International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 / https://doi.org/10.14445/22315381/IJETT-V70I4P214

Volume 70 Issue 4, 164-173, April 2022
© 2022 Seventh Sense Research Group®

Original Article

A Neuro-Fuzzy based Automated System for Estimating Software Quality

Ritu[1], O. P. Sangwan[2]

[1,2]*Department of Computer Science & Engineering, Guru Jambheshwar University of Science and Technology, Hisar, Haryana, India*

[1]rituchopra84@gmail.com

Received: 23 February 2022 Revised: 02 April 2022 Accepted: 04 April 2022 Published: 26 April 2022

Abstract *- In this increasing digital software era, various software becomes of daily use in human life, ranging from shopping to meeting, working from home, etc. It is a necessity for good software that is easy to operate, highly secure, and highly accurate. These properties constitute the quality of software. Generally, the quality of the software is estimated based on the expert's opinion or from any other user of that software, which can be time-consuming and may not be highly accurate as it depends upon user-to-user experience. It is a demand for an automated system that the quality of software can be estimated by providing some inputs or features. Due to recent development in machine learning, the neural network has been largely employed in academia as well as industry. Thus, this paper presents a neuro-fuzzy-based automated system to estimate the quality of given software. The user needs to feed only five parameters, namely Reliability, Usability, Functionality, Efficiency, Portability, and Maintainability, and the proposed model automatically calculates the quality of software. The proposed model is based on the data collected from the 128 software, where 100 data-set are used for training the proposed neuro-fuzzy model and 28 data-set for testing purposes. The obtained results with the proposed approach closely match the actual software quality. Moreover, two fuzzy rule generation techniques, i.e. 'grid partition' and 'sub-clustering', have been designed and compared the obtained results with both approaches. It is found that the proposed approach with sub-clustering has lesser error measures like MSE, MRE, and MARE in terms of performance indexes, among other methods.*

Keywords *- ANFIS, Fuzzy logic, Neuro-fuzzy, Neural Networks, MATLAB Simulation, Software Quality.*

1. Introduction

Software quality analysis is one of the most important criteria for determining the software life cycle and reliability [1], [2]. Due to the increasing development of computer-related hardware, the performance of software becomes very critical, and high-performance software is always required. Meanwhile, software development and maintenance costs, as well as development time, are predicted to a reduction in these. Despite a large number of resources and a substantial chunk of an organization's capital expenditures spent, most software purchased by clients or organizations does not match their needs. Poor-quality products and software failures have caused more than an annoyance, especially in this era of ubiquitous computing, where users can assess the system from anywhere at any time. Now that the majority of systems used at home, in hospitals, and in industries are embedded systems. Software faults have resulted in human mortality. Poorly designed user interfaces, specification misinterpretation, and outright programming faults have all been blamed because it is more cost-effective to uncover potential software quality concerns sooner or later in software development due to the cost of maintaining broken or unreliable systems.

It is always good if one is able to estimate the quality of software at the early stage of the design, which can simplify the design of a good final product. By knowing it at an early stage, the designer can take appropriate action on the design part to meet the expectations accordingly. As a result, it reduces the design time, development cost, and development cycle. However, no clearer dependency parameters of software quality and the unavailability of sufficient data at the early stage make it challenging to access the software quality. The very first step of developing a prediction model is needed to evaluate the factors that can affect the quality of software. Further, it is very difficult to precisely identify these parameters. In addition, the degree of influence is inherently ambiguous. The quality of the software is measured using a variety of metrics. There are no definite metrics that affect the software quality. However, according to the ISO/IEC 9126 international standard, six parameters, namely Reliability, Usability, Functionality, Efficiency, Portability, and Maintainability, can access the software quality. Based on the study in [9], it is argued that five attributes are sufficient enough to predict the quality of software. It is evident; that there is a certain relationship between these metrics and the software quality. So, the

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

objective of the proposed automated system of software quality is to determine an effective relationship between these metrics and the software quality.

The rest of the paper is structured as related works are described in section 2, section 3 explains software quality factors, the proposed methodology is given in section 3, section 5 discusses the simulation results, and finally, the conclusion is discussed in section 6.

2. Related Work

There have been numerous studies on soft computing techniques, e.g. neural networks [3] and fuzzy logic [4]. These schemes can be employed for black-box modelling, system identification, classification, prediction, etc. Even in the software field also, these approaches are widely used, whether it is software fault prediction [5], fault classification [6], software quality assessment [7], etc. Various fuzzy logic-based techniques have been introduced to access the quality of software. In [8], a fuzzy rule-based software quality model is developed based on 110 software by employing three inputs, e.g. GUI interface, user manual (UM), meaning error message (MEM). The obtained results are compared with the local regression method. A prediction approach is employed in [9] using a fuzzy technique where 243 rules are used based on the expert's opinion. The estimated quality using the fuzzy model is proved to be a match with the actual software quality.

Another software quality model is developed in [10] based on the inspection along with the defect data collected, estimated the error-prone modules, and the effectiveness of the inspection. The data set used can be influenced during an inspection by the method or type of inspection, and hence, this parameter is included in the prediction model. An object-oriented Petri-net-based quality prediction is proposed in [11] based on the defects at the various stages, starting from the requirement phase to the testing phase. Another fuzzy-based quality prediction model is proposed in [12] based on the software metrics related to the coding structure, design, and requirement. A fuzzy logic approach is employed to estimate the defect metrics of that software based on the three aforementioned parameters, and it is estimated the degree of defects in a particular phase. A priority-based prediction model is discussed in [13] using the Takagi-Sugeno fuzzy model, where the authors predict the priority of software based on the stakeholder's importance, time, cost, and risk factor. The fuzzy-based approach has also been used to access other software metrics, for example, software reliability analysis in [14], [15], software defect prediction in [16], software failure detection in [17], etc. Fuzzy logic-based approaches take into account the linguistic rules, which are based on the expert advice or data collected, and quantitative data is not used. Neural-network has excellent learning capability and adaptability based on the input data. So, on account of the data-based modelling, neural-network-based approaches, e.g. Radial basis function network (RBFN), Multi-layered neural network, and recurrent neural network, can be used. Some studies already have been considered the neuro-fuzzy scheme in software analysis. These are limited to risk analysis, design defects, reliability prediction, etc. Neuro-fuzzy-based reliability and efficiency prediction schemes are developed in [18] based on three attributes, namely DIT, RFC, and WMC. This approach applies to incomplete or missing information of any input parameter and the different data formats. Neural network-based fault-prone prediction of software is proposed in [19] for a particular software by analyzing the code routine structure. A fault is there if there is any change in the code structure upon any reported problem. The output obtained from the NN in the form of fault-prone software is again used to extract more comprehensive rules by using the genetic algorithm to enhance the accuracy of the NN model.

