

International Conference on Information and Communication Technologies (ICICT 2014)

## Design Of Software Fault Prediction Model Using BR Technique

Rohit Mahajan<sup>a,\*</sup>, Sunil Kumar Gupta<sup>b</sup>, Rajeev Kumar Bedi<sup>c</sup>

<sup>a</sup>PTU, Golden College of Engineering & Technology, Gurdaspur- 143521, Punjab, INDIA

<sup>b,c</sup>PTU, Beant College of Engg & Technology, Gurdaspur - 143521, Punjab, INDIA

---

### Abstract

During the previous years, the demand for producing the quality of software has been quickly increased. In this paper, Bayesian Regularization (BR) technique has been used for finding the software faults before the testing process. This technique helps us to reduce the cost of software testing which reduces the cost of the software project. The basic purpose of BR technique is to minimize a combination of squared errors and weights, and then determine the correct combination so as to produce an efficient network. BR Technique algorithm based neural network tool is used for finding the results on the given public dataset. The accuracy of BR algorithm based neural network has been compared with Levenberg-Marquardt (LM) algorithm and Back Propagation (BPA) algorithm for finding the software defects. Our results signify that the software fault prediction model using BR technique provide better accuracy than Levenberg-Marquardt (LM) algorithm and Back Propagation (BPA) algorithm.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Peer-review under responsibility of organizing committee of the International Conference on Information and Communication Technologies (ICICT 2014)

**Keywords:** Back Propagation (BPA) algorithm; Bayesian Regularization (BR) algorithm; Levenberg-Marquardt (LM) algorithm; Neural network; public dataset;

---

---

\* Rohit Mahajan. Tel.: +91- 9781159022  
E-mail address: rohitcse006@gmail.com

## 1. Introduction

Different Software metrics which are used to find the software faults before the process of testing are class level, method-level metrics etc. Methods used for finding the software faults are machine learning, statistical method and expert estimation<sup>2</sup>. But machine learning method is best method for finding the software faults because all the work is done by neural network based machine. Neural network is a machine learning approach and made up of number of artificial neurons. Each neuron in Neural Network receives a number of inputs and produces only one output<sup>2</sup>. The concept of hidden layer is also used to train any neural network. BR technique minimizes a combination of squared errors and weights, and then determines the correct combination so as to produce an efficient network. Bayesian Regularization (BR) technique is machine learning technique. The BR technique has already been used in cost estimation in the field of software Engineering but has never been explored in software fault prediction. The main advantage of the BR technique is that it consumes less memory space and provides better accuracy than all the previous techniques used in the software fault prediction model. When we train any neural network, we can measure the performance and regression of the neural network. Training of the neural network stops when any of these situations occurs:

- The maximum number of epochs is reached.
- The maximum amount of time is exceeded

Defects in system software lead to foremost difficulty in the software. A lot of software systems are sent to the clients with unnecessary defects. Testing is one of the most important approaches for finding the defect prone parts of the system. Software quality can be measured with various attributes like fault thickness, normalized rework, reusability, portability and maintainability etc.<sup>2</sup>. In this paper, we study the importance of Bayesian Regularization (BR) technique and also give the comparison of Bayesian Regularization (BR) technique with Levenberg-Marquardt (LM) and Back Propagation Algorithm (BPA) technique.

## 2. Review Of Literature

Norman E.Fenton<sup>5</sup> describes that software metrics and statistical models have been developed to find the number of defects in the software system and the majority of the prediction models use size and complexity metrics to find faults. To find a single complexity metrics, a large complex multivariate statistical model has been introduced. The limitations are that by using size and complexity metrics, accessible models cannot find the faults successfully.

Atcharamahaweerawat et al. describes that software fault prediction technique is the superlative approach for finding the software faults to enhance the quality and reliability of the software<sup>10</sup>. They used Method level metrics. The concept of neural networks is importantly used. Neural networks provide an important technique called Radial Basis Function (RBF)<sup>10</sup>. The main function of RBF is to find the faults in the software and provide better accuracy. The object Oriented software systems are used for predicting the number of faults in the software<sup>8,10</sup>. Inheritance and Polymorphism are the important features of Object Oriented systems. For finding the software faults, a large amount of data is required. Two important networks are used: -Multilayer Perceptron (MLP) is used for ruling the defective modules whereas Radial Basis functions Network are used to classify the defects according to a number of different types of faults<sup>8</sup>. Xing et al.<sup>16</sup> describes the importance of Support Vector Machine (SVM) model. This SVM model is used when only a little amount of data is obtainable. Data categorization is a significant use of SVM technique. SVM provides better accuracy than other techniques for the prediction of quality of the software but in public datasets, the performance of SVM is poor.

