the neural network with analytical model (i.e. Software Reliability Parametric model) so as to improve the prediction efficiency of analytical models. The matter of concern shown in this paper is although we have both parametric and non-parametric models but how to judge which one is better. Hence, the paper concludes the optimal model.

Keywords: Software Reliability, Software Reliability Model, Cumulative Failure, Execution time, Logistic Growth Curve Model, Goel-Okumoto NHPP model, Yamada S-shaped model, Neural Network, Feed Forward Neural Network model, Elman Back propagation Model, Cascade Correlation model,

I. Introduction

In Today's era, no human being can predict his/her life without software. Software has been used in every field whether it has been telecommunications, aircrafts, automobiles, home appliances and so on. If any inconsistency happens in the software it results in disastrous situation and at some point of time it becomes the question of life also. If the user has found an error in the software being used then the earliest detection and removal of that error is not a healthy solution. As fixing the error may arise more problems. In 1991, after changing three lines of code in a signaling program which contains millions lines of code, the local telephone systems in California and along the Eastern seaboard came to stop [1]. Even if the operating environment of software changes it also results in incalculable problems. After the success of Ariane 4 rocket, the maiden flight of Ariane 5 ended up in flames while design defects in the control software were unveiled by faster horizontal drifting speed of the new rocket [1]. Software Reliability is the probability of error free operations in a specified environment for a specified period of time [2]. Software Reliability model is used to measure the reliability of the software. If a user is rarely observing the number of failures then we can say the software is more reliable. Failure is an event when triggered results in preventing the successful execution of the system. So, the best decision to get rid off from this event is the earliest detection. But there are many cases where in spite of having careful observation while testing the software, there are some bugs that left in the program and are detected after its deployment. Hence, it becomes necessary to predict the reliability of software, so that the developer must be prepared to face the results. The researchers of software reliability engineering field has already discusses about thousand of reliability models. All of these models are best with their own conditions. In other words, none of the model can be used in all circumstances. Hence, there is a need to develop such models which must be executable at all times regardless of what assumptions are. Moreover, the neural networks are known for their predicting skills. Thus, for best prediction we have chosen different set model of neural network. The models are then classified as Parametric and Non-Parametric model respectively having common set of values for prediction. So, now we have two models i.e. Analytical models and Neural Network models, with an absurd that which one is better to opt. Hence, this paper covers the comparison between these two models which helps the user in finding the optimal one.

II. Neural Network

Our basic computational element (model neuron) is often called a **node** or **unit** [3]. It receives input from some other units, or perhaps from an external source. Each input has an associated **weight** w , which can be modified so as to model synaptic learning. The unit computes some function f of the weighted sum of its inputs.

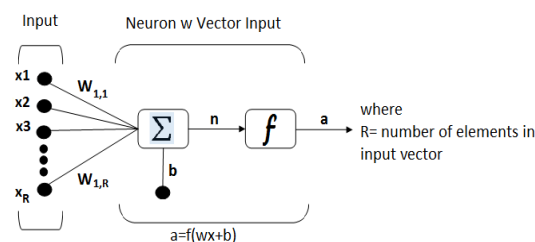


Figure 1: A simple neuron

While designing the neural network the individual element inputs are $x_1, x_2, \dots, x_R$ are multiplied by the weights $w_{1,1}, w_{1,2}, \dots, w_{1,R}$ and the weighted values are fed to the summing junction. Their sum is simply Wx , the dot product of the single row matrix W and the vector x . R is the number of elements in the input vector.

$$a = f(\sum w_{1,R} x_R + b) \quad (1)$$

- The weighted sum is $\sum w_{1,R} x_R$ called the **net input** to unit 1, often written net_1 .
- Note that $w_{1,R}$ refers to the weight from unit R to unit 1 (not the other way around).
- The function f is the unit's **activation function**. In the simplest case, f is the sigmoid function, and the unit's output is

$$F(n) = 1 / (1 + e^{-n}) \quad (2)$$

Neural networks learn by example. The *learning rule* is provided with a set of examples (the *training set*) of proper network behavior $\{x_1, t_1\}, \{x_2, t_2\}, \dots, \{x_Q, t_Q\}$ where x_q is an input to the network, and t_q is the corresponding correct (*target*) output. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets. The *Perceptron learning rule* falls in this supervised learning category. There is also the *Supervised Hebbian learning*.

