Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability

Lov Kumar\textsuperscript{a}, Debendra Kumar Naik\textsuperscript{b}, Santanu Ku. Rath\textsuperscript{c}
\textsuperscript{a}Dept. of CSE, NIT Rourkela, India
\textsuperscript{b}Dept. of CSE, NIT Rourkela, India
\textsuperscript{c}Dept. of CSE, NIT Rourkela, India

Abstract

In this study, empirically investigates the relationship of existing class level object-oriented metrics with a quality parameter i.e., maintainability. Here, different subset of Object-Oriented software metrics have been considered to provide requisite input data to design the models for predicting maintainability using Neuro-Genetic algorithm (hybrid approach of neural network and genetic algorithm). This technique is applied to estimate maintainability on two different case studies such as Quality Evaluation System (QUES) and User Interface System (UIMS). The performance parameters of this technique are evaluated based on the basis of Mean absolute error (MAE), Mean Absolute Relative Error (MARE), Root Mean Square Error (RMSE), and Standard Error of the Mean (SEM). The results reported that the identified subset metrics demonstrated an improved maintainability prediction with higher accuracy.

Keywords: CK metrics suite; Maintainability; Genetic algorithm; Neurons; Metrics; QUES; UIMS

1. Introduction

Present day software development process emphasis’s the methodology based on Object-Oriented paradigm. It is observed that the quality of Object-Oriented software can be best assessed by the use of different software metrics. A number of metrics have been proposed by researchers and practitioners to evaluate the quality of software. Some of the software metrics available in literature are as follows: Abreu MOOD metric suite\textsuperscript{1}, Bansiya and Davis (QMOOD metrics suite)\textsuperscript{2}, Briand \textit{et al.} \textsuperscript{3}, Etzkorn \textit{et al.} \textsuperscript{4}, Halstead\textsuperscript{5}, Henderson-sellers\textsuperscript{6}, Li and Henry\textsuperscript{7}, McCabe\textsuperscript{8}, Tegarden \textit{et al.}\textsuperscript{9}, Lorenz and Kidd\textsuperscript{10} and CK metric\textsuperscript{11} suite, etc.

The usefulness of these metrics lies in their ability to predict the quality of the developed software. Software quality attributes, identified by ISO/IEC 9126\textsuperscript{12} are efficiency, functionality, maintainability, portability, reliability and usability. In recent years, maintainability plays a high priority role for achieving considerable success in software
system and it is considered as an essential quality parameter. The ISO/IEC 9126 standard defines maintainability as the capability of the software product to be modified, including adaptation or improvements, corrections of the software to changes in environment and in requirements and functional specifications. In this paper, maintainability is considered as the number of source of lines changed per class. A line change can be an ‘addition’ or ‘deletion’ of lines of code in a class. In this proposed analysis of maintainability estimation, hybrid approach of ANN and genetic algorithm i.e., Neuro-genetic (Neuro-GA) approach has been used for predicting software maintainability on two software products available commercial such as Quality Evaluation System (QUES) and User Interface System (UIMS). To train these models, Object-Oriented software metrics are considered as input data.

The remainder of the paper is organized as follows: Section 2 shows the related work in the field of software maintainability estimation and Object-Oriented metrics. Section 3 emphasizes on mining of metrics values from data repository. Section 4 briefs about the methodologies used to estimate the maintainability. Section 5 highlights on the results for maintainability prediction, achieved by applying Neuro-GA approach. Section 6 gives a note (comparison) on the performance of the designed models based on the performance parameters. In Section 7 threats to validity have been discussed and Section 8 concludes the paper with scope for future work.

2. Related work

It is observed in literature that software metrics are used in design of prediction models which serve the purpose of computing the prediction rate in terms of accuracy such as fault, effort, re-work and maintainability. In this paper, emphasis is given on work done on the use of software metrics for maintainability prediction. Table 1 shows the summary of literature review done on the area of software maintainability, where it describes the applicability of numerous software metrics used by various researchers and practitioners in designing their respective prediction models.