The early lifecycle metrics play significant role in the software project management<sup>7</sup>. Early lifecycle metrics can be used to identify faulty modules. Method level metrics are widely used for software fault prediction. The authors used three NASA projects are: PC1, CM1 and JM1. After comparison of these different projects, they concluded that the requirement metrics have significant role in software fault prediction. In another paper<sup>4</sup>, the authors illustrated the potential of SVM for finding the defects in the software and compared the performance with different machine learning models. The models developed by them with the help of SVM, provide better accuracy than the other models. In the context of four NASA datasets, they calculate the ability of SVM in predicting defect-prone software modules and contrast the performance of the software fault prediction against eight statistical and machine learning models. Gondra et al.<sup>6</sup> used Artificial Neural Networks (ANN's) and Support Vector Machines (SVM's) to reduce the price and progress for the effectiveness of the software testing process. Data is taken from the public

dataset that is freely available from the Promise repository. Researchers use different software metrics like Lines of Code (LOC), McCabe (1976), and Halstead (1977) metrics.

Jun Zheng<sup>17</sup> described that the software fault prediction model can be built with the help of threshold-moving technique. The motive of the software developer is to develop the better quality software on time and inside the financial plan. Software fault prediction model classifies the modules into two classes: faulty modules and non – faulty modules. They discussed the use of different cost sensitive boosting algorithms for software fault prediction. The accuracy of the cost sensitive boosting algorithms is quite good than the other algorithms.

R.Shatnawi<sup>13</sup> states that the majority of the modules for finding the prediction performance are correct whereas some modules are defective. They applied technique to find the number of faults in the particular module. This technique is called Eclipse. This technique works well on real world objects called Object Oriented systems. In this Object Oriented System, they used the existing defected data for eliminating the defective modules<sup>13</sup>.

Singh et al.<sup>14</sup> describes that Levenberg- Marquardt (LM) algorithm based neural network tool is used for prediction of software defects at an early stage of SDLC. They used the class level metrics. The Defected data are collected from the NASA promise repository. LM Algorithm is based upon machine learning approach. The accuracy of LM Algorithm based neural network is better than the Polynomial function -based neural network for detection of software defects.

### 3. Proposed Methodology

In our proposed methodology, we used Ant 1.7 dataset<sup>14</sup> and this dataset consists of defected data which are coming from the PROMISE (Predict or Models in Software Engineering) repository of empirical software engineering data<sup>14</sup>. In this promise data repository, defected data is freely available and this type of dataset is called public dataset. Our main objective is to find the accuracy of the proposed Neural Network (NN) classifier i.e Bayesian Regularization (BR) technique and compare the accuracy of Bayesian Regularization (BR) technique with Levenberg-Marquardt (LM) algorithm and Back propagation Algorithm (BPA) algorithm.

In this experiment, our defected data is divided into 2 parts: Training and Testing. 85% of data is used for training and 15 % is used for testing. For the purpose of testing, 15% of the data is selected randomly using the random number generator formula and the rest of the data is used for training purposes.

Random Generator formula

$$Arr = \text{ceil} (1+ (745-1) * \text{rand} (100, 1)) \quad (1)$$

Total number of samples in a dataset is 745.650 samples are used for training purpose.100 samples are used for testing purpose.

#### 3.1 Public Dataset

Public dataset is that dataset which are frequently situated in Promise repositories and they are distributed freely. In this experiment, Ant-1.7 Public dataset is that dataset which comes from Promise repository (<http://promisedata.googlecode.com>). Ant 1.7 dataset uses class level metrics.

Table 1.below Show the different types of Inputs used in the Ant- 1.7 dataset

| S.No | Attributes / Inputs | Explanation                     | Suggested By                       |
|------|---------------------|---------------------------------|------------------------------------|
| 1.   | WMC                 | Weighted methods per class      | Chidamber and Kemerer <sup>3</sup> |
| 2.   | DIT                 | Depth of Inheritance Tree       | Chidamber and Kemerer <sup>3</sup> |
| 3.   | NOC                 | Number of Children              | Chidamber and Kemerer <sup>3</sup> |
| 4.   | CBO                 | Coupling between object classes | Chidamber and Kemerer <sup>3</sup> |
| 5.   | RFC                 | Response for a Class            | Chidamber and Kemerer <sup>3</sup> |
| 6.   | LCOM                | Lack of cohesion in methods     | Chidamber and Kemerer <sup>3</sup> |
| 7.   | LCOM3               | Lack of cohesion in methods     | Henderson-Sellers <sup>12</sup>    |
| 8.   | NPM                 | Number of Public Methods        | Bainsy and Davis <sup>1</sup>      |
| 9.   | DAM                 | Data Access Metric              | Bainsy and Davis <sup>1</sup>      |
| 10.  | MOA                 | Measure of Aggregation          | Bainsy and Davis <sup>1</sup>      |

|     |         |                                   |                               |
|-----|---------|-----------------------------------|-------------------------------|
| 11. | MFA     | Measure of Functional Abstraction | Bainsy and Davis <sup>1</sup> |
| 12. | CAM     | Cohesion Among Methods of Class   | Bainsy and Davis <sup>1</sup> |
| 13. | IC      | Inheritance Coupling              | Tang et al. <sup>15</sup>     |
| 14. | CBM     | Coupling Between Methods          | Tang et al. <sup>15</sup>     |
| 15. | AMC     | Average Method Complexity         | Tang et al. <sup>15</sup>     |
| 16. | Ca      | Afferent couplings                | Martin <sup>9</sup>           |
| 17. | Ce      | Efferent couplings                | Martin <sup>9</sup>           |
| 18. | CC      | Cyclomatic complexity             | McCabe <sup>11</sup>          |
| 19. | Max(CC) | The greatest value of CC          | McCabe <sup>11</sup>          |
| 20. | Avg(CC) | The arithmetic mean of the CC     | McCabe <sup>11</sup>          |