The three neural network models that are taken into account are Feed Forward Neural network, Elman Back propagation neural network and Cascade Correlation network. A brief description of these models is explained onwards.

A. **Feed Forward Neural Network**

Feed Forward neural network [3] comprises a layer of neurons that receive inputs from the outside world, and one or more hidden layers having no connection with external world.

B. **Elman Back propagation Network**

Elman neural network is used to construct memory in the network. The network has input layer, hidden and output layers. Special units called context units save previous output values of hidden layer neurons. Context unit values are then fed back fully connected to hidden layer neurons and thus they serve as additional inputs to the network.

C. **Cascade Correlation Network**

Cascade correlation network [4] consists of a cascade architecture, in which hidden neurons are added to the network one at a time and do not change after they have been added. It is called a cascade because the output from all neurons already in the network feed into new neurons. As new neurons are added to the hidden layer, the learning algorithm attempts to maximize the magnitude of the correlation between the new neuron's output and the residual error of the network which we are trying to minimize.

D. **Adapting NN for prediction**

The problem of software reliability prediction can be stated as follows. Given a sequence of cumulative execution time $((i_1, \dots, i_k) \in I_k(t))$ and the corresponding observed accumulated faults $((o_1, \dots, o_k) \in O_k(t))$ up to the present time t , and $(i_{k+h}(t + \Delta))$ representing the cumulative execution time at the end of a future test session $k+h$, predict the corresponding cumulative number of faults $o_{k+h}(t + \Delta)$. For the prediction horizon $h=1$, the prediction is called the next step prediction and for $h=n$ ($n \geq 2$) consecutive test intervals, it is known as n -step ahead prediction [5]. Here

$$\Delta = \sum_{j=k+1}^{k+h} \Delta_j \quad (3)$$

represents the cumulative execution time of h consecutive future test sessions. You can use Δ to predict the number of accumulated faults after some specified amount of testing.

III. Overview of Analytical Models

There are thousands of software reliability models. Some of them which are taken into account is Logistic growth curve model, Goel-Okumoto NHPP model and Yamada S-shaped model. All these models have certain assumptions which are inherently different from one other. And there is much possibility that under some circumstances these models fail to give the relevant output. Thus, to make them more consistent and atomic we have decided to use two parameters i.e. failure specification and execution time. Failure specification is the number of observed failures which are noticed earlier. Execution time is the time taken to observe that failure. A software reliability model is known as one of the fundamental technologies for quantitative software reliability assessment, and playing an important role in software project management for producing a highly-reliable software system [2]. A brief description of the models is as shown below:

E. **Logistic-Growth Curve Model**

In general, with the detection and removal of software failures with respect to time, the confidence of reliable software increased i.e. the software becomes more reliable with the number of fixed defects. Therefore, under some circumstances, the models developed to forecast the economic population growth could also be applied to predict software reliability growth. These models simply fit the cumulative number of detected faults at a given

time with a function of known form. Logistic growth curve model is one of them and it has an S-shaped curve. Its mean value function and intensity function are [6].

$$m(t) = \frac{a}{1 + k \cdot \exp[-bt]} \quad a > 0, b > 0, k > 0 \quad (4)$$

$$\lambda(t) = \frac{ab \exp[-bt]}{(1 + k \cdot \exp[-bt])^2} \quad a > 0, b > 0, k > 0 \quad (5)$$

where a is the expected total number of faults to be eventually detected and k and b are parameters which can be estimated by fitting the failure data.