An improved NN-based approach is presented in [20] using an optimization algorithm named Hybrid Cuckoo search (HCS). The weights are learned by using the HCS algorithm for better accuracy and to speed up the process of learning. However, it lacks a detailed analysis of the presented framework on software quality. An ANFIS model is proposed in [21], where the software quality model has been discussed using a three-level approach. At each level, sub-characteristics are estimated using given inputs, and finally, quality is estimated based on usability and efficiency. However, there is no correlation between the three levels. Another ANFIS architecture is provided in [22] to predict software quality in terms of software viability. The viability is divided into three categories, namely full viable, partial viable, and not viable. An FLNN (Fuzzy logic Neural Network) approach is discussed in [23] where instead of software quality prediction, different metrics of quality, e.g. functionality, usability, maintainability, efficiency, are predicted based on six applied inputs. The FLNN employs a neural network architecture with four fuzzy rules and three layers of NN. An adaptive neuro-fuzzy approach is presented in [22] for a web-based software quality prediction scheme by taking six software metrics as input. A total of 682 data was generated for a particular software in the AKTC warehouse. A testing error of 0.047 in terms of MSE is obtained using a back-propagation algorithm for 171 data pairs. By combining the fuzzy logic and neural network, an automated system is designed [24] to estimate the software quality using only three inputs, e.g. complexity, DIT, and reuse. However, these parameters may not be sufficient enough, and other parameters should be considered as well. In [28], an extension of the DRM model is proposed, which is called DRM. This technique is useful to predict the software development efforts and cost. The proposed technique proved that reusability plays a vital role in software cost estimation. In [29], the authors developed a model named Adaptive Neuro-Fuzzy Inference System to

predict the software defects using linear discriminant analysis. First, the data set was balanced by the linear discriminant technique. Then it was trained by the ANFIS model. The proposed model gives faster convergence and satisfactory results.

Due to the lack of knowledge, perhaps from the domain expert, the quality attribute of software products is sometimes overlooked throughout the development stage. However, various studies have been considered for the software quality prediction using soft computing approaches and neuro-fuzzy approach also, they lack the comprehensive analysis and also don't use software metrics directly, for example, Reliability, Usability, Efficiency, Portability, Maintainability, etc.

So, A combination of using linguistic rules as well as input data can be a good idea to increase the accuracy and robustness of the prediction model. Hence, the aim of this paper is to propose a neuro-fuzzy scheme to build an automated system for software quality prediction.

3. Factors Affecting Software Quality

A thorough analysis has been accomplished in [9] on the parameters that can affect the software quality. It is found that the software quality is greatly influenced by five parameters, namely Reliability, Usability, Efficiency, Portability, and Maintainability. Thus, the following five parameters have been considered as input for the proposed neuro-fuzzy model [25].

3.1 Reliability

Failures to provide an appropriate level of service are the subject of reliability requirements. They also set the software system's maximum allowed failure rate, which might pertain to the entire system or one or more of its operations. System reliability, hardware failure recovery, application reliability, and computational failure recovery are the four subfactors of reliability.

3.2 Usability

Usability can be explained as follows: the scope of staff resources required for training a new employee as well as operating the software system is addressed by usability requirements. Operability and training are two subfactors of usability.

3.3 Efficiency

Efficiency is concerned with the hardware resources needed to carry out all of the software's operations while meeting all other requirements also.

3.4 Portability

The adaptation of software in question to other settings constituted of different operating systems, different hardware, etc., are referred to as portability requirements. Modularity, self-description, and software independence are three subfactors of portability.

3.5 Maintainability

Maintainability requirements are concerned with determining the effort necessary by all potential users and maintenance personnel to discover the causes of software failures. Modularity, simplicity, compliance (consistency), document accessibility, code and documentation rules, and self-descriptiveness are six sub-factors of maintainability.

4. Proposed Methodology

First, it is discussed the neuro-fuzzy model employed, its layout, and functioning. Here, an adaptive neuro-fuzzy inference engine (ANFIS) has been employed for the purpose of building the automated system. Although the ANFIS model was first proposed in 1993, as discussed in [26], it has not been fully explored as an automated system for software quality prediction. Takagi-Sugeno fuzzy inference engine is introduced to access the linguistic information, whereas a feed-forward neural network is introduced to embed NN information. Two approaches, grid partition and sub clustering are introduced in this paper to generate the fuzzy rules. These rules are embedded by a bell-shaped fuzzy membership function by which each of the inputs is associated. To train the ANFIS model, the MATLAB tool [27] has been utilized. Five inputs, namely Reliability, Usability, Efficiency, Portability, Maintainability, and one output as software quality, have been used for the prediction of the software quality. To derive an effective relationship between these metrics and software quality, first, a data set of 128 pieces of software was collected from various studies, which were later verified by experts. Out of 128 software data, 100 data-set have been used to train the proposed neuro-fuzzy model and 28 pieces of software for testing the ANFIS model. The fuzzy rules are needed to train the ANFIS model, so, in this paper, two approaches, namely grid partition and sub-clustering (Subtractive clustering), have been proposed. Both approaches are used to generate the fuzzy rules. The results obtained by both introduced approaches are compared in terms of performance measures for further analysis of the ANFIS architecture. The error measures MARE and MRE have been used to evaluate the performance of each of the methods. It is found that the proposed approach with sub-clustering has lesser error measures, i.e. it more accurately predicts the software quality. The proposed approach with sub-clustering has the potential of estimating software quality in a real-time scenario.

