International Journal of Research and Development in Applied Science and Engineering (IJRDASE)
ISSN: 2454-6844

Model for Software Testing and Quality Assessment using ANN Approach

Khushter Kaifi
Department of CSE
Shri Venkateshwara University
k.kaifi@hotmail.com

Dr. Anshu Srivastava
Department of CSE
Shri Venkateshwara University
anshuqrat114@gmail.com

Dr. Qamar Parvez Rana
Dept of Computer Center
Jamia Hamdard University
qprana@jamaihamdard.ac.in

***Abstract*--Almost 50% of the software production development cost is expended in software testing. It consumes resources and adds nothing to the product in terms of functionality. Therefore, much effort has been spent in the development of automatic software testing tools in order to significantly reduce the cost of developing software. One objective of software testing is to find errors and program structure faults. Therefore, a systematic testing system has to differentiate good (suitable) test data from bad test (unsuitable) data, and so it should be able to detect good test data if they are generated. Artificial neural networks (ANNs) have been used in the past to handle several aspects of software testing. Experiments have been conducted to evaluate the effectiveness of generating test cases capable of exposing faults, Here prediction model is going to be developed using software failure data. As failure occurrences initiate the removal of faults, engineers reported failure times and time between failures (TBF). Both have been used to find the mean time between failures (MTBF), which is then used to investigate the reliability growth. In the present work feed forward neural network with back propagation learning algorithm, as an automated agent has been successfully implemented. The observations conclude that neural network model performs better in terms of less error in prediction as compared to existing analytical models and hence it is a better alternative to do software testing and quality assessment.**

Key Words: Software Testing, ANN, TBF, MTBF, Failure Rate

1. Introduction

Almost 50% of the software production development cost is expended in software testing. It consumes resources and adds nothing to the product in terms of functionality. Therefore, much effort has been spent in the development of automatic software testing tools in order to significantly reduce the cost of developing software [1]. Software testing is one of the main feasible methods to increase the confidence of the programmers in the correctness and reliability of software. The main goal of software testing is to increase one's confidence in the correctness of the program being tested. In order to test software, test data have to be generated and some test data are better at finding errors than others. Therefore, a search algorithm of a tool must decide where the best values (test data) lie and concentrate its search there.

Artificial neural networks (ANNs) have been used in the past to handle several aspects of software testing. Experiments have been conducted to evaluate the effectiveness of generating test cases capable of exposing faults, to use principle components analysis to find faults in a system, [5] to compare the capabilities of neural networks to other fault-exposing techniques,[4] [6] and to find faults in failure data. Hence prediction model is going to be developed using software failure data. As failure occurrences initiate the removal of faults, engineers reported failure times and time between failures (TBF). Both have been used to find the mean time between failures (MTBF), which is then used to investigate the reliability growth. Models that discuss the behaviour of MTBF are called SRMs.
For failure rate prediction model development a new application of neural networks as an automated "Agent" for a tested system has been presented. A multi-layer neural network is trained on the original software application by using randomly generated test data that conform to the specification. The neural network can be trained within a reasonable accuracy of the original program, though it may be unable to classify the test data 100 percent correctly. In effect, the trained neural network becomes a simulated model of the software application.

2 Literature Review

On reviewing literature, it is found that supervised, semisupervised and unsupervised learning approaches have been used for building a fault prediction models. Among these, supervised learning approach is widely used and found to be more useful FP module prediction if sufficient amount of fault data from previous releases are available. Generally, these models use software metrics of earlier software releases and fault data collected during testing phase. The supervised learning approaches cannot build powerful models with limited data. Unsupervised learning approaches such as clustering methods can be used in the absence of fault data. In most cases, software metrics and fault data obtained from a similar project or system release previously developed are used to train a software quality model. Below is given a brief review of the work done by many workers in the above filed of defect prediction with the objective of finding out future strategies in this field.
Norman Fenton et.al.(1999), have described a probabilistic model for software defect prediction. The aim here is to design a model which is a combination of diverse forms that may be often casual, with available evidence in development of software so that the work can be done in more natural and

