FAULT DATA DETECTION IN SOFTWARE USING A NOVEL FGRNN ALGORITHM

Neeta Rastogi

Department of Computer Science and Engineering,
Babu Banarasi Das National Institute of Technology and Management, Lucknow INDIA

Shishir Rastogi

School of Computer Applications, Babu Banarasi Das University, Lucknow INDIA

Manuj Darbari

Department of Computer Science and Engineering
School of Engineering, Babu Banarasi Das University, Lucknow INDIA

ABSTRACT

The use and dependence on software in various fields has been the reason why researchers for past decades have spent their efforts on finding better methods to predict software quality and reliability. Soft computing methods have been used to bring efficient improvement in prediction of software reliability. This study proposed a novel method called Fuzzy Greedy Recurrent Neural Network (FGRNN) to assess software reliability by detecting the faults in the software. A deep learning model based on the Recurrent Neural Network (RNN) has been used to predict the number of faults in software. The proposed model consists of four modules. The first module, attribute selection pre-processing, selects the relevant attributes and improves generalization that improves the prediction on unknown data. Second module called, Fuzzy conversion using membership function, smoothly collects the linear sub-models, joined together to provide results. Next, Greedy selection deals with the attribute subset selection problem. Finally, RNN technique is used to predict software failure using previously recorded failure data. To attest the performance of the software, the popular NASA Metric Data Program datasets are used. Experimental results show that the proposed FGRNN model has better performance in reliability prediction compared with existing other parameter based and NN based models.

Keywords: Software Quality, Software Reliability, Software Reliability Growth Model, DNN, RNN, FGRNN.

Cite this Article: Neeta Rastogi, Shishir Rastogi, Manuj Darbari, Fault Data Detection in Software Using a Novel FGRNN Algorithm, *International Journal of Computer Engineering and Technology*, 10(1), 2019, pp. 236-252.

<http://iaeme.com/Home/issue/IJCET?Volume=10&Issue=1>

1. INTRODUCTION

In recent days, quality of software is the main issue arising while predicting the fault in software systems. In the day to day life, the human life needs a perfect quality service with no defaults in it. The tolerance of unpredictable behavior of the software has been creating the irrecoverable loss situations at a high cost. The present need is the reliability of the system. In any program there will occur lots of failures; the optimization of the software should reduce the failures in the running time to make it reliable [1]. The reliability of the software is defined as the possibility of the low occurrence of a failure in the period in such an environment. In the present scenario, the designing of the software reliability has got vital importance [2]. Reliability of the software is the ability of the system to perform the required operations based on specified conditions for a stated period, and it is a significant feature that is essential and defines the quality of the systems.

In this paper a novel FGRNN method has been proposed to overcome the limitations of the existing methods comprising of four modules. In the first module attribute selection is done in a transparent, easier to interpret manner which assists faster model induction and structural knowledge which is important to our application. The second module uses a fuzzy logic system having mathematical concepts with fuzzy reasoning and it is flexible to add or delete the rules. It takes imprecise, distorted and noisy input information. In the third module a greedy selection algorithm is used which is easy to describe and code up than other algorithms. The fourth module uses RNN model which provides efficient results in software fault prediction. This paper is divided into five sections. Section I is introduction to the requirement of accurate fault prediction methods of software to ensure software reliability and the concept of novel FGRNN method is given. Section II gives a brief insight into the work already existing in this area. Section III describes the research methodology used with experimental setup. Section IV gives the results obtained from implementing the proposed model and discussions. Section V describes the conclusions.