F. **Goel-Okumoto model**

The Goel Okumoto model was first proposed by Goel and Okumoto that lies in the category of Non homogeneous poisson process model i.e. NHPP which signifies that the mean value function is non linear. Hence, this model is also known as exponential NHPP model. From the broader perspective the goel-okumoto model is the failure count model by which the software reliability is calculated through calculating the number of faults in the specific time interval. The model further comprises of some assumptions like the cumulative number of failures by time t follows a poisson process and must be independent; defects must be repaired immediately after the discovery and the repair must be perfect. Its mean vale function and intensity function are [6].

$$m(t) = a(1 - \exp[-bt]) \quad a > 0, b > 0 \quad (6)$$

$$\lambda(t) = ab \cdot \exp(-bt) \quad a > 0, b > 0 \quad (7)$$

where a is the expected total number of faults to be eventually detected and b represents the fault detection rate. In fact, it follows that [4] .

$$\lim_{t \rightarrow \infty} m(t) = a$$

G. **Yamada S-shaped Model**

The S-shaped reliability growth model was proposed by Ohba and is the illustrative of the gamma distribution class. Here per fault distribution is gamma. The software error detection process can be described as an S-shaped growth curve to reflect the initial learning curve at the beginning, as team members become familiar with the software, followed by growth and then leveling off as the residual faults become more difficult to uncover. Its mean value function and intensity function are:

$$m(t) = \frac{a \cdot (1 - \exp[-bt])}{1 + k \cdot \exp[-bt]} \quad a > 0, b > 0, k > 0 \quad (8)$$

These models start working as follows. Firstly, the network is trained through execution time as input and cumulative number of faults as the target output. These two parameters are selfsame in all the three models which make it consistent. For training the network, the first thing that comes in our mind is the evaluation of the transfer function, without which the training is not possible. The new activation function has been constructed for every model is as depicted in equation (9), (10), (11)

$$F(n) = \frac{1}{1 + \exp[-w_{1,R} t_{1,R}]} \quad \text{logistic growth curve} \quad (9)$$

$$F(n) = (1 - \exp[-w_{1,R} t_{1,R}]) \quad \text{Goel-okumuto NHPP model} \quad (10)$$

$$F(n) = (1 - ((1 + w_{1,R} t_{1,R}) \cdot \exp[-w_{1,R} t_{1,R}])) \quad \text{Yamada S-shaped model} \quad (11)$$

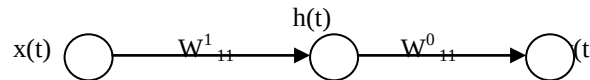


Figure 2: Feed Forward Neural Network

Where $x(t)$ is the execution time which acts as the input while $w(t)$ is the weight whose range lies in between -0.5 to 0.5.

IV. **Software Reliability Demonstration**

For implementing these two above mentioned models, we need to first choose the testing and debugging data. For that, we have select the data set observed by Yoshira Tohma taken from [7] Table 4. The data set comprises of execution time and the cumulative faults. In the data set, execution time was reported in terms of days and faults in terms of cumulative faults at the end of each day. The total testing and debugging time was 46 days and there were 266 faults.

The next step is to train the network with 100 epochs, only half of the data set is used for training and the other residual data set is used for testing the correct output. As already discussed the training pair is built with the combination of failures and execution time i.e. $\{t_i, f_i\}$ where t_i is the execution time of the module and f_i is the failure observed within the module. This data set will act as the failure history provided to the network. Hence, the total inputs (i.e. time) were 23 days and the total targets (i.e. number of faults) were 174 faults. For training the network first we need to tune the model with the weights and bias. The value for bias for every model is +1 while the weights must be in the range between $[-0.5, 0.5]$.

After the successful training of the network, the next task is to simulate the network so that we can test the accurateness of the network. The next step prediction is analyzed i.e. the output value of $(t+\Delta)$ is observed where Δ can be 1. If there is any inconsistency lies in between the actual and predicted faults, then the error is calculated. So that we can know where we are wrong. The final output is as shown in figure 3, plotting predicted faults with respect to the time. The plot shows both the predicted results for parametric models (above) and non-parametric models (below) respectively.