#### 4. Implementation of proposed technique

##### 4.1 Creation of GUI

(GUI) is developed with the help of Matrix Laboratory (MATLAB R2011a). For the construction of GUI, Inputs are taken from the above table and the parameters are taken from class level metrics. Parameters are taken from Chidamber and Kemerer (CK) metrics, Henderson-Sellers, Bainsy and Davis, Tang et al., Martin and McCabe Metrics. In this study, software fault Prediction Model is developed using Bayesian Regularization (BR) algorithm.

##### 4.2 Assembling the Data

Record for classification problems consists of textual/ non-numeric data. But with textual data, the training of Neural Network is not possible. Therefore, we need a translator that can convert non-numeric data into numeric form. Different translation techniques are available for training of neural network. But unary encoding is the best technique for converting non-numeric data into numeric form and to train the neural network also.

##### 4.3 Constructing the Neural Network classifier

To train the neural network using BR technique, we can use function trainbr. This function is a network training function that updates the weight and bias values according to Levenberg-Marquardt (LM) optimization. Twister seed is used to avoid the randomness. The concept of hidden layer is also used. In this study, three hidden layer (13, 13, 13) feed forward network is created with 13 neurons in every hidden layers. After that, our neural network is ready to be trained.

##### 4.4 Testing the BR classifier

Our next step is to test our trained neural network with different testing samples. First of all, training of neural network is required and after that we can find the predicted output with 100 testing samples.