2. RELATED WORKS

Manmath Kumar et.al, (2016) proposed fuzzy min-max algorithm together with recurrent neural network technique. This model gave good results for prediction capabilities of the developed fuzzy-neural networks model for software reliability [3]. Mohamad Mahdi Askari et.al, (2014) proposed a hybrid method in which, a new learning approach was used in multi-layer perceptron neural networks algorithm that increased network efficiency significantly. The efficiency of the used hybrid method was compared and analyzed against 11 statistical and machine learning models on 3 NASA datasets and showed that this model has better efficiency [4]. Jian Li et.al, (2017) used a defect prediction framework called DP-CNN (Defect Prediction via Convolutional Neural Network), that used CNN for automatically generating feature from source code while preserving the semantic and structural information [5]. Hirali Amrutiya et.al, (2017) implemented software defect detection system using Fuzzy C- Means clustering and Support vector machine [6]. Tyagi, & Sharma (2014) discussed a model for estimating CBSS reliability, known as an adaptive neuro fuzzy inference system (ANFIS) based on two essential elements of soft computing, neural network, and fuzzy logic. As compared to FIS model, ANFIS gives a more accurate measure of reliability by reducing error from 11.74%, in FIS model, to 6.66% in ANFIS [7]. Dimov, & Punnekkat, (2010) proposed the component-based reliability systems by using fuzzy networks [8]. Su and Huang (2007) applied neural networks to predict software reliability. They then used the neural network approach to build a dynamic weighted combinational model (DWCM), and proved by experimental results that the proposed model gave significantly better predictions [9].

Study of related works suggest a categorization of soft computing techniques that may be used for the faults' recognition of software as represented in figure 1.

Figure 1. Soft Computing techniques

3. RESEARCH METHODOLOGY

3.1. Deep Neural Networks

Neural networks are nets of processing elements that can learn the mapping existent between input and output data. The neuron computes a weighted sum of its inputs and generates an output if the sum exceeds a certain threshold. This output then becomes an excitatory (positive) or inhibitory (negative) input to other neurons in the network. The process continues until one or more outputs are generated [10].

3.2. RNN Encoder-Decoder

In software testing, the software reliability is assessed and prediction of fault number by using the deep learning model known as RNN encoder-decoder. This model encodes an input sentence into a vector and then decodes the vector into another sentence [11]. The RNN encoder decoder includes two aspects: the encode process [i.e., the input sequence $\{X_1, X_2, \dots, X_t\}$ through the encoder is converted to vector V], and the decode process [i.e. vector V that passes through the decoder is converted to the output sequence $\{Y_1, Y_2, \dots, Y_t\}$]. The basic structure of the RNN encoder–decoder is shown in Figure 2. The decoding process is an inverse encoding process in the RNN. It can be applied to predict the next output Y_{t+1} through vector V and after the predicted output $\{Y_1, Y_2, \dots, Y_t\}$. It can also be written in the RNN as follows:

$$P(Y_{t+1}|\{Y_1, Y_2, \dots, Y_t\}, V) = G(Y_t, H_{t+1}, V) \quad (1)$$

Figure 2. RNN encoder-decoder

3.3. Proposed Fuzzy Greedy Recurrent Neural Network Algorithm

The purpose of the proposed work is to predict the software faults through the deep NN model which can capture the stable and accurate features using proposed FGRNN (Fuzzy Greedy Recurrent Neural Network) algorithm. It gives a globally optimal solution through a FGRNN algorithm. It can also automatically learn better feature representations from software faults than other methods. FGRNN is used to examine the software data effectively and providing the better software reliability level prediction results. The performance of fault detection data and fault correction data as training data for deep NN using FGRNN algorithm are obtained here. This method provides better robustness when compared with other traditional methods.

3.3.1. Pre-processing of dataset

The Attribute Selection Pre-processing using Greedy Stepwise Search is as follows:

Step 1: Load the dataset

Step 2: Initialize the Source Data

```
source ← new DataSource(fp);
```

Step 3: Initialize the Data Set

```
dataset ← source.getDataSet();
```

Step 4: create AttributeSelection object

```
filter ← new AttributeSelection();
```

Step 5: create evaluator and search algorithm objects

```
eval ← new CfsSubsetEval();
```

Step 6: Initialize the GreedyStepwise Search

```
search ← new GreedyStepwise();
```

Step 7: Set the algorithm to search backward

```
search.setSearchBackwards(true);
```

Step 8: Set the filter to use the evaluator and search algorithm

```
filter.setEvaluator → eval;
```

```
filter.setSearch → search;
```