efficient manner than it was previously done. Ahmet Okutan, et.al.(2012), proposed a novel method using Bayesian networks to explore the relationships among software metrics and defect proneness. Mrinal Singh Rawat et. al.(2012), identified causative factors which in turn suggest the remedies to improve software quality and productivity. They showed how the various defect prediction models are implemented resulting in reduced magnitude of defects. Manu Banga, (2013), here a new computational intelligence sequential hybrid architectures involving Genetic Programming (GP) and Group Method of Data Handling (GMDH) viz. GPGMDH have been discussed. Mohamad Mahdi Askari and Vahid Khatibi Bardsiri (2014) for the prediction of software defects used artificial neural network in order to better the generalization capability of the algorithm. Mrs.Agasta Adline, Ramachandran. M(2014) Predicting the fault-proneness of program modules when the fault labels for modules are unavailable is a challenging task frequently raised in the software industry. They attempted to predict the fault–proneness of a program modules when fault labels for modules are not present. K.Venkata Subba Reddy and Dr.B.Raveendra Babu (2013) we propose a software reliability growth model, which relatively early in the testing and debugging phase, provides accurate parameters estimation, gives a very good failure behavior prediction and enable software developers to predict when to conclude testing, release the software and avoid over testing in order to cut the cost during the development and the maintenance of the software.

3. Dataset Used

Failure data during system testing phase of various projects collected at Bell Tele-phone Laboratories, Cyber Security and Information Systems In-formation Analysis Centre(CSIAC) by John D. Musa are considered.

Five numbers of application software testing data set for demonstration of predictive performance and prediction accuracy as shown in Table 1 has been considered. 70% of each dataset is used for training the model and the rest failure data is used for validating the model. The datasets are downloaded from http://www.cse.cuhk.edu.hk/~lyu/book/reliability/data.html.

Table 1: Table of different software failure datasets used

Project Code	Project Name	No. of Failures	Development Phases
SYS1	Real Time Command & Command System	136	System Test Operations
SS3	Real Time Command & Command System	278	System Test Operations
CSR1	Real Time Command & Command System	397	System Test Operations
SS4	Real Time Command & Command System	197	System Test Operations
SYS3	Real Time Command & Command System	207	System Test Operations

4 Model Inputs and Structure

One among the foremost important steps within the development of any prediction model is that the choice of suitable input variables that may enable any classification model to successfully produce the specified results. In the present work for two types of prediction models have been developed.

MODEL-I

One is the development of Cumulative Time Between Failure (CTBF) prediction model, using FFNN algorithm, with Cumulative number of Failures is taken as input and Cumulative Time Between Failure (CTBF) is taken as output variable. Here five different prediction models have been developed using five different datasets. The nomenclature used are as follows;

Table 2: Structure of Forecasting model both for Model-1

Project Code	Model Nomenclature Used	Input Variables	Output Variable
SYS1	M1-SYS1	No. of Failures	CTBF
SS3	M1-SS3	No. of Failures	CTBF
CSR1	M1-CSR1	No. of Failures	CTBF
SS4	M1-SS4	No. of Failures	CTBF
SYS3	M1-SYS3	No. of Failures	CTBF

MODEL-II

Taking y(t1),y(t2),y(t3)…y(tk) as inputs to the neural network and Predicting y'(t(k+1)) as output(where y(t(k+1)) is taken as target value) is known as short term prediction or 1-step ahead prediction.

Thus the nomenclature used for different models are tabulated as below.

Table 3: Structure of Forecasting model both for Model-2

Project Code	Model Nomenclature Used	Input Variables	Output Variable

SYS1	M2-SYS1	CTBF(t-2), CTBF(t-1), CTBF(t)	CTBF(t+1)
SS3	M2-SS3	CTBF(t-2), CTBF(t-1), CTBF(t)	CTBF(t+1)
CSR1	M2-CSR1	CTBF(t-2), CTBF(t-1), CTBF(t)	CTBF(t+1)
SS4	M2-SS4	CTBF(t-2), CTBF(t-1), CTBF(t)	CTBF(t+1)
SYS3	M2-SYS3	CTBF(t-2), CTBF(t-1), CTBF(t)	CTBF(t+1)

5 Artificial Neural Network (ANN) Model Development

The various steps involved in the development of optimum prediction model are given below.