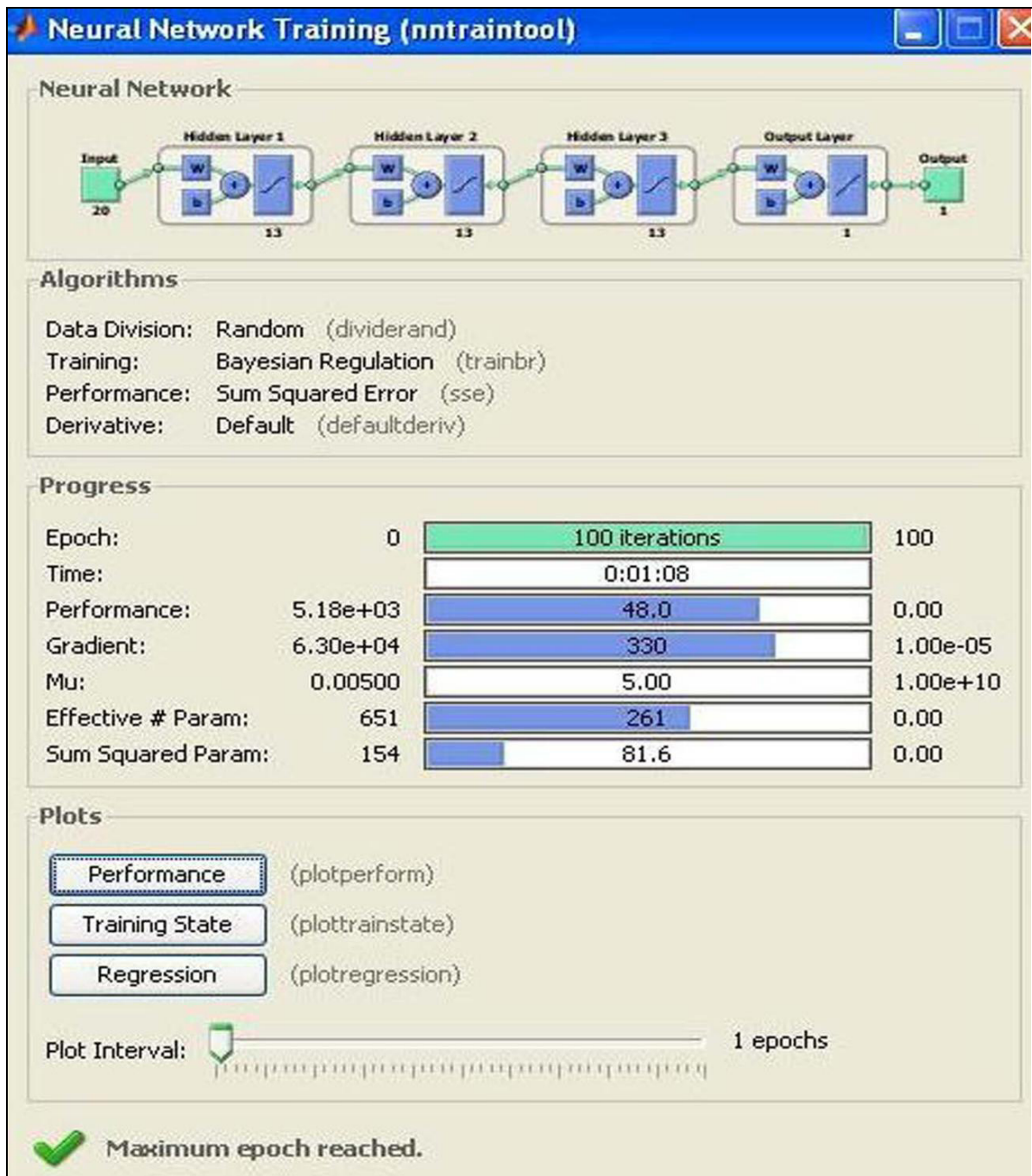


Fig. 1. Training of neural network

From the above figure, Bayesian Regularization based neural network is trained in which 1 input layer, 3 hidden layers and 1 output layer are used. BR algorithm has a maximum number of epochs i.e. is 100. BR Algorithm may be halt, when the maximum number of epochs crosses the limit of 100.

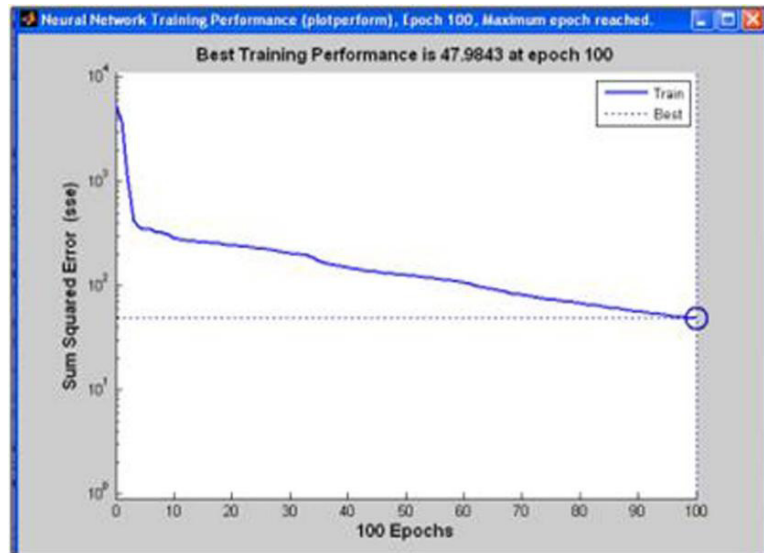


Fig.2. performance of BR neural network

In Figure2, the performance plot shows the value of the performance function versus the iteration number. The best training performance is 47.9843 at epoch 100. Dotted line shows that the performance is best and straight line shows the training performance. From the above figure, it has been observed that the training performance meets at the 100 epoch which provide the best training function. The function 'trainbr' is used to plot the training performance. Performance is measured in terms of sum squared error (sse). Sum squared error is a performance function. It measures the performance according to the sum of squared errors. Performance of sum squared error is calculated by the following formula:

$$\text{Per} = \text{sse}(\text{net}, \text{t}, \text{y}, \text{ew}) \quad ; \quad (2)$$

Where, net stands for neuralnetwork, t stands for Matrix or cell array of target vectors, y stands for Matrix or cell array of output vectors, ew stands for error weights.