Step 9: specify the dataset

`filter.setInputFormat → dataset;`

Step 10: apply the Filter

`newData ← Filter.useFilter(dataset, filter);`

Step 11: Get Greedy Ranking Tip Text

`Greedy Ranking Tip Text ← search.generateRankingTipText();`

Step 12: Get Greedy Calculated Select

`Greedy Select 1 ← search.getCalculatedNumToSelect();`

Step 13: Get Greedy Number to Select

`Greedy Select 2 ← search.getNumToSelect();`

Step 14: Search the get greedy revision

`Greedy Revision ← search.getRevision();`

Step 15: Set the Greedy Threshold

`Greedy Threshold ← search.getThreshold();`

Step 16: Get the Filter Evaluator

`Filter Evaluator ← filter.getEvaluator();`

Step 17: Search the Filter

`Filter Search ← filter.getSearch();`

Step 18: Get the dataset Number of Attributes

`Number of Attributes ← newData.numAttributes();`

Step 19: get the Number of Class present in the dataset.

`Number of Classes ← newData.numClasses();`

Step 20: Initialize the instances

`Number of Instances ← newData.numInstances();`

Step 21: Map the Class Index

`Number of Class Index ← newData.classIndex();`

Step 22: Update the sum of weights

`Sum of Weights ← newData.sumOfWeights();`

Step 23: Greedy Stepwise Search Completed

Step 24: Attribute Selection Preprocessing Completed

Figure 3. Flowchart of the attribute selection pre-processing

3.4. Mathematical modeling of FGRNN

Initial phase of the proposed method is to select the attributes through pre-processing method. Let attribute A has v distinct values. Let s_{ij} be number of samples of class C_i in a subset S_j . It contains those samples in S that have value a_j of A. The entropy, or expected information based on the partitioning into subsets by A, is given by,

$$E(A) = -\sum_{i=1}^m I(S) \frac{s_{1i}+s_{2i}+...+s_{mi}}{s} \quad (2)$$

The above value represents the information generated by splitting the training data set S into v partitions corresponding to v outcomes of a test on the attribute A. The greedy evaluation function denotes a degree of priority for incorporating the corresponding element 'x' into the solution under construction. An alternative approach is the set X as a fuzzy set with a well-defined membership function $\mu(x)$, the form of which is given by,

$$\mu(x) = \frac{1}{1+\lambda^2 \left[\left(\frac{1-\lambda}{\lambda} \right) x - \theta \right]^2} \quad (3)$$

$$greedy_{dsize} = \sum_{j=0}^{dsize} (dset(j) > dset(j+1)) \quad (4)$$

The RNN model has two external input variables x_1 , x_2 and a single output y . Consequently, RNN has two nodes in layer 1 and one node in layer 5. This method is described layer by layer in order to understand about the mathematical function of each node clearly.

- In layer 1, each node corresponds to one input variable and directly transmits input values to the next layer, thus requires no computation.
- In layer 2, each node corresponds to one fuzzy set and calculates a membership value.

$$S_{testsetsize} = \sum_{i=0}^{testsetsize} (testset(i) < i) \quad (5)$$

$$\Phi(z) = \frac{1}{1+e^{-1}} \quad (6)$$

When using FGRNN, each internal variable has a single corresponding fuzzy set. Links in layer 2 are all set to unity.

- In layer 3, each node represents a fuzzy logic rule and performs antecedent matching of this rule using the following AND operation,

$$\mu_i = M_S(h_i) \cdot \prod_{j=1}^n M_G^i(x_j) = \frac{1}{1+\exp\{-h_i\}} \prod_{j=1}^n \exp\left\{-\left(\frac{x_j - m_{ij}}{\sigma_{ij}^2}\right)^2\right\} \quad (7)$$

where, n is the number of external inputs. The link weights are all set to unity.

- In layer 4, each node corresponds to one context node and performs a de-fuzzification operation for internal variables h . The simple weighted sum is calculated in each node,

$$h_i = \sum_{k=1}^r \mu_k w_{ik} \quad (8)$$

- In layer 5, each node corresponds to one output variable and performs weighted average operations for output y . The mathematical function is,

$$y = \frac{\sum_{i=1}^r \mu_i b_i}{\sum_{i=1}^r \mu_i} \quad (9)$$

where, b_i is a fuzzy singleton value functioning as the consequent part of output variable y .