5.1 Model Selection

In the present work optimal network geometry was investigated, using trial and error approach as discussed earlier, in an attempt to create more optimum model. The number of hidden nodes was used to guide the trial and error search approach for the optimal geometry [8]; however since an optimum model has been sought, only models containing less than ten hidden nodes were considered. Thus to minimise the number of networks that required training and testing, ANN's containing 1 to 10 nodes were considered in order to narrow down the search. Once this range was determined, the trial and error approach was repeated, with the number of hidden nodes increasing in increment of one from minimum nodes onwards. Finally the optimum nodes were found for the best developed network and the networks on either side of the best developed network were also tested (if the model with the best generalisability contained 6 hidden nodes, networks with 5 and 7 hidden nodes were also tested). As mentioned in earlier, all the ANN models developed in this research contained a single hidden layer with sigmoid logistic activation function in the hidden nodes and output node. The number of nodes in the input layer has been kept fixed to three in all the six models considered [8][9].

5.2 *Training*

Training is the process by which the weights of an ANN are estimated, by using an iterative procedure to minimise a predetermined error, or objective function, such as the MSE. Therefore, ANN training is essentially a nonlinear least squares problem, which can be solved using standard nonlinear least squares methods. Here in this work Back-Propagation algorithm has been used for training the Feed Forward Neural Network architecture [10]. Once the training is complete, the weights are frozen. Training is the only time data is back propagated through the network. During recall, the network is strictly feed forward.
Initially the training data set was fed to the network for the development of the optimum model, keeping the initial weights as small and randomly distributed. The number of nodes was initially kept to one and was gradually increased in increment of one till ten nodes were reached and simultaneously monitoring the network performance. The learning rate was also initially kept to minimum and slowly increased. Also as discussed earlier in chapter 3, in order to resolve the contradiction of using slow or high learning rate (η), a momentum (φ) term was introduced which provided a built in inertia allowing a slow learning rate but faster learning. Thus various permutation and combinations of both these factors were used during the training process. The fixed period stops of 1000 cycles was used for training the network and the target error was set to stop during training when the average error reaches below 0.00999.
Model parameter values for Back Propagation Algorithm are given below in ***table 4***.

Table 4: Range of Model parameter values for Back Propagation Algorithm for all the models

Parameters	**Range of values**
Training Function	'trainlm'
Adaptation Learning Function	'learnGD'
Training mode	Supervise
Gradient mode	Jacobian
Performance Function	MSE, SSE, MAE
Transfer Function	For Hidden layer – tansigmoid For output layer - linear
Number of Hidden nodes	2-5
Learning Rate (η)	0.1 to 0.9
Momemtum (φ)	0.1 to 0.9
No. of epochs	100

Various network architectures were investigated in order to determine the optimal MLP architecture (i.e. the lowest mean square error and the optimum regression value) for the given combination of sixteen input variables. Different training algorithms were used with changes in the number of neurons in the hidden layers. In addition, the effect of transfer functions i.e. tangent sigmoid in the hidden layer were also investigated.

6. Results and Discussions:

(A) Comparison of ANN models

Both the models M1 and M2 as given above in ***table 2 and 3*** having one and three input variables respectively and one output variable with one hidden layer, were subjected to feed forward neural network with back propagation learning algorithm and sigmoid logistic activation function. Various permutations and combinations of network parameters viz. learning rate (η), momentum (φ) and number of hidden layer nodes were carried out for the development of the optimum network.

Model-1 (using only two variables)

Given below in Table 5 are the error values for best developed Model-1 obtained using 1-5-1 network configuration for both the datasets. Further Fig. 1 shows the comparative error plots for all the six dataset models. From the perusal of the figure it is seen that the network model 1-5-1 for datasets M1-SYS3 has least error as compared to all the other models.