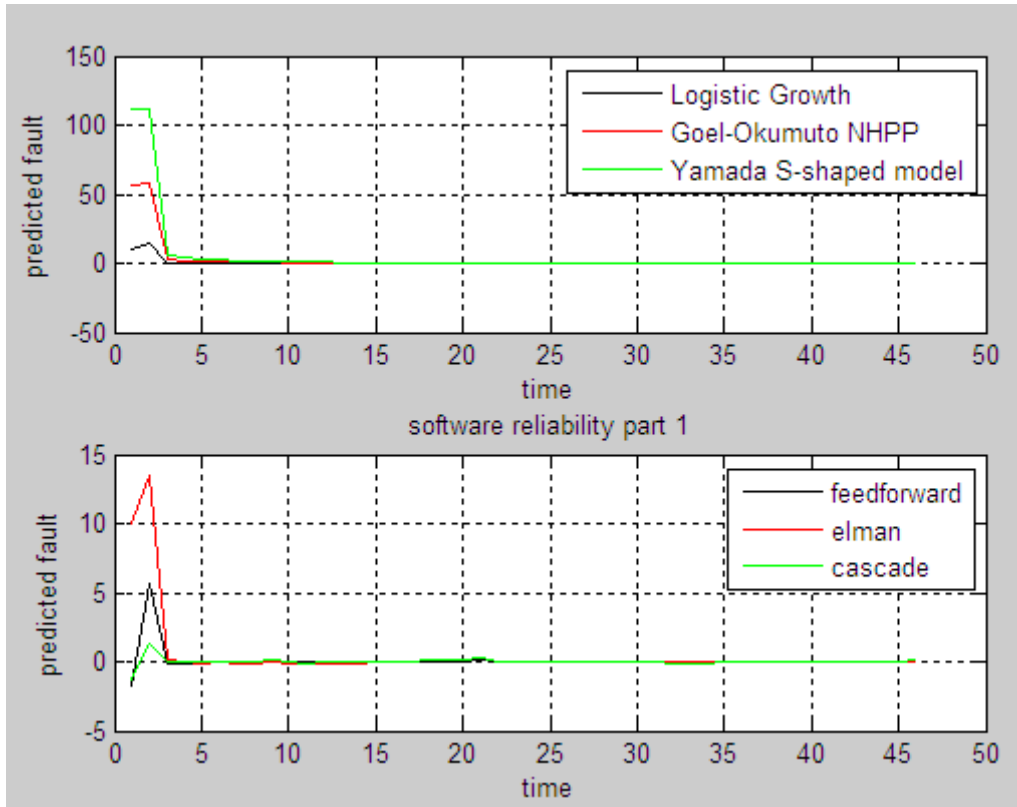


Figure 3: Plot of Predicted Faults with respect to time

V. Analytical model vs Neural Network Model

This section describes the factors used for comparing the models i.e. analytical and neural network model. The factors are relative error, average error, average bias, normalized average error and at last a rank is provided to every model according to its efficiency. The comparison criteria we engage to compare various model's performance are described as follows [2].

1. The Relative Error (RE)

$$RE = \frac{\mu_i - f_i}{f_i} \quad (12)$$

where f_i is the actual faults and μ_i is the predicted faults at the end of testing.

2. The Average Error (AE)

$$AE = \frac{1}{n-1} \sum_{i=1}^{n-1} |RE| \quad (13)$$

the AE measures how well a model predicts throughout the test phase. The average error is calculated for overall testing and debugging session.

3. The Average Bias (AB)

$$n-1$$

$$AB = \frac{1}{n-1} \sum_{i=1} RE \quad (14)$$

All the parameters mentioned above can be used to compare the predictive accuracy of models within a single data set only.

4. The Normalized AE (NAE) aids in ranking the models according to the predictive accuracy across different software projects (i.e. for more than one data set).

$$NAE^m = \frac{AE^m}{AE^{Mmax}} \quad (15)$$

where Mmax be the competing model that has the average error measure for the data set s. Let AE^{Mmax} be its average error measure. Thus $NAE^m=1.0$ when $m=Mmax$ and $0.0 < NAE^m < 1.0$ for $m > Mmax$ or $m < Mmax$.