4.1 ANFIS architecture

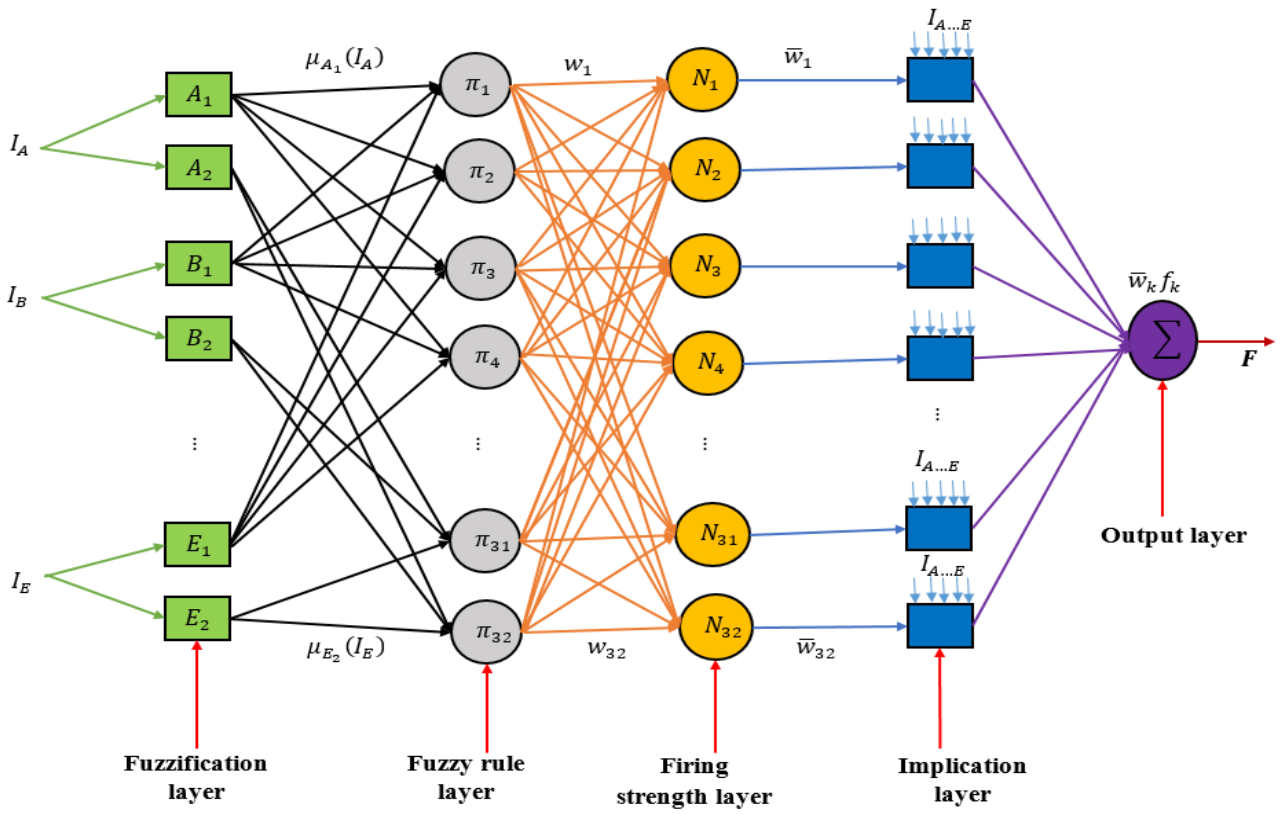

Fig. 1 A general ANFIS architecture

The architecture of the ANFIS model is shown in Fig. 1, which consists of a five-layer architecture. Five input parameters are denoted as I_A, I_B, I_C, I_D, I_E. The first layer consists of premise parameters which depend upon no. of input parameters (n). No. of nodes in the first layer can be calculated as $n \times m$ where m denotes the no. of fuzzy sets equipped with membership function for each input. Layer 2 contains all the nodes which are originated from the output of each membership function in Layer 1, and thus there are m^n no. of nodes in Layer 2. Moreover, subsequent layers, i.e. Layers 3 and 4, have the same no. of nodes as layer 2. Whereas the fifth layer has one node, which infers the output of the ANFIS model, i.e. software quality. For example, two fuzzy sets are associated with each input, as shown in Fig. 1. It is seen that the no. of nodes in layers 2, 3, and 4 are $2^5 = 32\ nodes$ whereas Layer 1 has $2 \times 5 = 10\ nodes$. The total no. of nodes in this ANFIS architecture can be calculated as follows:

$$No.\ of\ nodes = n + n \times m + m^n + m^n + m^n + 1$$
$$= 5 + 5 \times 2 + 3 \times 2^5 + 15 + 10 + 96 + 1$$
$$= 5 + 10 + 96 + 1 = 112$$

4.2 Functioning of ANFIS model

Here is the detailed discussion of the functioning of the ANFIS model given in Fig. 1 at each layer.

4.2.1 Layer 1

First Layer is the fuzzification layer which consists of input nodes associated with the appropriate fuzzy membership function. Let $\mu_{A_1}(I_A)$ is the membership output for input I_A and it expresses the extent to which a given I_A meets its quantifier A_1. Here, it is employed a bell-shaped membership function expressed in Eq. (1).

$$\mu_{A_1}(I_A) = \frac{1}{1+(\frac{I_A - c}{a})^{2b}} \quad (1)$$

Where value of $\mu_{A_1}(I_A)$ lies between 0 and 1. a, b, c decides the shape of the membership function and is refereed as premise parameters and needs to be learned here.