<table>
<thead>
<tr>
<th>Author</th>
<th>Prediction technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Li and Henry (1993)7</td>
<td>Regression based models</td>
</tr>
<tr>
<td>Paul Oman (1994) et al.15</td>
<td>Regression based models</td>
</tr>
<tr>
<td>Scott L. Schneberger (1997)17</td>
<td>Regression based models</td>
</tr>
<tr>
<td>Binkley et al. (1998)18</td>
<td>Regression analysis</td>
</tr>
<tr>
<td>Bandi et al. (2003)19</td>
<td>variance, correlation, and regression analysis.</td>
</tr>
<tr>
<td>Van Koten et al. (2006)20</td>
<td>Bayesian Network, Backward Elimination and Stepwise Selection, Regression Tree.</td>
</tr>
<tr>
<td>Yuming Zhou and Hareton Leung (2007)13</td>
<td>Artificial neural network, Multivariate linear regression, Multivariate adaptive regression splines, Regression tree, Support vector regression and</td>
</tr>
<tr>
<td>Jie-Cherng Chen and Sun-Jen Huang (2009)21</td>
<td>Regression analysis</td>
</tr>
</tbody>
</table>

From the above table 1, it can be interpreted that many of the authors have used statistical methods such as regression based analysis and their forms in predicting the maintainability. But keen observation reveals that very less work has been carried out on using neural network models for designing their respective prediction models. Neural network models over the years have seen an explosion of interest, and their applicability across a wide range of problem domains. Indeed, neural network models can be mainly used to solve problems related to prediction and classification. Neural network models act as efficient predictors of dependent and independent variables due to its character in modeling where in they possess the ability to model complex functions. In this paper, software metrics have been considered for predicting maintainability by applying three artificial intelligence techniques.
3. Research background

The following subsections highlight on the metrics set used for computing maintainability. Data are normalized to obtain better accuracy and then dependent and independent variables are chosen for maintainability estimation.

3.1. Metrics set and empirical data collection

Metrics suites are defined for different goals such as effort estimation, fault prediction, re-usability and maintenance effort. In this paper, different set of CK metrics, Li and Henry metrics and size metrics have been considered for predicting software maintainability i.e., the number of lines changed per class is considered as a criterion in determining the maintainability of a class. A line change can be an ‘addition’ or ‘deletion’ of lines of code in a class. Classic-ADa metrics analyzer was used to gather metrics from Classic-ADa’s design and source code. Classic-ADa is an Object-Oriented programming language that adds the capability of Object-Oriented programming to ADa by providing Object-Oriented construct in addition to the ADa constructs. Figure 1 depicts the flow chart to extract the change i.e., altered lines of code in the developed software.

![Flow chart for extraction of change in a software product](image)

3.2. Effectiveness of metrics

Once the software maintainability data values are determined, an attempt is made to establish a relationship between the required software maintainability and the metrics. The maintainability ‘change’ is measure as “the number of lines changed per class”. Hence in this approach, change is considered as a dependent variable and each of the metric group as a set of independent variables while developing the relation. Maintainability is thus assumed to be a function of the used metrics. To analyze the effectiveness of the metrics used, in the paper the various metrics are categorized into different groups as follows:

a. **Analysis 1 (A1)**: CK metrics (WMC, DIT, NOC, CBO, RFC, and LCOM) are mostly used for evaluate the quality of Object-Oriented software. The important aspect of CK metrics suite is that it covers most of the features of the Object-Oriented software i.e., size, cohesion, coupling and inheritance. WMC indicates size or complexity of class, DIT and NOC indicate the class hierarchy, CBO and RFC indicate class coupling and LCOM indicates cohesion. In this first category, CK metrics suite is considered for finding the maintainability. CBO metric of the CK metrics suite has not been considered for computing maintainability as it measures “non-inheritance related coupling”. Maintainability may be modeled as a function of different CK metrics such as:

\[
\text{Maintainability} = \text{Change} = f(\text{WMC, DIT, NOC, RFC, LCOM})
\]  

b. **Analysis 2 (A2)**: In this second category, the combination of both CK metrics suite and that of Li and Henry metrics (DAC, MPC and NOM) are considered for finding the maintainability. The relationship can be represented as follows:

\[
\text{Maintainability} = \text{Change} = f(\text{WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM})
\]
c. **Analysis 3 (A3):** In this third category, the SIZE metrics along with the combination of CK metrics suite and Li and Henry metrics are used for estimating the maintainability. The relationship is represented as follows:

\[
\text{Maintainability} = \text{Change} = f(WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, SIZE1, SIZE2)
\]  