The Proposed FGRNN algorithm is as follows:

Step 1: Initiate attribute_selected_dataset
 Step 2: Fuzzy Set Conversion.
 Step 3: Configure the parameter with length=20, epochs=30000 and weighs = -100 to 100.
 Step 4: Generate the RNN network using Equation (6).
 Step 5: Select the greedy selection algorithm and construct the training sets using Equation (4).
 Step 6: The sample of data is used to fit the model. The actual dataset trains the model with weights and biases to generate the samples from the trained set by $trainset = \sum_{i=1}^{10} N(x_i, y_i)$ (10)
 Step 7: Validate the test set and trained set values. The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper parameters and the test set is used to evaluate competing models.
 Step 8: Evaluate the learning process using Equation (5).
 Step 9: Endorse the RNN classification technique.
 Step 10: Initiate the greedy select algorithm.
 For i , select the first activity constantly
 For $j < n$, consider rest of the activities
 Step 11: Select the activity time greater than or equal to the finish time of previously selected activity. $s[j] \geq f[i]$ using Equation (4)
 Step 12: Generate new RNN process and compute a sequence of hidden states

$(h_1, \dots, h_T)$ and sequence of outputs $(o_1, \dots, o_T)$ using $h_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$ and $o_t = W_{oh}h_t + b_o$ (11)

Step 13: Generate the input layer, hidden layer and output layer with Sigmoid Equation (6).

Step 14: Next, Generate the net core generator while forming the network and it runs web applications in Kestrel web server and also included in .net core SDK. If it is false, then, last layer will be -1.

Step 15: Update the last layer using MLP generator of the input layer in RNN.

Step 16: Link the last layer and the current layer

Step 17: Verify the hidden layer, if last layer < 0 , there is no defined input layer.

Step 18: Update the last layer and verify the hidden layer

Step 19: Create the feed forward links and recurrent links and enter into the return layer.

Step 20: Use final value of Boolean bias and double bias in the output layer. Here, in double bias condition, the weight w and bias b of a node gives the output of the activation function by $f(x, a, b) = \frac{1}{1+e^{-(ax+b)}}$ (12)

Then, the Boolean bias gives $p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$, where p and q are inputs of the Boolean function.

Step 21: Generate the bias of the net core generator with double values.
Use bias = epoch (trainer)

Step 22: The step 20 and 21 define the min error present in the software to predict the fault and non-fault values.

Step 23: End the process of FGRNN

Figure 4. Flowchart of the proposed FGRNN

3.5. Experimental Setup

Benchmark datasets were collected from 12 NASA MDP datasets to demonstrate the effectiveness of the software reliability. The probability of detection (PD) rises with effort and rarely rises above it. High PDs are associated with high PFs (Probability of failures). It can change significantly while accuracy remains essentially stable. With two notable exceptions, detectors learned from one data set (e.g. KC1) have nearly the same properties when applied to another (e.g. PC1). It is used in the attribute benchmark dataset values of the proposed method during pre-processing. Many researchers use static measures to guide software quality predictions. The four types of the software metrics used in this proposed method are as follows:

1. Cyclomatic Complexity [V(G)]

The total number of linearly independent paths in a program in graphic representation is called cyclomatic complexity V(G) indicated by,

$$V(G) = E - N + P$$

Where E denotes total no of edges of any graph G, N is number of nodes and P is the number of connected components in graph G. It is used as measure to avoid excessive complexities that cause reliability problems and is used to quantify the test program that detects errors.

2. Essential Complexity [eV(G)]

It gives the measure of the degree of unstructured constructs contained in a module hence a measure of the quality of the code. It is used to predict the effort required in maintenance also further aiding in the modularization process.

3. Design Complexity [iV(G)]

It measures the number of calls to other modules in a system, basically a measure of the module's decision-making components.

4. Lines of Code (LOC)

LOC are a size metric giving total number of lines of code, which quantifies the software size.