4.2.2 Layer 2

Layer 2 denotes the fuzzy rule layer, which outputs the product of membership value originating from each of the input sets. The output of layer 2 can be described in Eq. (2).

$$w_o = \mu_{A_i}(I_A) \times \mu_{B_j}(I_B) \times \mu_{C_k}(I_C) \times \mu_{D_l}(I_D) \times \mu_{E_p}(I_E) \quad (2)$$

Where, $o = 1, 2, \ldots 32$ and $i, j, k, l, p = 1,2$

4.2.3 Layer 3

Layer 3 is the firing strength layer which outputs the normalized value of the strength of each fired rule. Thus, the output of layer 3 can be described in Eq. (3).

$$\bar{w}_o = \frac{w_o}{\sum_{i=1}^{32} w_o} \quad (3)$$

4.2.4 Layer 4

The structure of a fuzzy rule for the proposed ANFIS model is in the following Eq. (4)

Rule 1: *if* I_A is A_1 *and* I_B is B_1 *and* I_C is C_1 *and* I_D is D_1 *and* I_E is E_1 then

$$f_1(\text{Output}) = p_1 I_A + p_2 I_B + p_3 I_C + p_4 I_D + p_5 I_E + p_6 \quad (4)$$

Wherep_i, $\{i = 1, 2, 3, 4, 5, 6\}$ are referred as the consequent parameters. Thus, Layer 4 is the implication layer, and its output is estimated using Eq. (5)

$$O_{4o\,(output)} = \bar{w}_o\, f_o \quad (5)$$

Where, f_o Is the output of $O's$ rules.

4.2.5 Layer 5

Finally, Layer 5 decides the output of the ANFIS model and is calculated using Eq. (6).

$$\text{F (Final Output)} = \sum_{i=1}^{32} \bar{w}_o\, f_o \quad (6)$$

The objective of the ANFIS is to learn the premise and consequent parameters. The working of ANFIS can be described as follows: for some initial value of premise parameters in the forward pass, the consequent parameters can be identified using least-square whereas, in the backward pass, premise parameters can be updated using the gradient-descent algorithm. Thus after training the ANFIS model, the optimal value of premise and consequent parameters has been got. The output for a new input data pair can then be calculated by using these optimal parameters.

4.3 ANFIS training

The training of the ANFIS model is done in MATLAB 2016a on a PC with a Core i7 processor running windows 8. The parameters used for training are shown in Table 1.

Table 1. Various parameters for ANFIS training using grid partition

Parameter	Value
No. of an Input signal	5
No. of Output	1
No. of fuzzy sets in each input	3
Learning method	Gradient descent Least-square algorithm
No. of iterations	300
Membership function type	Bell-shaped
Size of training data	100
Size of testing data	28

ANFIS setup in MATLAB is shown in Fig. 2, where first, training and testing data were uploaded. A fuzzy inference system (FIS) has been generated using grid partition, by which 243 fuzzy rules are obtained. After that, error tolerance is set accordingly, which has been set at 0.0 and epochs no. as 300. Then by clicking on 'train now', the proposed ANFIS model is trained. This trained model can be verified by clicking on 'test now' for both training data and testing data. The structure of the above ANFIS model in MATLAB is shown below in Fig. 3 for better understanding.

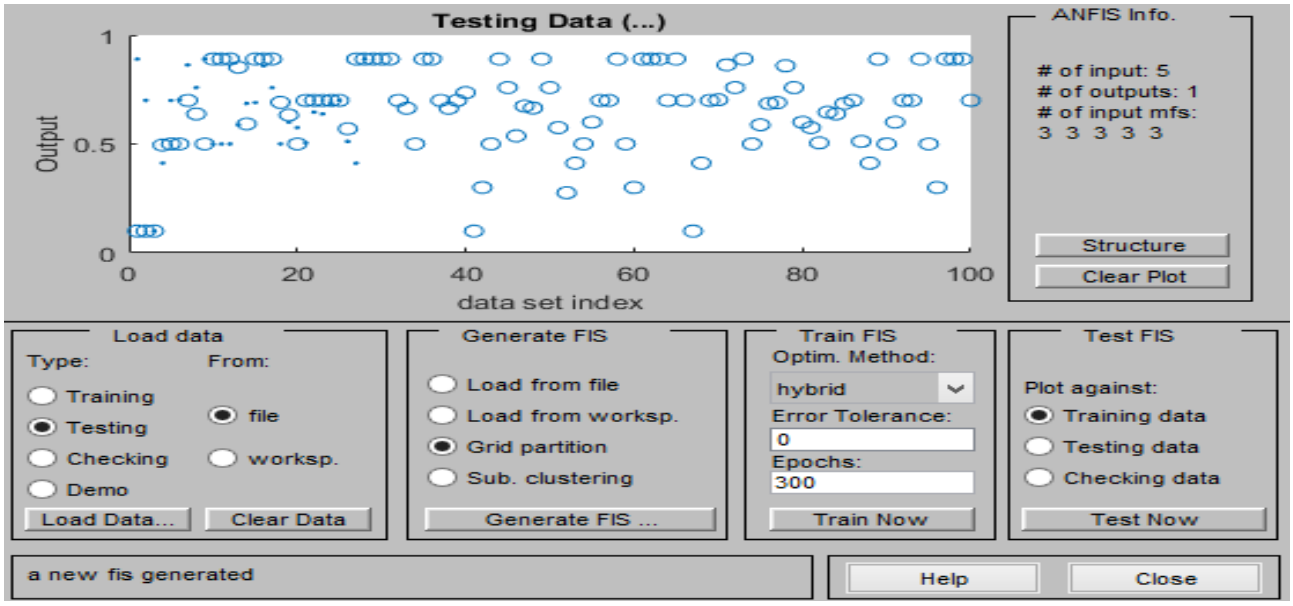

Fig. 2 ANFIS MATLAB setup

A bell-shaped membership function is used for each of the five inputs, where the structure of the membership function is shown in Fig. 4.