(3)

d. **Analysis 4 (A4):** In the fourth analysis, only the SIZE metric is used in finding the effectiveness of predicting software maintainability. The relationship between the size metrics and Maintainability can be shown as follows:

\[
\text{Maintainability} = \text{Change} = f(SIZE1, SIZE2)
\]  

(4)

### 3.3. Case study

In this paper, in order to analyze the effectiveness of the proposed approach, two Object-Oriented software data sets published by Li and Henry (1993) are used as case studies. Softwares such as User Interface System (UIMS) and Quality Evaluation System (QUES) are chosen for computing the maintenance effort. The softwares systems viz., UIMS and QUES have 39 and 71 classes respectively. The data have been collected over the past three years. The maintainability of a software is measured by the number of lines changed per class. Table 2 shows the Min, Max, Median and Standard deviation values of the two software systems (UIMS and QUES). In this analysis, the derivative of inheritance metric ‘NOC’ in QUES software product, has all its 71 classes with NOC values zero. This indicates that there are no immediate sub-classes of a class in the class hierarchy and hence NOC is not considered in computing maintainability in this analysis.

<table>
<thead>
<tr>
<th></th>
<th>UIMS</th>
<th>WMC</th>
<th>NOC</th>
<th>DIT</th>
<th>RFC</th>
<th>LCOM</th>
<th>MPC</th>
<th>DAC</th>
<th>NOM</th>
<th>SIZE1</th>
<th>SIZE2</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max.</td>
<td>69</td>
<td>8</td>
<td>4</td>
<td>101</td>
<td>31</td>
<td>12</td>
<td>21</td>
<td>40</td>
<td>1</td>
<td>439</td>
<td>61</td>
<td>289</td>
</tr>
<tr>
<td>Min.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Median</td>
<td>2.15</td>
<td>11.38</td>
<td>2</td>
<td>17</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>74</td>
<td>9</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>15.89</td>
<td>2.01</td>
<td>0.94</td>
<td>23.20</td>
<td>7.48</td>
<td>4.33</td>
<td>2.41</td>
<td>11.38</td>
<td>106.44</td>
<td>13.97</td>
<td>46.82</td>
<td></td>
</tr>
<tr>
<td>Std Dev.</td>
<td>15.89</td>
<td>2.01</td>
<td>0.94</td>
<td>23.20</td>
<td>7.48</td>
<td>4.33</td>
<td>2.41</td>
<td>11.38</td>
<td>106.44</td>
<td>13.97</td>
<td>46.82</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>QUES</th>
<th>WMC</th>
<th>NOC</th>
<th>DIT</th>
<th>RFC</th>
<th>LCOM</th>
<th>MPC</th>
<th>DAC</th>
<th>NOM</th>
<th>SIZE1</th>
<th>SIZE2</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max.</td>
<td>83</td>
<td>0</td>
<td>4</td>
<td>156</td>
<td>33</td>
<td>42</td>
<td>25</td>
<td>57</td>
<td>1</td>
<td>1009</td>
<td>82</td>
<td>42.09</td>
</tr>
<tr>
<td>Min.</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>115</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Median</td>
<td>9</td>
<td>NA</td>
<td>2</td>
<td>40</td>
<td>5</td>
<td>17</td>
<td>2</td>
<td>6</td>
<td>211</td>
<td>10</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>14.95</td>
<td>0</td>
<td>1.91</td>
<td>54.38</td>
<td>9.18</td>
<td>17.75</td>
<td>3.44</td>
<td>13.41</td>
<td>275.58</td>
<td>18.03</td>
<td>62.18</td>
<td></td>
</tr>
<tr>
<td>Std Dev.</td>
<td>17.05</td>
<td>0</td>
<td>0.52</td>
<td>32.67</td>
<td>7.30</td>
<td>8.33</td>
<td>3.91</td>
<td>12.00</td>
<td>171.60</td>
<td>15.21</td>
<td>42.09</td>
<td></td>
</tr>
</tbody>
</table>

### 4. Proposed work for predicting Maintainability

This section gives a brief note on the use of a hybrid approach on Artificial neural network (ANN) and genetic algorithm (GA) i.e., Neuro-GA approach for predicting software maintainability.