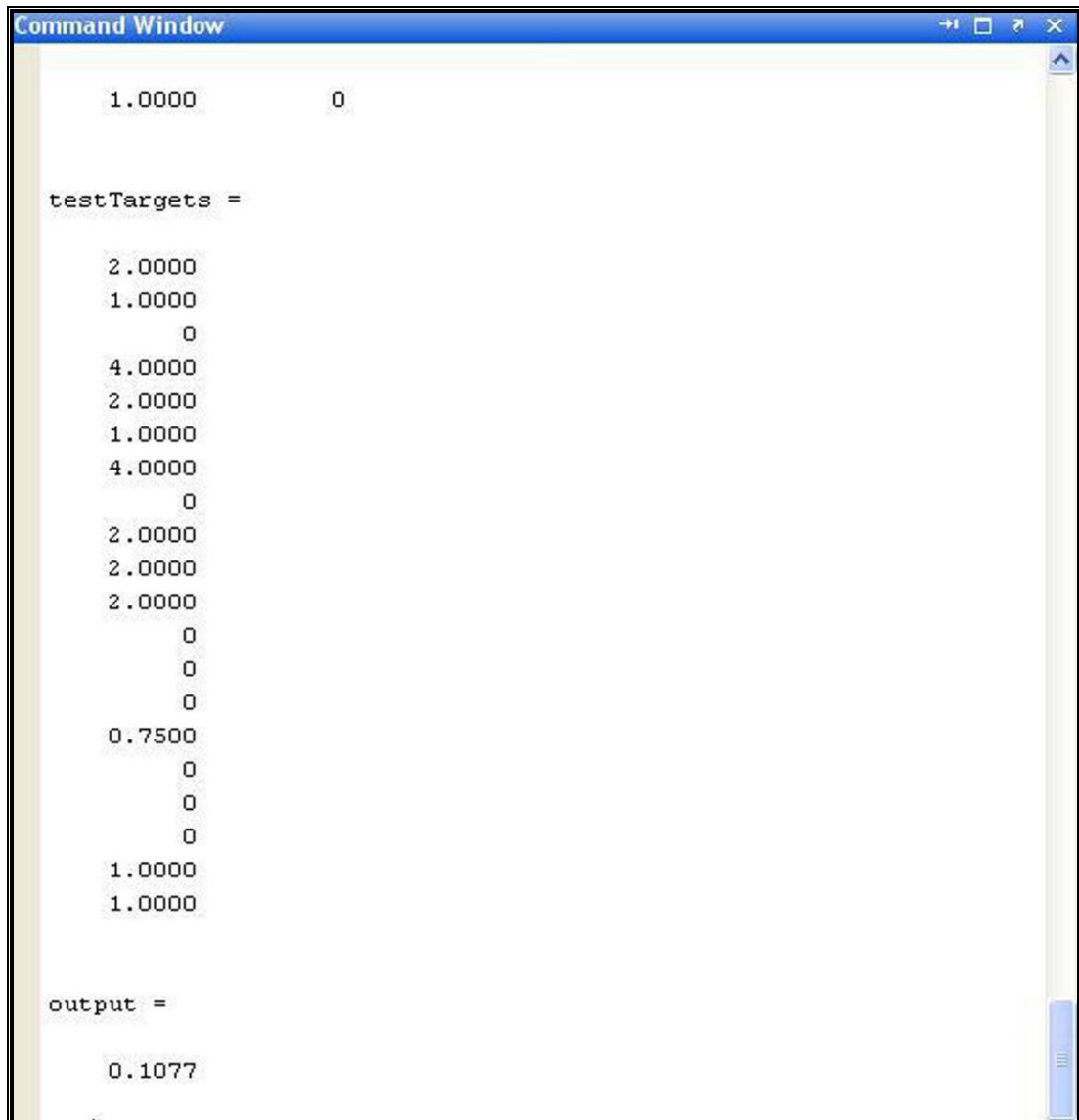
| Software Defect Analysis and Prediction Model using Bayesian Regularization Algorithm |   |                                         |      |
|---------------------------------------------------------------------------------------|---|-----------------------------------------|------|
| Weighted Method Per class (VMC)                                                       | 2 | Lines of code (LOC)                     | 2    |
| Depth of Inheritance tree (DIT)                                                       | 1 | Data Access Metric (DAM)                | 0    |
| Number of Children (NOC)                                                              | 0 | Measure of Aggregation (MOA)            | 0    |
| Coupling between Object classes (CBO)                                                 | 4 | Measure of Functional Abstraction (MFA) | 0    |
| Response for a Class (RFC)                                                            | 2 | Cohesion Among Methods of Class (CAM)   | 0.75 |
| Lack of Cohesion in Methods (LCOM)                                                    | 1 | Inheritance Coupling (IC)               | 0    |
| Afferent Couplings (ca)                                                               | 4 | Coupling Between Methods (CBM)          | 0    |
| Efferent Couplings (ce)                                                               | 0 | Average Method Complexity (AMC)         | 0    |
| Number of Public Methods (NPM)                                                        | 2 | The greatest value of CC (max_cc)       | 1    |
| Lack of cohesion in methods (LCOM3)                                                   | 2 | The arithmetic mean of the CC (avg_cc)  | 1    |

Buttons: Train N.N, Predict, Exit

Fig.3. inputs to neural network



In Fig 3, there are 20 inputs and only one output and three buttons (Train N.N, Predict and Exit button). The function of Train Neural network button is used to train the neural network using training data from the Ant 1.7 dataset. The function of predict button is to calculate the desired output of different testing samples. Exit button is used to close the GUI.



```
Command Window

1.0000      0

testTargets =

2.0000
1.0000
      0
4.0000
2.0000
1.0000
4.0000
      0
2.0000
2.0000
2.0000
      0
      0
      0
0.7500
      0
      0
      0
1.0000
1.0000

output =

0.1077
```

Fig.4. prediction of output

In Fig.4, for 20 input values, only one output is calculated. In the same manner, other outputs are predicted using tested data.

## 5. Results and Comparison

Our proposed Software fault prediction model is implemented in MATLAB 2011. The predicted output of BPA, LM and BR techniques are given below in table 2:

Table2. Predicted Output

| S.No | Record No | Desired Output | Predicted Output using BPA | Predicted Output using LM | Predicted Output using BR | S.No         | Record No      | Desired Output | Predicted Output using BPA | Predicted Output using LM | Predicted Output using BR |
|------|-----------|----------------|----------------------------|---------------------------|---------------------------|--------------|----------------|----------------|----------------------------|---------------------------|---------------------------|
| 1    | 608       | 0              | 0.0609                     | -0.18                     | 0.1077                    | 51           | 207            | 0              | 0.1325                     | 0.59                      | -0.0048                   |
| 2    | 675       | 0              | 0.1221                     | -0.19                     | -1.9767                   | 52           | 507            | 0              | 0.0465                     | -0.01                     | 0.1138                    |
| 3    | 96        | 0              | -0.1824                    | 0.01                      | 0.1098                    | 53           | 331            | 0              | 0.1134                     | 0.03                      | -0.079                    |
| 4    | 681       | 0              | 0.0759                     | 0.02                      | 0.0551                    | 54           | 122            | 0              | 0.1821                     | 0.04                      | -0.2524                   |
| 5    | 472       | 0              | 0.1943                     | -0.03                     | 0.0533                    | 55           | 90             | 0              | 0.0639                     | -0.02                     | 0.0845                    |
| 6    | 74        | 0              | 0.2227                     | -0.02                     | -0.1966                   | 56           | 372            | 4              | 0.0859                     | -7.22                     | 0.6419                    |
| 7    | 209       | 0              | 0.0484                     | 0.01                      | 0.1432                    | 57           | 716            | 0              | 0.1948                     | -0.65                     | 0.0102                    |
| 8    | 408       | 0              | 0.061                      | -0.02                     | 0.0773                    | 58           | 255            | 0              | 0.2374                     | 3.92                      | 0.01                      |
| 9    | 64        | 6              | 2.6196                     | 1.52                      | 4.9669                    | 59           | 437            | 1              | 0.8988                     | 0.05                      | 0.5961                    |
| 10   | 719       | 0              | 0.2248                     | 0                         | 0.1168                    | 60           | 168            | 0              | 1.2915                     | 10.42                     | 1.8918                    |
| 11   | 119       | 0              | 0.2364                     | -0.01                     | 0.0842                    | 61           | 560            | 6              | 2.6501                     | 0.79                      | -7.2142                   |
| 12   | 724       | 0              | 0.0407                     | -0.06                     | 0.0735                    | 62           | 191            | 0              | 0.0164                     | -0.22                     | -0.0635                   |
| 13   | 714       | 0              | -0.0119                    | 0.01                      | 0.0808                    | 63           | 378            | 0              | 0.1419                     | 0.02                      | 0.5942                    |
| 14   | 363       | 0              | 0.0461                     | 0                         | 0.1103                    | 64           | 522            | 3              | 0.7388                     | -0.54                     | 2.0016                    |
| 15   | 597       | 1              | 0.5935                     | 1.93                      | -6.4653                   | 65           | 664            | 1              | 0.2390                     | -0.01                     | -0.1004                   |
| 16   | 107       | 0              | -1.0851                    | 14.77                     | -8.4357                   | 66           | 5              | 1              | 0.1868                     | -0.01                     | -0.2468                   |
| 17   | 315       | 0              | 0.2739                     | -1.48                     | 0.0220                    | 67           | 409            | 0              | 0.0647                     | -0.22                     | 0.1563                    |
| 18   | 683       | 1              | 0.5499                     | -0.25                     | 0.048                     | 68           | 105            | 0              | -0.0849                    | -0.13                     | 0.2117                    |
| 19   | 591       | 0              | 0.0417                     | 0.06                      | 0.0488                    | 69           | 113            | 0              | -0.2124                    | -0.03                     | -0.4047                   |
| 20   | 715       | 0              | 0.0504                     | 0.01                      | 0.0321                    | 70           | 193            | 3              | 2.3536                     | 1.99                      | 6.2149                    |
| 21   | 489       | 1              | 0.3433                     | 0.05                      | 0.5315                    | 71           | 627            | 0              | 0.2125                     | -0.03                     | 0.4856                    |
| 22   | 28        | 0              | 0.9461                     | -0.41                     | -0.7174                   | 72           | 110            | 0              | 0.0985                     | 0.12                      | -0.0014                   |
| 23   | 633       | 0              | 0.2525                     | 0.03                      | 0.5637                    | 73           | 607            | 0              | 0.1489                     | -0.01                     | -0.1212                   |
| 24   | 696       | 0              | 0.0484                     | 0.01                      | 0.1432                    | 74           | 183            | 1              | 0.0871                     | 2.22                      | 0.1496                    |
| 25   | 506       | 0              | 0.2732                     | -0.52                     | -0.0719                   | 75           | 693            | 1              | 0.967                      | 2.93                      | 0.4142                    |
| 26   | 565       | 0              | 0.0611                     | -0.03                     | -0.4186                   | 76           | 262            | 4              | -0.0138                    | 9.5                       | -0.0958                   |
| 27   | 554       | 0              | 0.1075                     | 4.94                      | 0.6226                    | 77           | 148            | 0              | 0.2206                     | 0.14                      | 0.3305                    |
| 28   | 293       | 1              | -0.2974                    | 3.43                      | -5.2029                   | 78           | 188            | 0              | 0.1041                     | 0.02                      | 0.0245                    |
| 29   | 558       | 0              | -0.1437                    | -0.01                     | 0.2459                    | 79           | 460            | 0              | 1.3298                     | 2.77                      | 1.3246                    |
| 30   | 129       | 0              | 0.1365                     | 0                         | -0.0463                   | 80           | 354            | 1              | 0.7091                     | -0.75                     | 2.9705                    |
| 31   | 527       | 0              | 0.1078                     | 0.02                      | -0.0367                   | 81           | 263            | 0              | 0.1261                     | 0                         | -0.115                    |
| 32   | 25        | 0              | 0.035                      | 0.02                      | 0.2319                    | 82           | 620            | 0              | 0.5496                     | 0.24                      | 0.4236                    |
| 33   | 208       | 3              | 1.4125                     | -4.62                     | 9.6417                    | 83           | 300            | 0              | 0.3154                     | 0.01                      | -0.3171                   |
| 34   | 36        | 0              | 0.1592                     | -0.24                     | 0.1079                    | 84           | 410            | 0              | 0.1137                     | -0.01                     | 0.1025                    |
| 35   | 395       | 1              | -0.0432                    | -0.04                     | 1.6488                    | 85           | 684            | 0              | 0.1141                     | 0.01                      | 0.1257                    |
| 36   | 614       | 0              | 0.0488                     | 0.01                      | 0.1469                    | 86           | 214            | 1              | 6.7042                     | -0.01                     | 0.5376                    |
| 37   | 518       | 0              | 0.2655                     | -6.35                     | 0.2403                    | 87           | 180            | 0              | 0.0168                     | 0.07                      | 0.0545                    |
| 38   | 237       | 0              | 0.2234                     | 0.06                      | -0.6041                   | 88           | 562            | 1              | 0.2528                     | -0.02                     | 2.0592                    |
| 39   | 708       | 0              | 0.2957                     | 1.21                      | 0.7068                    | 89           | 93             | 0              | 0.0769                     | 0.03                      | 0.0156                    |
| 40   | 27        | 0              | 0.1562                     | -0.02                     | -0.1931                   | 90           | 424            | 0              | 0.3896                     | -7.79                     | -0.0950                   |
| 41   | 328       | 2              | -0.1902                    | -0.02                     | 0.1149                    | 91           | 58             | 3              | 0.7364                     | -2.47                     | -0.0950                   |
| 42   | 285       | 0              | 0.0793                     | 0                         | 0.4                       | 92           | 42             | 0              | 0.514                      | 0.06                      | 0.1179                    |
| 43   | 571       | 0              | 0.8971                     | 2.37                      | 0.3802                    | 93           | 396            | 0              | 0.0603                     | -0.13                     | 0.1014                    |
| 44   | 593       | 1              | 10.6657                    | 14.82                     | 11.4321                   | 94           | 581            | 0              | 0.1013                     | -0.03                     | 0.0338                    |
| 45   | 141       | 1              | 1.0461                     | 5.84                      | 2.2118                    | 95           | 312            | 1              | 7.5987                     | -0.56                     | 6.0763                    |
| 46   | 366       | 0              | 0.2917                     | -0.01                     | -1.8301                   | 96           | 98             | 0              | 1.0343                     | 6.37                      | 1.0083                    |
| 47   | 333       | 4              | 1.2594                     | 0.76                      | 0.3203                    | 97           | 425            | 0              | 0.0435                     | -0.69                     | -0.0465                   |
| 48   | 482       | 0              | -0.0628                    | -0.02                     | 0.1639                    | 98           | 351            | 0              | 0.1991                     | 0.02                      | -0.0429                   |
| 49   | 529       | 0              | 1.6884                     | 0.08                      | 2.1614                    | 99           | 10             | 0              | 0.5371                     | -0.02                     | -0.1406                   |
| 50   | 563       | 0              | 0.4032                     | -0.02                     | 0.7564                    | 100          | 252            | 0              | 0.1489                     | 2.21                      | 0.1804                    |
|      |           |                |                            |                           |                           | <b>Total</b> | <b>Average</b> | <b>0.54</b>    | <b>0.6296</b>              | <b>0.6043</b>             | <b>0.5808</b>             |