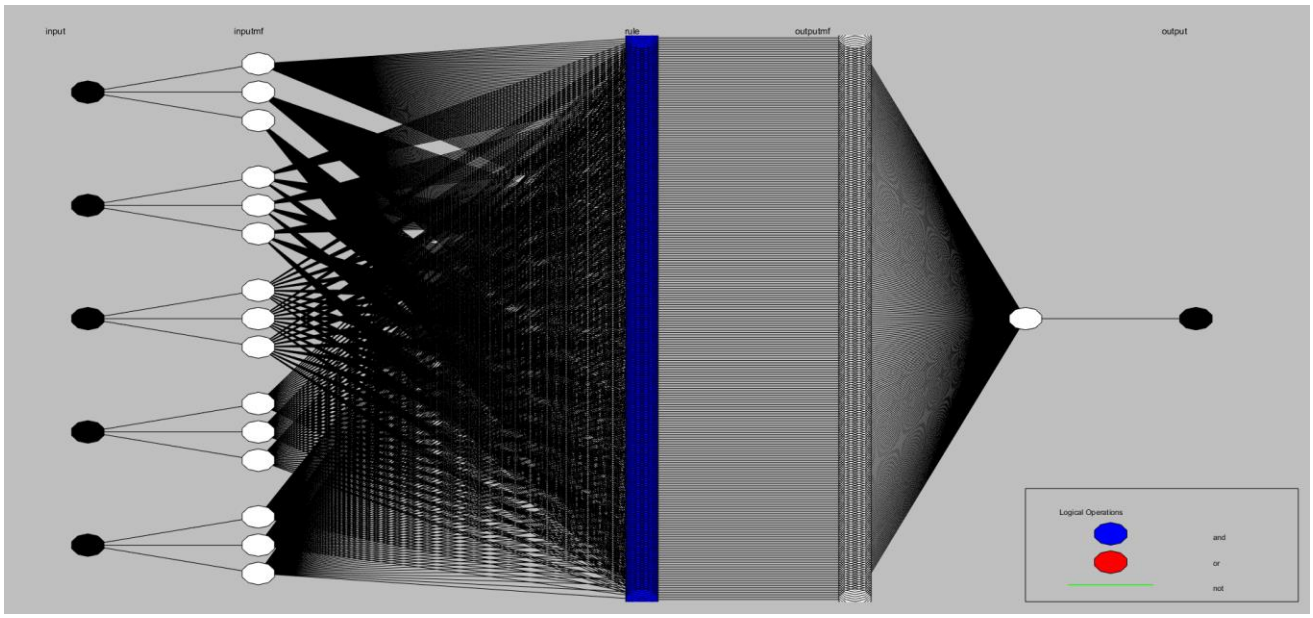

Fig. 3 ANFIS MATLAB architecture

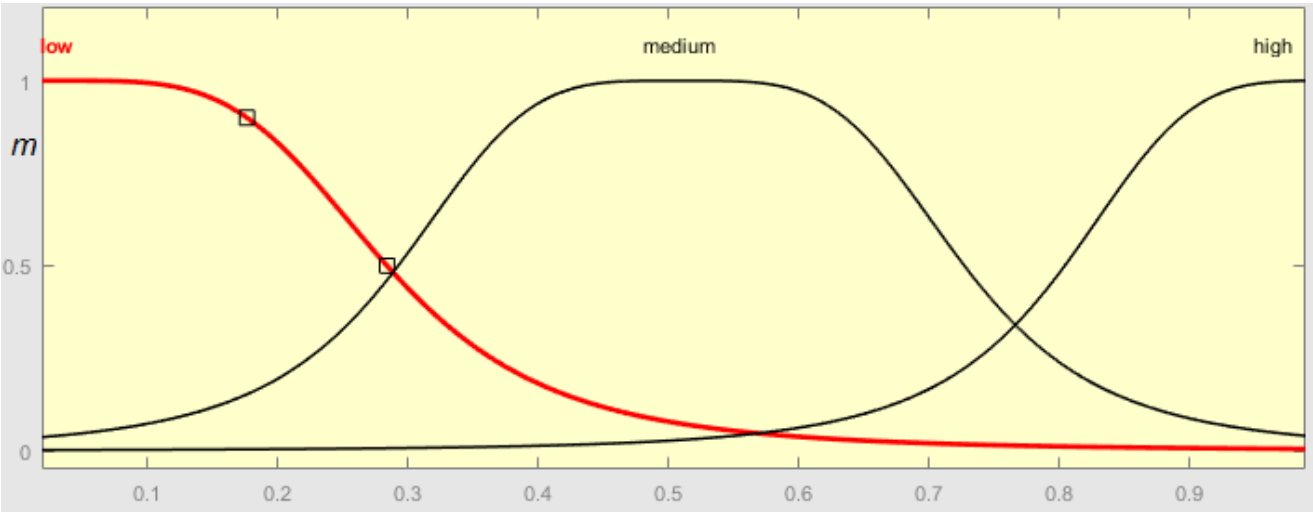

Fig. 4 Bell-shaped membership function for each input

5. Simulation Results and Discussion

After training the ANFIS model, the tests were conducted for the remaining 28 data pairs that were not used for training. Software quality was predicted using the five data pairs and compared with the actual quality. First, the accuracy of the trained model was tested in the case of training data so that it can be shown 'how effective the training was'? Both the approaches, grid partition and sub-clustering, are used to generate the fuzzy rules and compare the obtained results.