#### 4.1. Neuro-GA Approach

In this approach, genetic algorithm is used for updating the weight during learning phase. A neural network with a configuration of ‘l-m-n’ is considered for estimation i.e., the network consists of ‘l’ number of input neurons, ‘m’ number of hidden neurons, and ‘n’ number of output neurons. In this paper, for input layer, linear activation function is used i.e., the output of the input layer is treated as input of the input layer. It is represented as:

\[
O_i = I_i
\]  

(5)
For hidden layer and output layer, sigmoidal function or squashed-S function is used. The output of hidden layer ‘\( O'_h \) for input of hidden layer ‘\( I'_h \) is represented as:

\[
O_h = \frac{1}{1 + e^{-l_h}} \tag{6}
\]

and output of the output layer ‘\( O'_o \) for the input of the output layer ‘\( I'_o \) is represented as:

\[
O_o = \frac{1}{1 + e^{-l_o}} \tag{7}
\]

The number of weights \( N \) required for this network with a configuration of ‘l-m-n’ can be computed using the following equation:

\[
N = (l + n) * m \tag{8}
\]

with each weight (gene) being a real number and assuming the number of digits (gene length) in weights to be \( d \). The length of the chromosome \( L \) is computed using the following equation:

\[
L = N * d = (l + n) * m * d \tag{9}
\]

For determining the fitness value of each chromosome, weights are extracted from each chromosome using the following equation:

\[
W_k = \begin{cases} 
\frac{1}{10^d} & \text{if } 0 \leq x_{kd+i} < 5 \\
\frac{1}{10^{d-1}} & \text{if } 5 \leq x_{kd+i} < 9 
\end{cases} \tag{10}
\]

The fitness values of each chromosome is determined based on the derived fitness function. The algorithm for deriving fitness function is as follows:

Let \( \bar{I}_j, \bar{T}_j \); \( i=1,2,3,....,N \) where

\( \bar{I}_j = (I_{j1}, I_{j2}, I_{j3}, \ldots, I_{ji}) \) and \( \bar{T}_j = (T_{j1}, T_{j2}, T_{j3}, \ldots, T_{jn}) \)

The fitness values of each chromosome is determined based on the derived fitness function. The algorithm for deriving fitness function is as follows:

Let \( \bar{I}_j, \bar{T}_j \); \( i=1,2,3,\ldots,N \) where

\( \bar{I}_j = (I_{j1}, I_{j2}, I_{j3}, \ldots, I_{ji}) \) and \( \bar{T}_j = (T_{j1}, T_{j2}, T_{j3}, \ldots, T_{jn}) \)

represent the respective input and output pairs of the neural network with a configuration of l-m-n. For each chromosome \( C_i, i = 1, 2, 3, \ldots, p \), belonging to the current population \( P_i \) whose size is \( P \). The following algorithm indicates the steps to find the fitness value of the individual chromosomes in the population.

**Algorithm for fitness function: FITGEN()**

**Input:** \( \bar{I}_i = (I_{i1}, I_{i2}, I_{i3}, \ldots, I_{ii}) \)

**Output:** \( \bar{T}_i = (T_{i1}, T_{i2}, T_{i3}, \ldots, T_{in}) \)

where \( \bar{I}_i, \bar{T}_i \) represent the input and output pairs of the l-m-n configuration of neural network.

**Step 1:** Weights \( \bar{W}_i \) from \( C_i \) are calculated using equation 10.

**Step 2:** Considering \( \bar{W}_i \) as a constant weight, the network is trained for \( N \) input instances and the estimate value \( O_i \) is found.

**Step 3:** Error \( E_j \) for each input instance \( j \) is computed using following equation:

\[
E_j = (T_{ji} - O_{ji})^2 \tag{11}
\]

**Step 4:** Root mean square error (RMSE) for the chromosome \( C_i \) is computed using the following equation:

\[
E_i = \sqrt{\frac{\sum_{j=1}^{N} E_j}{N}} \tag{12}
\]

where \( N \) is the total number of training data set.
**Step 5:** Fitness value for chromosome $C_i$ using the following equation is found out as:

$$F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\sum_{j=1}^{N} E_j/N}}$$  

(13)

Figure 2 shows the block diagram for Neuro-GA approach, which represent the steps followed to design the model.

![Flow chart representing Neuro-GA execution](image)

**5. Result and Analysis**

Accuracy of Software maintainability estimation model designed by application of AI techniques is determined by using performance evaluation parameters such as: Mean Relative Error (MRE)$^{24}$, Mean Absolute Relative Error (MARE)$^{24}$, and Standard Error of the Mean (SEM)$^{25}$. Parameters like True error ($e$) and Estimate of true error ($\hat{e}$) are being used for evaluating models involving cross validation approach$^{26}$.

