SOM-FTS: A Hybrid Model for Software Reliability Prediction and MCDM-Based Evaluation

Ajay Kumar*, Kamaldeep Kaur

University School of Information, Communication, and Technology (USICT), Guru Gobind Singh Indraprastha University, New Delhi, India

Received 23 September 2021; received in revised form 21 January 2022; accepted 22 January 2022

DOI: https://doi.org/10.46604/ijeti.2022.8546

Abstract

The objective of this study is to propose a hybrid model based on self-organized maps (SOM) and fuzzy time series (FTS) for predicting the reliability of software systems. The proposed SOM-FTS model is compared with eleven traditional machine learning-based models. The problem of selecting a suitable software reliability prediction model is represented as a multi-criteria decision-making (MCDM) problem. Twelve software reliability prediction models, including the proposed SOM-FTS model, are evaluated using three MCDM methods, four performance measures, and three software failure datasets. The results show that the proposed SOM-FTS model is the most suitable model among the twelve software reliability prediction models on the basis of MCDM ranking.

Keywords: software reliability, multi-criteria decision making, WSM, TOPSIS, EDAS

1. Introduction

Human society has become extremely dependent on reliable software systems for economic and social activities, and the software industry has to respond rapidly to ever-changing business and social environments. An important question for software industry professionals is how much testing should be done and when it is appropriate to release the software so that users do not face any software failures when using the software, as failures may cause huge losses to human life and business. To aid software industry professionals in this critical decision-making problem, researchers have proposed many software reliability predictive models for estimating the software testing resources and release time. As per IEEE literature, the definition of software reliability is the "probability with which the software has an operation without experiencing any failures for a measured time and under unambiguously specified operating environment" [1]. Therefore, achieving high software reliability through understandable and accurate models for software reliability is a key agenda for software quality researchers [1].

In the initial seminal works on the development of software reliability models, the emphasis was to represent the failure phenomenon during the testing process as a non-homogeneous poison process (NHPP) [2]. This resulted in extreme difficulty to validate underlying assumptions about software failure phenomenon and led to the introduction of non-parametric non-linear tools such as artificial neural networks (ANNs), which could generate a software reliability model from historical data of observed failures [3]. However, ANNs have the problem of getting stuck in a local minimum and slow convergence, which leads the software reliability researchers to explore ensemble and hybrid techniques for estimating software reliability. Current initiatives in software reliability research are directed towards more accurate hybrid techniques based on integrating ANNS with other computational intelligence techniques [3-4, 5-6].

* Corresponding author. E-mail address: ajaygarg100@gmail.com
Tel.: +91-9999350881

This study contributes to the problem of developing accurate software reliability models based on the hybridization of self-organizing maps (SOM) and fuzzy time series (FTS). The prediction capabilities of the intelligent techniques built around the state-of-the-art machine and deep learners are highly varied depending on the datasets of different software products and the prediction accuracy or model performance measures. It is difficult to find an intelligent technique that performs better than all other techniques across all conflicting model performance measures for a particular application domain. Thus, the selection of an optimal software reliability modeling technique is challenging. To get around this challenge, a multi-criteria decision-making (MCDM)-based approach is used in this study.

The organization of this study is as follows. Section 2 describes the related work and highlights the contribution of this study in software reliability prediction. Section 3 presents the hybrid SOM-FTS approach, the performance measures, and the MCDM methods used in this study. Section 4 provides the datasets used in this study. Section 5 discusses the experimental results. Finally, section 6 concludes the study.

2. Related Work

A considerable body of research work is dedicated to the development and assessment of deep neural-based techniques for software reliability prediction. Related work is summarized as follows. Roy et al. [5] proposed a software reliability prediction model by incorporating neighborhood particle swarm optimization (PSO) in an ANN to determine the optimal weights of the network. They concluded that the neighborhood PSO-ANN approach performed better than the standard PSO-ANN approach and singular ANN approach on the average error performance metric. However, it is difficult to determine the optimal architecture of ANNs in terms of the number of layers in the network and the number of neurons in each layer.

Rani and Mahapatra [6] developed a hybrid model using a feed-forward neural network (FFNN) and PSO algorithm for software reliability prediction. Their experimental study determined that the hybrid model (FFNN-PSO) can be used as an efficient method for software reliability prediction.