5.1 Sub-clustering based ANFIS

In the case of sub-clustering, the three rules for three membership functions associated with input data were got. Then training is performed as shown in the previous section, and obtained results are shown here.

Fig. 5 shows the target by the data set and actual software quality by the ANFIS model. It is seen that the quality predicted by the ANFIS model closely matches the actual/target software quality, which shows a good training model.

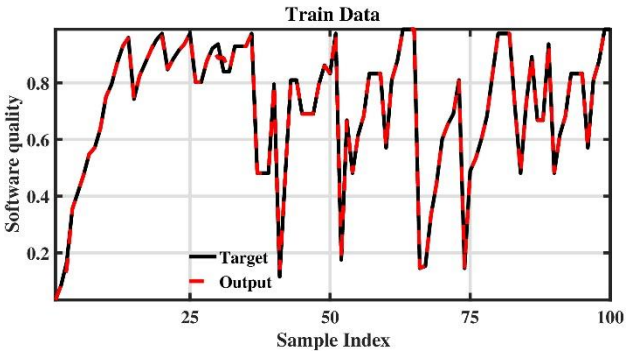

Fig. 5 ANFIS training using sub-clustering FIS

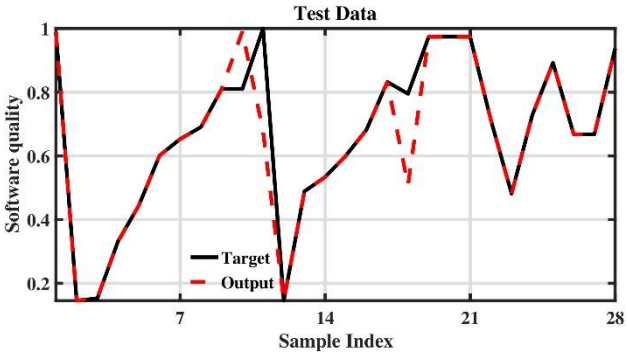

Fig. 6 ANFIS testing using sub-clustering FIS

For better visualization, the combined results of training and testing of the ANFIS model are also shown in Fig. 7.

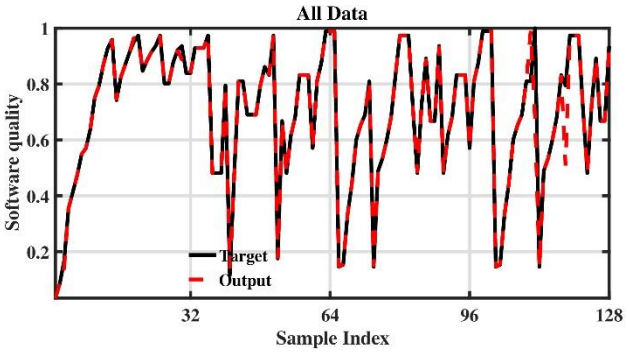

Fig. 7 ANFIS all data using sub-clustering FIS

5.2 Grid partition-based ANFIS

Using grid partition, a total of $m^n = 3^5 = 243$ fuzzy rules are generated and used in the ANFIS training. The obtained results are shown below. The training results are shown in Fig. 8, which shows the close matching of predicted quality with the target software quality. The prediction results for 28 unknown data pairs are shown in Fig. 9, where it can be noticed that the ANFIS model correctly predicts the software quality at a good level.

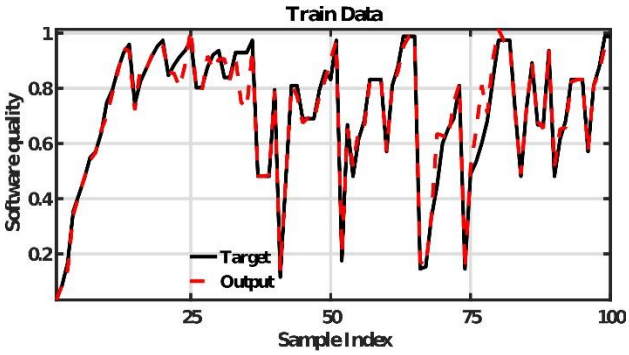

Fig. 8 ANFIS training using Grid partition FIS

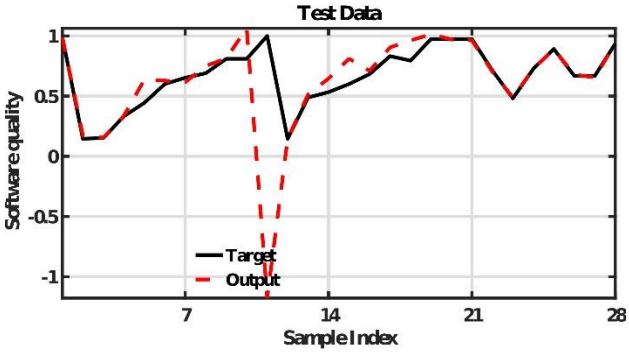

Fig. 9 ANFIS training using Grid partition FIS

All data, i.e. training and testing data, are shown in Fig. 10, along with the predicted software quality.

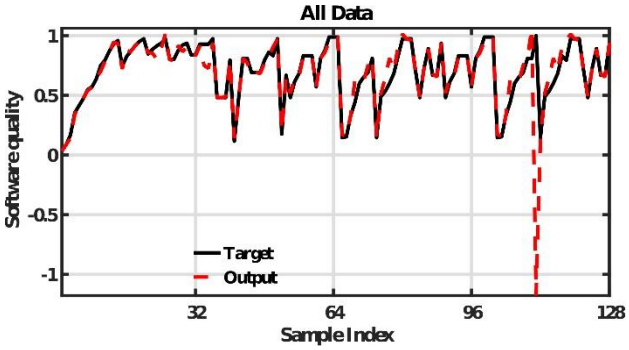

Fig. 10 ANFIS training using Grid partition FIS

5.3 Comparative Analysis

Both of the discussed training methods of ANFIS were compared in terms of error measures. The performance index in terms of error is taken as a parameter to show the accuracy of the proposed automated system. These performance indexes are MSE (mean square error), Mean Absolute Relative Error (MARE), and Mean Relative Error (MRE) given in Eq. (7), Eq. (8), and Eq. (9), respectively.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(F_d(i) - F(i))^2 \quad (7)$$

$$MARE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{F_d(i)-F(i)}{F(i)}\right| \quad (8)$$

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{F_d(i)-F(i)}{F(i)}\right) \quad (9)$$

where F_d Is the target software quality and F is the predicted software quality by the ANFIS model, and N is the total no. of samples taken.