4. RESULTS AND DISCUSSIONS

Models are built as abstractions of real-world problems for analysis like predicting software quality and reliability as is the case taken for this paper. If a model is underfit, it takes into account so less attributes that important changes on the object may be ignored. Therefore, the model may make wrong predictions. To overcome this problem of underfitting an expected value of target variable is modeled as nth degree polynomial yielding the general polynomial. The training error will decrease when the degree of the polynomial is increased. So, the performance may be improved by increasing the number of training epochs. The capacity of the model is increased by increasing the number of memory cells in a hidden layer or number of hidden layers. So far, adopting the suitable types and number of fault datasets to assess the performance of software reliability models had remained an unresolved issue. This proposed method focused on all data types in the area of software analytics in order to predict the accuracy of our proposed algorithm to give better results in terms of accuracy, ROC curve, Precision, Recall and F- measure.

In this study, we used the original versions of the data sets from the NASA Metrics Data Program (MDP) Repository. The proposed FGRNN algorithm was implemented over a simulation software developed using JAVA/J2EE. The dataset used should be in CSV format. Working of FGRNN on MDP dataset PC4 is included in this paper and detailed here:

LOC	BRANCH	CALL	PARAM	LOC CODE AND COMMENT	LOC COMMENTS	CONDITION COUNT	CYCLOMATIC COMPLEXITY	CYCLOMATIC DENSITY	DETECT
17	14	2	0	0	0	10	0	0.23	
2	0	0	0	0	1	10	0	0.26	
3	0	1	1	1	0	0	0	0.17	
4	3	1	0	0	0	0	2	0.2	
7	0	1	2	0	0	0	0	0.23	
8	11	0	0	0	0	20	0	0.46	
2	1	0	0	0	0	0	1	0.23	
7	0	0	0	0	0	0	0	0.13	
10	11	1	10	43	20	11	0.19		
1	0	0	0	0	0	0	0	0.20	
0	0	1	1	4	0	2	0.23		
0	0	0	0	0	0	25	1		
1	0	0	0	0	0	0	0	0.09	

Total Number of Instances/Records:1458
Total Number of Instances/Records Attributes:38

Attribute Selection Pre-Processing

Figure 5. View dataset (PC4)

Figure 5 shows the obtained dataset values with total number of instances/records and its attributes. For the selected dataset, Total number of Instances/Records: 1458

Total number of Instances/Records Attributes: 38

LOC CODE AND COMMENT	CONDITION COUNT	PARAMETER COUNT	PERCENT COMMENTS
1	20	1	11.23
0	10	1	10
1	0	1	10.53
0	1	0	0
3	0	2	13
0	20	10	0
0	0	0	0
0	0	1	0
10	10	0	50.10
0	0	0	0
1	1	1	31.25
0	0	0	0
1	0	1	10.70

Total Number of Instances/Records:1458
Total Number of Instances/Records Attributes:5

Fuzzy Conversion

Figure 6. Attribute selection for pre-processing (PC4)

From the 38 attributes the datasets are pre-processed by using only 5 attributes to give the optimized results. It consumes the buffer memory to proceed further.

CODE AND COMMENT	CONDITION COUNT	PARAMETER COUNT	PERCENT COMMENTS
1	20	1	11.23
0	10	1	10
1	0	1	10.53
0	1	0	0
3	0	2	13
0	20	10	0
0	0	0	0
0	0	1	0
10	10	0	50.10
0	0	0	0
1	1	1	31.25
0	0	0	0
1	0	1	10.70

Total Number of Instances/Records:1458
Total Number of Instances/Records Attributes:5

Fuzzy Conversion

Figure 7 Fuzzy conversion (PC4)

A Fuzzy conversion of the given dataset is shown in Figure 7. It chooses the 0 and 1 values randomly.

CODE AND COMMENT	CONDITION COUNT	PARAMETER COUNT	PERCENT COMMENTS
2	26	7	31.25
9	16	1	16
1	8	1	10.25
9	8	9	9
3	6	2	15
9	26	9	9
9	9	9	9
9	9	1	9
16	26	9	16.66
9	9	9	9
1	8	1	11.25
9	9	5	9
2	8	5	25.25

Total Number of Instances/Records:1458
Total Number of Instances/Records Attributes:5

Download Dataset

Figure 8. Proceed fuzzy greedy recurrent neural network (PC4)

After getting the fuzzy results, the proposed FGRNN is applied to predict the fault number of datasets.