**5.1. Experimental setup**

Following are the numerical values used in execution of Neuro-GA approach for predicting software maintainability.

i. Initialization of chromosome: Let the population of size $N=50$ is considered, initially generated by random process.

ii. Extraction of weight: Each chromosome contains the weight of input to hidden node and hidden node to output. Weight is extracted using Equation 10.

iii. Computing fitness value: The fitness of individual chromosomes is found using the proposed algorithm $FITGEN$. This algorithm is executed with an aim of minimizing the mean square error.
iv. Ranking of chromosomes: The chromosomes in the pool are ranked based on their fitness value. Minimum fitness value chromosome is stripped of by Maximum fitness value chromosome.

v. Crossover: Two point cross-over approach is performed.

vi. Stopping criteria: The execution of the proposed algorithm terminates when 95% of the chromosomes in the pool obtain unique fitness value, beyond this level the fitness value of chromosome get almost saturated.

5.2. Neuro-GA Approach

In this paper, 10-fold cross-validation concept has been considered in QUES and 5-fold cross-validation has been considered in UIMS for comparing the models. True error and estimate of true error determine the suitable model to be chosen for effort estimation. Figure 3 depicts the distribution of the hidden nodes of the suitable model in each fold. The final model chosen for effort estimation is based on the median values of the hidden nodes in their respective folds. Table 3, shows the obtained performance metrics for UIMS and QUES software.

![Fig. 3: NO of Hidden node in each fold](image)

<table>
<thead>
<tr>
<th></th>
<th>UIMS</th>
<th>QUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0.8108</td>
<td>0.8227</td>
</tr>
<tr>
<td></td>
<td>76</td>
<td>74</td>
</tr>
<tr>
<td>Epochs</td>
<td>0.1553</td>
<td>0.1480</td>
</tr>
<tr>
<td>MAE</td>
<td>0.5331</td>
<td>0.4180</td>
</tr>
<tr>
<td>MARE</td>
<td>0.0288</td>
<td>0.0200</td>
</tr>
<tr>
<td>SEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>0.8105</td>
<td>0.8643</td>
</tr>
<tr>
<td></td>
<td>86</td>
<td>72</td>
</tr>
<tr>
<td>Epochs</td>
<td>0.1364</td>
<td>0.1534</td>
</tr>
<tr>
<td>MAE</td>
<td>0.5465</td>
<td>0.4744</td>
</tr>
<tr>
<td>MARE</td>
<td>0.0413</td>
<td>0.0155</td>
</tr>
<tr>
<td>SEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>0.8831</td>
<td>0.8831</td>
</tr>
<tr>
<td></td>
<td>76</td>
<td>64</td>
</tr>
<tr>
<td>Epochs</td>
<td>0.0831</td>
<td>0.1296</td>
</tr>
<tr>
<td>MAE</td>
<td>0.3155</td>
<td>0.3775</td>
</tr>
<tr>
<td>MARE</td>
<td>0.0525</td>
<td>0.0134</td>
</tr>
<tr>
<td>SEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>0.8872</td>
<td>0.8839</td>
</tr>
<tr>
<td></td>
<td>82</td>
<td>68</td>
</tr>
<tr>
<td>Epochs</td>
<td>0.1283</td>
<td>0.1245</td>
</tr>
<tr>
<td>MAE</td>
<td>0.6134</td>
<td>0.4588</td>
</tr>
<tr>
<td>MARE</td>
<td>0.0269</td>
<td>0.0192</td>
</tr>
<tr>
<td>SEM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Table 3, it can be concluded that the performance in estimating software maintainability was very poor when SIZE metric was considered for QUES and UIMS software products. When the comparison is done for the A1 and A2 analysis, the performance is much similar for QUES and UIMS data sets. And when all the metrics (A3 analysis) are considered, the performance is much better for QUES and UIMS but also the complexity of the model increased due to increase in input nodes. The high value of ‘R’ shows the strong affinity between software metrics and maintainability. Figure 4 shows the variance of number of chromosomes having same fitness value and generation number of UIMS and QUES.