The obtained value of these performance measures are shown in Table 2, Table 3 and Table 4 for MSE, MRE and MARE, respectively. It is noticed that sub-clustering achieves lower MSE than the grid-partition in both the training and testing phases. Furthermore, MARE and MRE scores are also achieved as lower by the sub-clustering approach than the grid partition approach.

Table 2. Comparison of MSE between two methods

S. No.	ANFIS Method	Mean Square Error (MSE) Training	Testing
1	Grid-partition	0.0024	0.1766
2	Sub-clustering	0.000065	0.0078

Table 3. Comparison of MRE between two methods

S. No.	ANFIS Method	Mean Relative Error (MRE) Training	Testing
1	Grid-partition	-0.0091	0.0199
2	Sub-clustering	-0.000075	0.0160

Table 4. Comparison of MARE between two methods

S. No.	ANFIS Method	Mean Absolute Relative Error (MARE) Training	Testing
1	Grid-partition	0.0434	0.1576
2	Sub-clustering	0.0053	0.0329

Graphical analysis is also performed for both the approaches, and the results are shown in Fig. 11 and Fig. 12, where it has been noticed that sub-clustering has lesser performance measures while testing, as testing is a crucial part of the model development, sub-clustering achieves lower error measures.

Overall, it can be stated that the sub-clustering approaches have good performance in both the training and testing phases and can be used in real-time analysis.

An analysis of the time taken to train the ANFIS using both approaches was also performed to show a clearer picture of the computation complexity of both techniques. It is identified that sub-clustering takes around 52 seconds to train the ANFIS for three membership functions with input data. If it is used five membership functions, MATLAB crashes, and the PC hangs without any output. At the same time, sub-clustering takes around 77 seconds to train the ANFIS, larger than the grid partition for an influence radius of 0.03. It takes a long time if it is increased epochs or further lower the radius. Both the approaches are performed for 300 epochs.

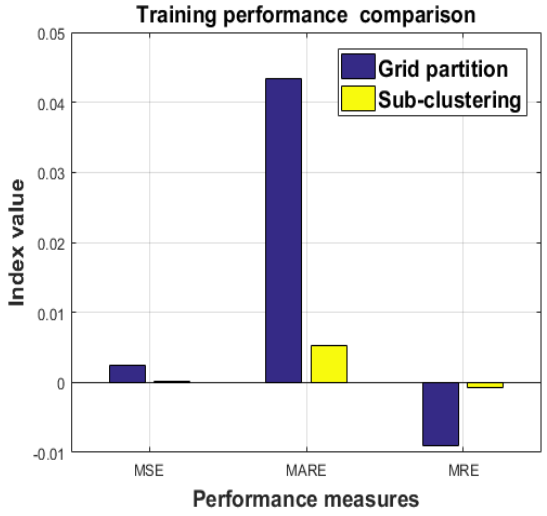

Fig. 11 ANFIS training using Grid partition FIS

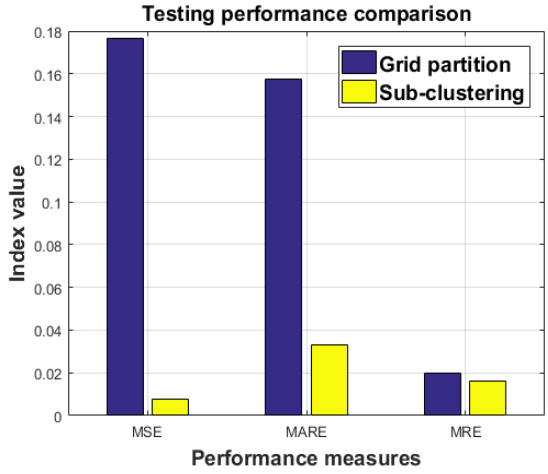

Fig. 12 ANFIS training using Grid partition FIS

From the above discussion, it can be concluded that the proposed sub-clustering takes a comparatively longer time than the proposed grid-partition approach, but the accuracy is higher in the case of the proposed sub-clustering. It is because the addition of the linguistic rule increases the function approximation capability and accuracy of a neural network model.

6. Conclusion and Future Scope

This paper presents an automated system to predict software quality based on five inputs, namely Reliability, Usability, Efficiency, Portability, and Maintainability. A combination of neural network and fuzzy logic, namely the ANFIS model, is used to build the automated system. To generate the fuzzy rules, two approaches, namely sub-clustering and grid partition, have been introduced. The ANFIS architecture is trained for 100 data pairs while updating premise parameters and consequent parameters with the help of a gradient descent algorithm and least-square approach. After training, the optimal value of the premise parameters and consequent parameters are obtained, which is further used to predict the software quality for an unknown data pair. The proposed approach was tested for 28 data pairs and found that the proposed approach correctly predicts the software quality for an unknown data set.

Moreover, the results obtained with both the approaches, i.e. sub-clustering and grid partition, are compared in terms of the performance measures. It is found that the grid partition takes a longer time for training if the no. of membership function with the input data is increased or with an increment of the no. of epochs. On the other hand, the sub-clustering approach infers lesser fuzzy rules but can take more time for training if the influence radius is lower. Finally, it can be concluded that the sub-clustering method can be used to obtain better accuracy if the training time is not of great interest.