Figure 9. FGRNN Output: Performance Metrics (PC4)

The performance metrics of the FGRNN applied on dataset PC4 is shown in Figure 9. The following performance measures are used to validate the proposed models such as TP rate, FP rate, Precision, Recall, F-measure, ROC area and class. Here, the ROC curve value of the proposed system is 0.8268.

Table 1. Detailed Accuracy by Class for PC4

Metrics	TP rate	FP rate	Precision	Recall	F-measure	ROC area	Class
Correct	0.981	0.736	0.906	0.981	0.942	0.856	False
Incorrect	0.264	0.019	0.662	0.264	0.378	0.856	True
Weighted Average	0.894	0.648	0.876	0.894	0.873	0.856	

Table 2. Confusion Matrix for PC4

a	b	<-- classified as
1256	24	a = FALSE
131	47	b = TRUE

The confusion matrix values are used for calculating following parameters by using the formula given in front of them:

$$\begin{aligned}
 \text{Accuracy} &= ((\text{TP}+\text{TN})/(\text{TP}+\text{TN}+\text{FP}+\text{FN})); \\
 \text{Sensitivity} &= (\text{TP}/(\text{TP}+\text{FN})); \\
 \text{Specificity} &= (\text{TN}/(\text{TN}+\text{FP})); \\
 \text{Precision} &= (\text{TP}/(\text{TP}+\text{FP})); \\
 \text{Recall} &= (\text{TP}/(\text{TP}+\text{FN})); \\
 \text{F-measure} &= 2*((\text{Precision}*\text{Recall}) / (\text{Precision}+\text{Recall}));
 \end{aligned}$$

True Positive rate (TPR i.e. Sensitivity) represents the ratio of instances correctly predicted as fault samples and actual fault samples, as $(\text{TP} / (\text{TP}+\text{FN}))$. False positive rate (FPR) represents the ratio of incorrectly predicted as fault samples and actual non-fault samples, as $(\text{FP} / (\text{FP}+\text{TN}))$. Specificity is $(1-\text{FPR})$. Error rate represents the ratio of number of all incorrect predictions divided by the of the dataset, as $((\text{FP}+\text{FN}) / (\text{P}+\text{N}))$. Accuracy is the ratio of number of all correct predictions divided by the total number of the dataset, as $((\text{TP}+\text{TN}) / (\text{P}+\text{N}))$, i.e. $(1 - \text{Error rate})$. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations, as $(\text{TP} / (\text{TP}+\text{FP}))$. Recall is the ratio of correctly predicted positive observations to the all observations in actual class, as $(\text{TP} / (\text{TP}+\text{FN}))$. F- measure is the weighted average of Precision and Recall, as $\text{F-measure} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$.

ROC (Receiver Operator Curve) is the ratio of the true positive rate (Sensitivity) against the false positive rate $(1 - \text{Specificity})$ for all possible cutoff values. ROC shows the relationship between TPR and FPR, and AUC (Area Under Curve) is equal to the area under the ROC. The range of AUC is between 0 and 1. The larger value of Area Under Curve (AUC) demonstrates better performance of the prediction model.

Table 3. Predicted values for PC4

Accuracy	0.8937
Sensitivity	0.9055
Specificity	0.6619
Precision	0.98125
Recall	0.9639
F-measure	0.9725
ROC area	0.856

Here Table 1, 2 and 3 represents the performance metrics of PC4 dataset. The detailed accuracy by class shows the weighted average of the TP rate, FP rate, Precision, Recall, F-Measure and ROC area. The confusion matrix gives the Accuracy, Sensitivity, Specificity, Precision, Recall and F-measure values.

Figure 10. Performance Metrics

The Total weighted average of various performance markers obtained from the proposed method is shown in Figure 10. The achieved accuracy gives better results when compared with existing methods. It gives accurate values for processing and found software fault present in the data. Here, TP rate increases to 0.894 while FP rate reduces to 0.648. Then, it drastically increases the value of precision to 0.876. High TP or low FP comes at the cost of high PF or low PD respectively. This linkage can be seen in a standard receiver operator curve (ROC). The mean and average value can be calculated using the confusion matrix which is shown in Figure 10.