Table 5: Performance Evaluation results of M1 Models

Model No.	MAE	MSE	SSE	Best Val. Perf.
M1-SYS1	0.0259	0.0013	0.0798	0.0019
M1-SS3	0.0233	0.000855	0.0907	0.000852
M1-CSR1	0.0191	0.000651	0.0976	0.000629
M1-SS4	0.0212	0.00075	0.0525	0.00075622
M1-SYS3	***0.0183***	***0.000542***	***0.0406***	***0.00081199***

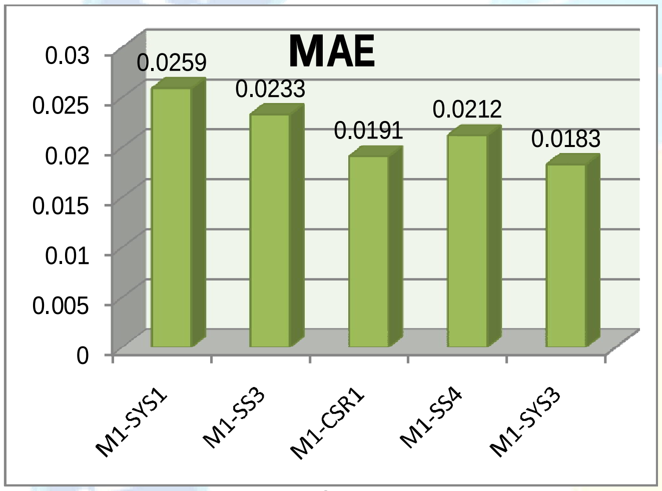

Fig. 1: MAE Plot for M-1 models

ModeL-2 (using four variables)

Next, Table4.5 depicts the error values for best developed Model-2 obtained using 3-5-1 network configuration for all the datasets. Fig. 2 shows the comparative error plot for these datasets. From the perusal of these figures it is seen that the network model 3-5-1 for datasets M2-CSR1 has least error as compared to other models.

Table 6: Error Values for Model-2

Model No.	MAE	MSE	SSE	Best Val. Perf.
M2-SYS1	0.0202	0.000904	0.0542	0.000482
M2-SS3	0.0204	0.000712	0.0755	0.0007873
M2-CSR1	***0.0128***	***0.000293***	***0.0439***	***0.0004124***
M2-SS4	0.022	0.000804	0.0603	0.0007702
M2-SYS3	0.0253	0.0012	0.0648	0.0021129

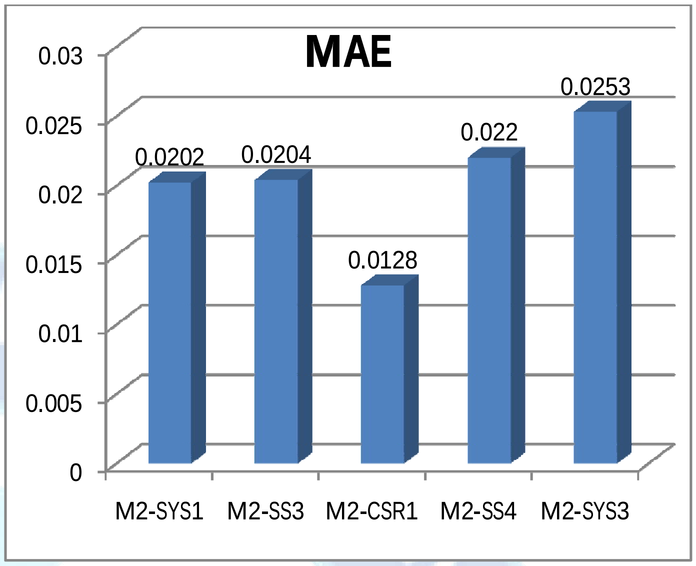

Fig. 2: MAE Plot for M-2 models

Comparative Analysis Of Model-1 & Model-2

A comparative error plots of both the models using one and three input variables is given in figures 3 to 4 below.