In the future, the type 2 Neuro-fuzzy model can be incorporated with feedback-neural networks like recurrent neural networks, LSTM etc.

References

[1] B. W. Boehm, J. R. Brown, and M. Lipow, Quantitative evaluation of software quality, in Proc. of 2nd international conference on Software engineering, (1976) 592-605.

[2] A. Monden, D. Nakae, T. Kamiya, S. I. Sato, and K. I. Matsumoto, Software quality analysis by code clones in industrial legacy software, in Proc. of 8th IEEE Symposium on Software Metrics, (2002) 87-94.

[3] Z. H. Zhou, Neural networks, in Proc. of Machine Learning, Springer, Singapore. (2021) 103-128.

[4] T. J., Ross, Fuzzy Logic With Engineering Applications, John Wiley & Sons. (2005).

[5] C. Catal, Software Fault Prediction: A Literature Review and Current Trends, Expert Systems with Applications, 38(4) (2011) 4626-4636.

[6] J., Klomjit, and A., Ngaopitakkul, Comparison of Artificial Intelligence Methods for Fault Classification of the 115-kV Hybrid Transmission System, Applied Sciences, 10(11) (2020).

[7] S. Yadav, and B. Kishan, Analysis and Assessment of Existing Software Quality Models to Predict the Reliability of Component-based Software, International Journal of Emerging Trends in Engineering Research, 8(6) (2020) 2824-2840.

[8] J. Paul and V. Bhattacherjee, Software Quality Prediction using Fuzzy Rule-Based System, International Journal of Current Research, 7(12) (2015) 24181-24185.

[9] Ritu, and O. P. Sangwan, Software Quality Prediction Method using Fuzzy Logic, Turkish Journal of Computer and Mathematics Education, 12(11) (2021) 807-817.

[10] S. S. So, S. D. Cha, and Y. R. Kwon, Empirical Evaluation of a Fuzzy Logic-based Software Quality Prediction Model, Fuzzy Sets and Systems, 127(2) (2002) 199-208.

[11] F. He, A. Ren, and Z. Ding, Software Quality Prediction Model Research Based on Object-Oriented Petri Nets, International Journal of Electronics and Electrical Engineering, 3(3) (2015) 225-229.

[12] A. Rai, T. Choudhury, S. Sharma, and K. C. Ting, An efficient method to predict software quality using soft computing techniques, in Proc. of 3rd International Conference on Applied and Theoretical Computing and Communication Technology, (2017) 347-353.

[13] A. Sandanasamy and R. T. Selvi, A Quality-Based Software Requirement Prioritization Using TakagiSugeno Neuro-Fuzzy Inference, International Journal of Computer Sciences and Engineering, 7(5) (2019).

[14] K. Sahu and R. K. Srivastava, Soft Computing Approach for Prediction of Software Reliability, ICIC Express Letters, 12(12) (2018) 1213-1222.

[15] C. Diwaker, P. Tomar, R. C. Poonia, and V. Singh, Prediction of Software Reliability using Bio-inspired Soft Computing Techniques, Journal of Medical Systems, 42(5) (2018) 1-16.

[16] N. Li, M. Shepperd, and Y. Guo, A Systematic Review of Unsupervised Learning Techniques for Software Defect Prediction, Information and Software Technology, 122 (2020).

[17] S. K. S. Durai, B. Duraisamy, and J. T. Thirukrishna, Fuzzy Interference System based Link Failure Prediction in MANET, in Proc. of Journal of Physics: Conference Series, 1964(7) (2021).

[18] W. Peng, L. Yao, and Q. Miao, An approach of software quality prediction based on relationship analysis and prediction model, in Proc. of 8th International Conference on Reliability, Maintainability and Safety, (2009) 713-717.

[19] Q. Wang, B. Yu, and J. Zhu Extract rule from software quality prediction model based on neural network, in Proc. of 16th IEEE International Conference on Tools with Artificial Intelligence, (2004) 191-195.

[20] K. Sheoran, P. Tomar, and R. Mishra, Software Quality Prediction Model with The Aid of Advanced Neural Network with HCS, Procedia Computer Science, 92 (2016) 418-424.

[21] A. Barzegar, and Y. Barzegar, Adaptive Neuro-Fuzzy Inference System for Measuring Software Quality Product, 2021.

[22] Sharma, Pragati, Software Quality Prediction using Hybrid Approach, International Journal of Computer Applications, 180(4) (2017) 0975–8887.

[23] S. Pattnaik, B. K. Pattanayak, and S. Patnaik, Prediction of Software Quality using Neuro-fuzzy Model, International Journal of Intelligent Enterprise, 5(3) (2018) 292-307.
[24] S. Sahar, and U. Qamar, and S. Ayaz, Multilayer Neural Network and Fuzzy Logic Based Software Quality Prediction, International Journal of Computer and Systems Engineering, 11(9) (2017) 1024-1028.
[25] B. S. Dhillon, Applied Reliability and Quality: Fundamentals, Methods and Procedures, Springer Science & Business Media, 2007.
[26] J. S. Jang, ANFIS: Adaptive-network-based Fuzzy Inference System, IEEE Transactions on Systems, Man, and Cybernetics, 23(3) (1993) 665-685.
[27] Y. Gershteyn, and L. Perman, Matlab: ANFIS Toolbox, The MathWorks, 2003.
[28] P. Mangayarkarasi, and R. Selvarani, A Novel Software Cost Estimation Technique: Inclusion of Reusability, International Journal of Engineering Trends and Technology, 44(1) (2017) 42-47.
[29] A. Thakur, and A. Goel, A Hybrid Neuro-Fuzzy Approach for Bug Prediction using Software Metrics, International Journal of Engineering Trends and Technology, 38(2) (2016) 85-92.