Correctly classified instances = TP + TN = 1256 + 47 = 1303

Incorrectly classified instances = FP + FN = 24 + 131 = 155

The calculated results for PC4 are as follows:

Table 4. Calculation results for PC4

Correctly classified instances (1303/1458)	0.8937
Incorrectly classified instances (155/1458)	0.1063
Kappa statistic	0.3309
Mean absolute error	0.1508
Root mean squared error	0.2868
Mean Relative absolute error	0.7068
Root relative squared error	0.8761

Figure 11. Graph of calculated results

Similar to PC4, the FGRNN execution results for the CM1 dataset are as follows:

Table 5. Detailed accuracy for CM1

Metrics	TP rate	FP rate	Precision	Recall	F-measure	ROC area	Class
Correct	1	1	0.902	1	0.948	0.463	False
Incorrect	0	0	0	0	0	0.463	True
Weighted Average	0.902	0.902	0.813	0.902	0.855	0.463	

Table 6. Confusion Matrix for CM1

a	b	--	classified as
449	0		a = FALSE
49	0		b = TRUE

Table 6 shows the confusion matrix for the CM1 dataset which is classified as fault and non-fault values. When a=449, then it is the fault value. When a=49, then it is the non-fault value.

Table 7. Predicted values for CM1

Accuracy	0.9016
Sensitivity	0.9016
Specificity	0
Precision	1
Recall	1
F-measure	1
ROC area	0.6914

Table 5, 6 & 7 represents the performance metrics obtained by applying proposed FGRNN prediction model for the NASA MDP Dataset CM1.

Figure 16. ROC curve for CM1

Likewise, the proposed FGRNN Prediction model has been applied on NASA MDP Datasets KC1 and PC1 and the results were obtained then all the results were compiled into Table 8.

Table 8. Validation results

Algorithm	CM1	KC1	PC1	PC4
RIPPER	84.52	82.91	92.07	93.07
C4.5	85.12	81.34	88.39	82.35
CBA2	80.36	83.71	91.78	84.72
NB2	86.74	82.86	89.21	84.95
BBN	76.83	75.99	90.44	76.99
RBF	89.91	84.81	92.84	85.21
KNN	83.27	83.99	91.82	84.12
CFS+NB	84.84	82.46	89.00	83.45
RF	85.75	85.06	91.43	86.23
NB1	82.26	82.44	88.27	84.46
J48	85.46	84.16	90.90	88.24
SVM+EA	90.16	85.4	93.24	87.547
Proposed FGRNN Method	90.1606 (ROC 0.6914)	85.3959 (ROC 0.7845)	93.688 (ROC 0.6894)	89.378 (ROC 0.8268)

Figure 17. Performance graph

The Table 8 and Figure 17 shows the validation of the proposed FGRNN method using NASA MDP Datasets CM1, KC1, PC1 and PC4. The proposed FGRNN method shows accuracy of 90.1606%, 85.3959%, 93.688% and 89.378% for CM1, KC1, PC1 and PC4 datasets respectively. These values provide higher accuracy when compared with existing best method Support Vector Machine and Evolutionary Algorithms (SVM+EA).

Figure 18. ROC curve comparison

Figure 18 shows the ROC curve for the various software prediction methods. The obtained results predicted the fault dataset number in the software system using soft computing methods and when compared to existing techniques, the proposed FGRNN provides higher accuracy.

5. CONCLUSION

There have been many approaches explored for predicting software reliability, some based on analytical models using mathematical formulas while some used soft computing techniques. Neural network-based models were very effective for large datasets. In this paper a novel FGRNN algorithm was proposed which is based on a combination of soft computing techniques. The algorithm was applied on CM1, KC1 PC1 and PC4 datasets of NASA MDP. The results were compared with various other algorithms applied on these datasets as shown in Table 8. Consider the accuracy value achieved using FGRNN: it is 90.1606 for CM1, for KC1 it is 85.3959, for PC1 it comes to 93.688. Thus, it shows that FGRNN achieved higher accuracy than other methods like Support Vector Machine and Evolutionary Algorithms (SVM+EA), NB1, J48 etc. applied on the same datasets. All data types in the area of software analytics were taken care of and FGRNN method proved robust when compared with conventional methods.