From the above table, It has been established that the result of the predicted output of BR, LM and BPA is calculated by using testing data from the Ant 1.7 dataset. Desired output is that output which is given in the Ant 1.7 dataset. As a result, the average of actual outputs is 0.54 and average Predicted O/P using BPA, average Predicted O/P using LM and average Predicted O/P using BR is 0.6296, 0.6043 and 0.5808 respectively. General formula, for finding the percentage error and percentage accuracy of BR Algorithm, LM algorithm and BPA algorithm are given below:

Percentage Error =  $|\text{average predicted outputs} - \text{average actual outputs}| / \text{average actual outputs} * 100$

(i) Accuracy of BPA Algorithm:

Percentage Error =  $|0.6296 - 0.54| / 0.54 * 100 = 16.59\%$

Percentage Accuracy =  $100 - 16.59\% = 83.41\%$  (3)

(ii) Accuracy of LM algorithm:

Percentage Error =  $|0.6043 - 0.54| / 0.54 * 100 = 11.91\%$

Percentage Accuracy =  $100 - 11.91\% = 88.09\%$  (4)

(iii) Accuracy of BR Algorithm:

Percentage Error =  $|0.5808 - 0.54| / 0.54 * 100 = 7.56\%$

Percentage Accuracy =  $100 - 7.56\% = 92.44\%$  (5)

From the above calculations, it has been found that Back propagation (BPA) Algorithm based neural network for finding the software defects before testing provides the accuracy 83.41%. Levenberg-Marquardt (LM) algorithm based neural network for predicting the software defects before testing provides the accuracy 88.09% and Bayesian Regularization (BR) algorithm based neural network for finding the software defects before testing provides the accuracy 92.44%. So it has been observed that designing of software fault prediction model using Bayesian Regularization technique provides the highest accuracy than all the previous techniques.