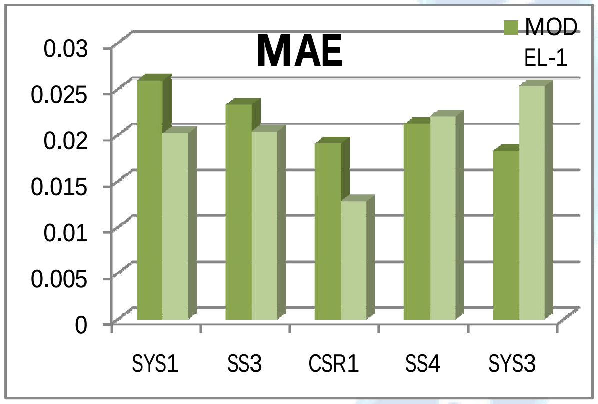

Fig. 3: MAE Plot for M-1 & M-2 models

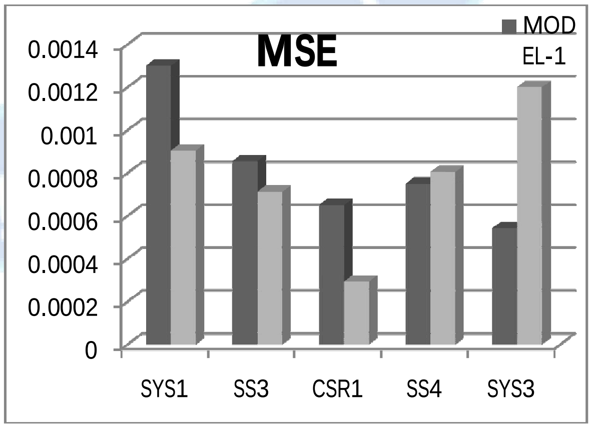

Fig. 4: MSE Plot for M-1 & M-2 models

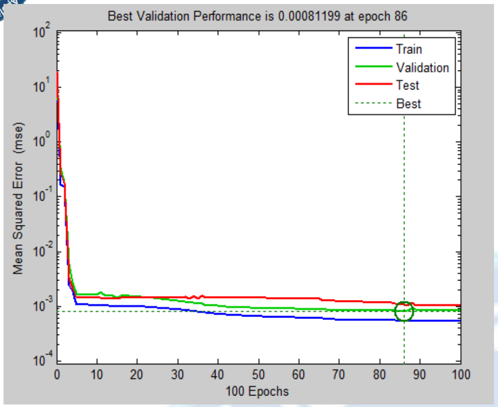

Fig. 5: Depicts the training of NN Model gauged by MSE for N=5 using Levenberg-Marquardt training algorithm with Gradient Descent with momentum as training function for M1-SYS3 Model

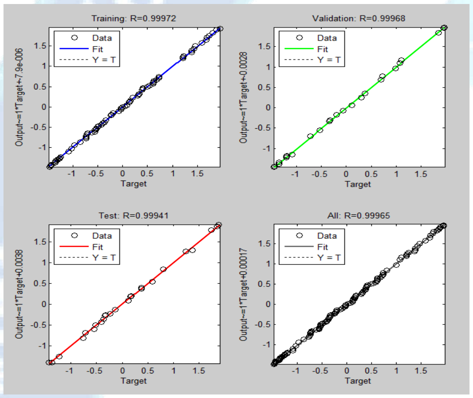

Fig. 6: Regression Plot during training for M1-SYS3 Model

M2-CSR1 (Using four variables)

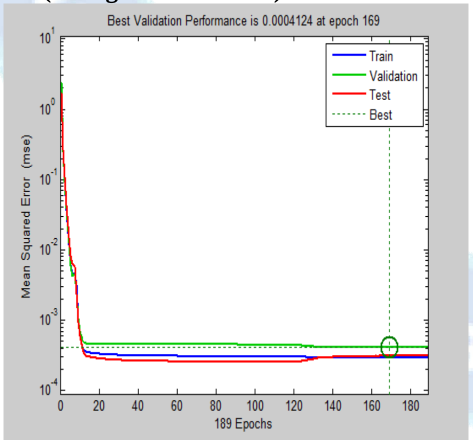

Fig. 7: Depicts the training of NN Model gauged by MSE for N=5 using Levenberg-Marquardt training algorithm with Gradient Descent with momentum as training function for for M2-CSR1 Model