Figure 5 shows the Pearson residual boxplots for all four analysis (A1 to A4) for the normalized data of UIMS and QUES data set, allowing a visual comparison. The line in the middle of each box represents the median of the Pearson residual. In case of UIMS and QUES data set, A1, A2 and A3 analysis have a median residual close to zero but in A4 analysis it is observed that median residual is slightly greater then zero. It interprets that the data is slightly underestimated. Of all the analysis, A3 analysis has the narrowest box and the smallest whiskers, as well as the few number of outliers. Based on these boxplots, it is evident that A3 analysis gave best estimation accuracy as compared to other three analysis i.e, A1, A2 and A4. Hence it is observed that the model designed by combining all the three metrics (CK metrics suite, Li and Henry metrics and SIZE metrics ) yields better maintenance accuracy value in comparison with the result obtained on the basis of analyzing them individually.
6. Comparison of models

Apart from the comparative analysis done to find the suitable model which can predict the best software maintainability, this paper also makes the comparison of the proposed work with the work done by Yuming Zhou et al.\textsuperscript{13} and Van Koten et al.\textsuperscript{20}. Yuming Zhou et al.\textsuperscript{13} and Van Koten et al.\textsuperscript{20} have used same dataset i.e., UIMS and QUES for predicting maintainability based on different regression and neural network models. They have considered ‘MMRE’ as a performance parameter to compare the models designed for predicting maintainability of Object-Oriented software systems. Table 4 shows the MMRE value of the proposed work and the work done by Yuming Zhou et al.\textsuperscript{13} and Van Koten et al.\textsuperscript{20}. From Table 4, it can be observed that, in case of QUES software MMRE value is almost same but in case of UIMS software, the proposed approach obtained better prediction rate for maintainability.

Table 4: Performance based on MMRE for UIMS and QUES

<table>
<thead>
<tr>
<th>Author</th>
<th>Technique</th>
<th>UIMS</th>
<th>QUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Van Koten et al.\textsuperscript{20}</td>
<td>Bayesian Network</td>
<td>0.972</td>
<td>0.452</td>
</tr>
<tr>
<td></td>
<td>Regression Tree</td>
<td>1.538</td>
<td>0.493</td>
</tr>
<tr>
<td></td>
<td>Backward Elimination</td>
<td>2.586</td>
<td>0.403</td>
</tr>
<tr>
<td></td>
<td>Stepwise Selection</td>
<td>2.473</td>
<td>0.392</td>
</tr>
<tr>
<td>Zhou et al.\textsuperscript{13}</td>
<td>Multivariate linear regression</td>
<td>2.70</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>Artificial neural network</td>
<td>1.95</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>Regression tree</td>
<td>4.95</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>SVR</td>
<td>1.68</td>
<td>0.43</td>
</tr>
<tr>
<td></td>
<td>MARS</td>
<td>1.86</td>
<td>0.32</td>
</tr>
<tr>
<td>Proposed Work</td>
<td>Neuro-GA</td>
<td>0.3155</td>
<td>0.3775</td>
</tr>
</tbody>
</table>

7. Threat to Validity

Several issues affect the results of the experiment are:

- Two Object-Oriented systems, i.e., UIMS and QUES used in this study are design in ADA language. The models design in this study are likely to be valid for other Object-Oriented programing language, i.e., Java or C++. further research can extend to design a model for other development paradigms.
- In this study, only eleven set of software metrics are used to design a models. Some of the metrics which are widely used for Object-Oriented software are further considered for predicting maintainability.
- we only consider AI techniques for designing the prediction models to predict maintainability. Further, we can extend the work to reduce the feature using feature reduction techniques, i.e., PCA, RST, statistical test etc.
8. Conclusion

In this paper, an attempt has been made to use subset of class level object-oriented metrics in order to predicting software maintainability. Neuro-GA approach was used to design a model by employing 10-fold and 5-fold cross-validation technique for QUES and UIMS software. Four analysis approaches of different metrics sets were considered for estimating the software maintainability for QUES and UIMS software. These techniques have the ability to predict the output based on historical data. The software metrics are taken as input data to train the network and estimate the maintainability of the software product. From this analysis it can be concluded that Neuro-GA approach obtained promising results when compared with the work done by Van Koten et al. and Zhou et al. Also the results reported that the identified subset metrics demonstrated an improved maintainability prediction with higher accuracy.

Further, work can be replicated in estimating the maintainability of the software products by combining neural network with other techniques such as Particle swarm optimization, Fuzzy logic, Clonal selection algorithm etc.

References