5. The Overall Rank Metric (R_m)

$$R_m = \sum_{s=1}^{ND} \gamma NAE^m \quad (16)$$

where ND is the number of data sets used in comparing models and γ is the weighting factor. If the weights for both the data set are taken same then the weighting factor is assumed to be 1.

VI. Prophecy Results

Most training methods initialize neural network and analytical weights with random values at the beginning of training, which causes the network to converge to different weight set at the end of each training session. You can thus get different prediction results at the end of each training session. To compensate for these prediction variations, you can take an average over large number of trails. Here, we have examined our data for 100 epochs only. For calculating the averaged error, we need to determine the relative error. The relative error is calculated for whole 46 days and 60days data as shown in figure 3.

Further, the table 1 shows the values observed after the long procedure of training and simulation. The values also aid the user to determine the optimal model that should be used for prediction.

Table 1: Showing the Observed Values for the Comparison Parameters

Models Parameters	Logistic Growth model	Goel- Okumuto NHPP model	Yamada S- shaped model	Feed Forward Neural Network model	Elman Back propagation model	Cascade Correlation model
Average Error	0.3190(3)	1.5470 (5)	13.5657 (6)	0.1646(2)	0.5439(4)	0.0698(1)
Average Bias	0.2672 (3)	1.5444 (5)	13.2864 (6)	0.0523(2)	0.4651 (4)	0.0140(1)
Normalized Average Error	0.0235 (1)	0.1140 (2)	1 (5)	0.3027 (4)	1 (5)	0.1283(3)
Rank	3	5	6	2	4	1

VII. Conclusion

After having a broad discussion about the parameters chosen for comparison, the selected data set has been applied over these parameters. So that the values has been calculated for each model. The values calculated can depict itself which case is best. As shown in table 1 the values for average error, average bias and normalized average error to the cascade correlation model are the least values, which makes cascade the best model among others. As lowest the average error and average bias for the data set involved, results in more reliable software. But if we talk only about the analytical models then logistic growth curve model gains the optimality with the values, $AE= 0.3190$, $AB= 0.2672$ and $NAE= 0.0235$ i.e. > Logistic Growth model > Goel-Okumuto NHPP model > Yamada S-shaped model. While the cascade correlation model is the best among all the neural networks whose $AE= 0.0698$, $AB= 0.0140$ and $NAE= 0.1283$ that says, Cascade Correlation model > Feed Forward Neural Network model > Elman Back propagation model. Thus, the overall comparison depicts Cascade Correlation model > Feed Forward Neural Network model > Logistic Growth model > Elman Back propagation model > Goel-Okumuto NHPP model > Yamada S-shaped model. Hence, we can conclude cascade correlation model is the superior one.

References

- [1] Jiantao Pan , Spring, 1999, "Software Reliability", Carnegie Mellon University.
- [2] Lyu, M.R; 1996, "Handbook of Software Reliability Engineering", McGraw-Hill.
- [3] N.Karunanithi, Y.Malaiya, and D. Whitley, "Prediction of Software Reliability using Neural Networks," Proc. Int'l Symp. Software Reliability Engineering; May 1991,pp. 124-130.
- [4] S.Fahlman and C.Lebiere, "The Cascade Co-relation Learning Architecture" Pittsburgh, Feb,1990.
- [5] Arora Mamta, Choudhary Shaillee 2013, "Software Reliability Prediction using Neural Networks", Manav Rachna College of Engineering.
- [6] Mariam Rahmani, Azad Azadmanesh, 2011, "Exploitation of Quantitative Approaches to Software Reliability", Technical Report, University of Nebraska at Omaha.
- [7] Y. Tohma et al; "Parameter Estimation of the Hyper-Geometric Distribution Model for Real Test/Debug Data," Tokyo Institute of Technology, 1990.