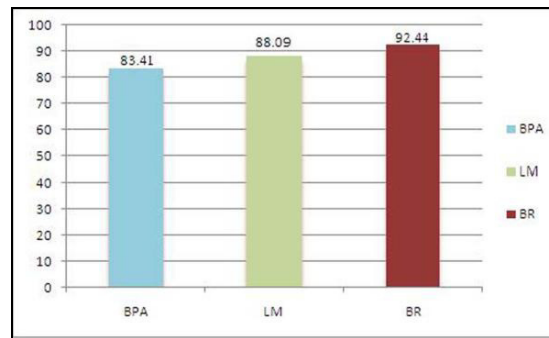


Fig.5.comparison of the Accuracy of BR with BPA and LM

## 6. Conclusion and Future Scope

For predicting the software defects before the process of testing, it is required to have a superior prediction system. Our proposed model uses Object Oriented metrics to predict the faults during design phase. Bayesian Regularization (BR) algorithm is proved to be the best algorithm as compared to the other algorithms like Levenberg-Marquardt (LM) algorithm and Back propagation (BPA) algorithm. With the help of neural network technique and Bayesian Regularization (BR) algorithm, the accuracy of our proposed system is better than Back propagation (BPA) algorithm and Levenberg-Marquardt (LM) algorithm. Neural Network is based upon machine learning approach. As a result, it is found that machine learning models are importantly used and provide superior results. In future, some other training algorithms may be tried to raise the accuracy level for finding the software faults at an early stage of software development life cycle. By using class level metrics, more studies can be conducted on fault prediction models.

## 7. References

1. Bansiya, J., & Davis, C. G. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software Engineering*, Vol. 28 (1), 2002; 4-17.
2. Catal, C., & Diri, B. A systematic review of software fault prediction studies. *J. Expert Systems with Applications*, Vol. 36 (4), 2009; 7346-7354.
3. Chidamber, S. R. & Kemerer, C. F. A metrics suite for object oriented design. *IEEE transaction on software engineering*, Vol. 20(6), 1994; 476-493.
4. Elish, K. O., & Elish, M. O. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, ACM, Vol. 81(5), 2008; 649-660.
5. Fenton, N. E., & Neil, M., A critique of software defect prediction models. *IEEE Transactions on software Engineering*, Vol. 25(5); 675-689.
6. Gondra, I. Applying machine learning to software fault-proneness prediction. *ACM Journal of Systems and Software*, Vol. 81(2), 2008; 186-195.
7. Jiang, Y., Cukic, B., & Menzies, T. Fault prediction using early lifecycle data. Published in *18th IEEE international symposium on software reliability*, 2007; 237-246.
8. Mahaweerawat, A., Sophatsathit, P., Lursinsap, C., & Musilek, P. Fault prediction in object-oriented software using neural network techniques. Advanced Virtual and Intelligent Computing Center (A VIC), Department of Mathematics, Faculty of Science, Chulalongkorn University, Bangkok, Thailand, 2004; 1-8.
9. Martin, R. OO design quality metrics-An analysis of dependencies. *Workshop on Pragmatic and Theoretical Directions in Object -Oriented Software Metrics*, 1994; 1-8.
10. Mahaweerawat A., Sophasathit P., Lursinsap, C. Software fault prediction using fuzzy clustering and radial basis function network .In proceedings of *International conference on intelligent technologies*, 2002; 304-313.
11. McCabe, T. J. A complexity measure. *IEEE Transactions on software Engg.*, Vol. 2(4), 1976; 308-320.
12. Sellers, B. H. Object-Oriented Metrics. Measures of Complexity. Prentice Hall, 1996.
13. Shatnawi, R. Improving software fault-prediction for imbalanced data. *IEEE Proceedings of International Conference on Innovations in Information Technology*, 2012; 54-59.
14. Singh, M., & Salaria, D.S. Software Defect Prediction Tool based on Neural Network. *International Journal of Computer Applications*, Vol. 70(22), 2013; 22-27.
15. Tang, M. H., Kao, M. H., & Chen, M. H. An empirical study on object-oriented metrics. *IEEE Engineering*; 2005.
16. Xing, F., Guo, P., & Lyu, M. R. A novel method for early software quality prediction based on support vector machine. *16<sup>th</sup> IEEE international symposium on Software Reliability*, Vol. 37(6), 4537-4543.
17. Zheng, J. Cost-sensitive boosting neural networks for software defect prediction. *J. Expert Systems with Applications: An international Journal*, ACM digital Library, proceedings of sixth International symposium on Software Metrics, 1999; 242-249.