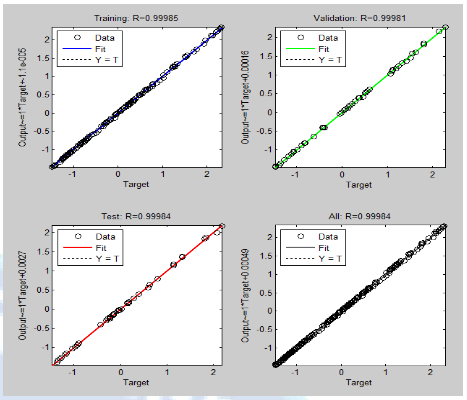

Fig. 8: Regression Plot during training for M2-CSR1 Model

Once the training process is complete and the results are obtained, then their accuracy is ascertained by using the testing data such that the predicted results are very close to the observed ones. Figure 5 and Fig. 8 shows the regression plot of training, testing and validation results for the best developed MI-SYS3 and M2-CSR1 using one and three input variables respectively, whereas on the other hand figures 6 and Fig. 7 demonstrates the plot of MSE for the bet developed ANN model, i.e. 1-5-1 and 3-5-1, using Levenberg Marquardt training algorithm for best developed M1-SYS3and M2-CSR1 Models. For finding the accuracy of the models during training, testing and validation stage, MSE criteria is used. As seen from figures 5 and 7 the performance in case of both the models improved even when the network error was low. It was noticed that in case of M1-SYS3 initially there was sharp fall in the nerwork error and at epoch 5 errors for training, testing and validation datasets got saturated and best validation result was obtained as 0.00081199 at epoch number 89, while in case of M2-CSR1 Model, though the initial fall in error values was the same but around epoch 10 it got saturated and the best validation result obtained was 0.0004124 at epoch number 169. The network model 1-5-1 and 3-5-1, which are the best can be considered as the best selection of network topology. This topology is able to maintain the number of layers, processing elements, generalization characteristics. This also can be seen that the training time elapsed has also been reduced due to less iteration required as each time.

7 Conclusions

In the present work feed forward neural network with back propagation learning algorithm, as an automated agent has been successfully implemented. The observations conclude that neural network model performs better in terms of less error in prediction as compared to existing analytical models and hence it is a better alternative to do software testing and quality assessment. As the connection weights are randomly initialized, thus the neural network gives different results for the same datasets and thus the performance of the network varies. The neural network is shown to be a promising method of testing a software application provided that the training data have a good coverage of the input range. The back propagation method of training the neural network is a relatively rigorous method capable of generalization, and one of its properties ensures that the network can be updated by learning new data. As the software that the network is trained to simulate is updated, so too can the trained neural network learn to classify the new data. Thus, the neural network is capable of learning new versions of evolving software.

References:

[1] Jogi John, (2011), "A Performance Based Study of Software Testing using Artificial Neural Network", International Journal of Logic Based Intelligent Systems, Vol. 1, No. 1, PP. 45-60.

[2] Voas JM, Miller KW. "Software Testability: The New Verification". *IEEE Software*; 1995.

Voas JM, McGraw G. Software Fault Injection; 1998.

[3] Khoshgoftaar TM, Allen EB, Hudepohl JP, Aud SJ. "Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System". *IEEE Transactions on Neural Networks* 1997; 8(4): 902-909.

[4] Khoshgoftaar TM, Szabo RM., "Using Neural Networks to Predict Software Faults During Testing". *IEEE Transactions on Reliability* 1996; 45(3): 456-462.

[5] Kirkland LV, Wright RG. "Using Neural Networks to Solve Testing Problems". *IEEE Aerospace and Electronics Systems Magazine* 1997; 12(8):36-40. http://www.cse.cuhk.edu.hk/~lyu/book/reliability/data.html.

[6] Kingston, G.B., (2006), "Bayesian Artificial Neural Network in Water Resources Engineering", P.hd. Faculty of Engineering, Computer and Mathematical Science. The University of Adelaide.

[7] Eklund, J. And KapetanioS, G., (2008), "A Review of Forecasting Techniques for Large Datasets". Working paper no. 625.

[8] Maier, H.R., (1995), "A review of Artificial Neural Network.", Research Report no. R131. Dept. of Civil and Env. Engg. The University of Adelaide.

[9] Zhang, G. P., (2003), Time series forecasting using a hybrid ARIMA and neural network model, Neuro computing, 50, pp 159–175.