REFERENCES

- [1] Erturk, E., & Sezer, E. A., A comparison of some soft computing methods for software fault prediction. *Expert Systems with Applications*, 42(4), 2015, 1872-1879.
- [2] Kiran, N. R., & Ravi, V. (2008). Software reliability prediction by soft computing techniques. *Journal of Systems and Software*, 81(4), 2008, 576-583.
- [3] Bhuyan, M. K., Mohapatra, D. P., Sethi, S., Software Reliability Assessment using Neural Networks of Computational Intelligence Based on Software Failure Data, *Baltic J. Modern Computing*, 4(4), 2016, 1016-1037.
- [4] Askari, M. M., & Bardsiri, K. V. Software Defect Prediction using a High-Performance Neural Network. *International Journal of Software Engineering and Its Applications*, 8(12), 2014, 177-188.
- [5] Li, J., He, P., Zhu, J., & Lyu, R. M. Software Defect Prediction via Convolutional Neural Network, *Proceedings of IEEE International Conference on Software Quality, Reliability and Security*, Prague, Czech Republic, QRS 2017, 318-328.

- [6] Amrutiya, H., Kotak, R., & Joiser, M. Software Fault Detection Using Fuzzy C- Means and Support Vector Machine. *Indian Journal of Science and Research*, 15(2), 2017, 53-56.
- [7] Tyagi, K., & Sharma, A. An adaptive neuro-fuzzy model for estimating the reliability of component-based software systems. *Applied Computing and Informatics*, 10(1-2), 2014, 38-51.
- [8] Dimov, A., & Punnekkat, S. Fuzzy reliability model for component-based software systems. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*, IEEE Computer Society, Lille, France 2010, 39-46.
- [9] Su, Y. S., & Huang, C. Y. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *Journal of Systems and Software*, 80(4), 2007, 606-615.
- [10] Sehra S. K., Brar, S. Y., & Kaur, N. Soft Computing Techniques for Software Project Effort Estimation. *International Journal of Advanced Computer and Mathematical Sciences*, 2(3), 2011, 160-167.
- [11] Wang, J. and Zhang, C., Software Reliability Prediction Using a Deep Learning Model based on the RNN Encoder-Decoder, *Reliability Engineering and System Safety*, 170, Feb 2018, 73-82.
- [12] Catal, C., Sevim, U., & Diri, B. Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. *Expert Systems with Applications*, 38(3), 2011, 2347-2353.
- [13] Jin, C., & Jin, S. W. Software reliability prediction model based on support vector regression with improved estimation of distribution algorithms. *Applied Soft Computing*, 15, 2014, 113-120.
- [14] Kaswan, K. S., Choudhary, S., & Sharma, K. Software reliability modeling using soft computing techniques: Critical review. *Journal of Information Technology and Software Engineering*, 5, 2015, 144.
- [15] Rawat, M. S. and Dubey, S. K. Software defect prediction models for quality improvement: A literature study. *International Journal of Computer Science Issues*, 9(5), 2012, 288-296.
- [16] Rajaganapathy, C. D. & Subramani, A. A Comparative Study of Different Software Fault Prediction and Classification Techniques. *Research Journal of Applied Sciences, Engineering and Technology* 10(7), 2015, 831-840.
- [17] Bhuyan, K. M., Mohapatra, P. D., & Srinivas, S. Software Reliability Prediction using Fuzzy Min-Max Algorithm and Recurrent Neural Network Approach. *International Journal of Electrical and Computer Engineering*, 6(4), 2016, 1929-1938.
- [18] Sehgal, P., & Meenal. Software Reliability Estimation Using Artificial Neural Networks. *International Journal of Research in Management, Science & Technology*, 4(2), 2016, 102-107.
- [19] Pandey, A., & Ahlawat, A. A Novel ANN based Approach for Reliability of Software using Object Oriented Metrics. *International Journal of Computer Applications*, 2(1), 2016, 1-4.
- [20] Rastogi, N., Rastogi, S., Darbari, M., Survey on Software Reliability Prediction using Soft Computing, *International Journal of Computer Engineering and Technology*, 9(4), 2018, 212-216