Wang et al. [7] proposed a deep learning model, namely DNN-RED, using a recurrent neural network (RNN) encoder-decoder for predicting software failure counts and assessing the reliability of a software system. They compared their proposed DNN-RED model with various software reliability models, including parametric models (i.e., an exponential model, a logarithmic model, a delayed S-shape model, an inverse polynomial model, and a power model) and non-parametric models (i.e., FFNN-Prediction, FFN-Generalization, JordanNet-Prediction, and JordanNet-Generalization), considering the average error as evaluation criteria. The experimental study concluded that the DNN-RED model outperformed all the models used in their comparative study.

For enhancing the prediction accuracy of existing software reliability growth models (SRGMs), Jabeen et al. [8] proposed a high precision error iterative analysis method (HPEIAM). They combined the residual errors obtained from the estimated results of SRGMs with the ANN sign estimator for enhancing the prediction accuracy of SRGMs. After applying HPEIAM on various SRGMs (J-M, GO, Littlewood, Musa, and Jinyong-GO), they concluded that HPEIAM enhanced the performance of these traditional SRGMs models.

Zhen et al. [9] developed a hybrid model by integrating two swarm intelligence algorithms, namely the wolf pack algorithm (WPA) and PSO, for estimating the parameters of software reliability models. They applied the WPA-PSO model to estimate the parameters of the classical GO model. They concluded that the hybrid model WPA-PSO could improve the parameter estimation and prediction of traditional SRGMs.

In a more recent study, Kassaymeh et al. [10] proposed a hybrid model for software reliability prediction by integrating the salp swarm algorithm (SSA) with backpropagation neural network (BPNN) to determine the optimal weights of the network. After a comparative study based on various performance measures, they concluded that the hybrid model SSA-BPNN performed better than the BPNN.

Pai and Hong [11] investigated the support vector regression (SVR) technique for software reliability prediction. They concluded that SVR performed better than Kalman filter-based models. However, the determination of SVR parameters like penalties for estimation errors and kernel bandwidth itself are open research areas, and there is no simple way to determine these parameters.

Lou et al. [12] applied relevance vector machine (RVM) for software reliability prediction. Although RVM shares functional similarities with support vector machine (SVM), it is a probabilistic model. RVM requires more training time than SVM because of the optimization of a non-convex function, but it does not require the use of free parameters. Regardless of the advantages of RVM over SVM, it requires the selection of a number of kernel parameters that vary according to the type of kernel used [10].

Zhang et al. [13] proposed a framework to evaluate software reliability by applying software metrics to RNN. They evaluated their proposed model on two open-source projects and concluded that the RNN-based approach enhanced the prediction ability of traditional SRGMs.

Mohammed et al. [14] developed a software reliability prediction model using time series and machine learning. To improve software reliability prediction, they explored the ability of machine learning techniques to learn from past experience and to predict future patterns. Based on their experimental results, they concluded that SVM outperformed the random forest (RF), linear discriminant analysis (LDA), k-nearest neighbors (KNN), autoregressive integrated moving average (ARIMA) (1,1,1), and classification and regression trees (CART) models.

Clearly, there are no silver bullet intelligent techniques for software reliability prediction, and it is imperative to explore competitive and alternative techniques. In addition to the high accuracy of a predictive model, the efficiency, robustness, and ease of interpretation for intelligent techniques are also desired. This motivates researchers to explore alternate techniques for software reliability prediction, which are based on a hybrid of intelligent techniques in different categories.

The motivation for applying an FTS approach is that fuzzy systems are easily interpretable in contrast to neural networks, which are regarded as black boxes and very difficult to interpret. In recent research, Kumar and Kaur [15] demonstrated the efficiency of applying the hybrid SOM-FTS technique to a complex time series problem.

This study presents an empirical study to investigate the SOM-FTS technique for software reliability prediction. The performance of predictive modeling techniques varies depending on different performance measures [16]. It is important to select a modeling technique that is optimal when considering all performance measures. To deal with the issue of conflicting performance across various measures, this study also presents an MCDM-based approach for the evaluation and selection of software reliability prediction models. The MCDM-based approach can be a valuable decision support system for software quality assurance teams to aid the selection of the most appropriate software reliability prediction model.

3. Research Methods

3.1. Proposed hybrid model for software reliability prediction

SOM is an unsupervised two-layered neural network based upon the principle of competitive learning [17]. The first layer is designated as the input layer, and the second layer is designated as the output layer. The output layer is also called a feature map. Usually, the output layer of SOM is a one-dimensional or two-dimensional lattice of neurons. The crux of the SOM neural technique is to map each training vector onto a feature space. SOM tries to visualize the similarity between data vectors within a low-dimensional feature space.

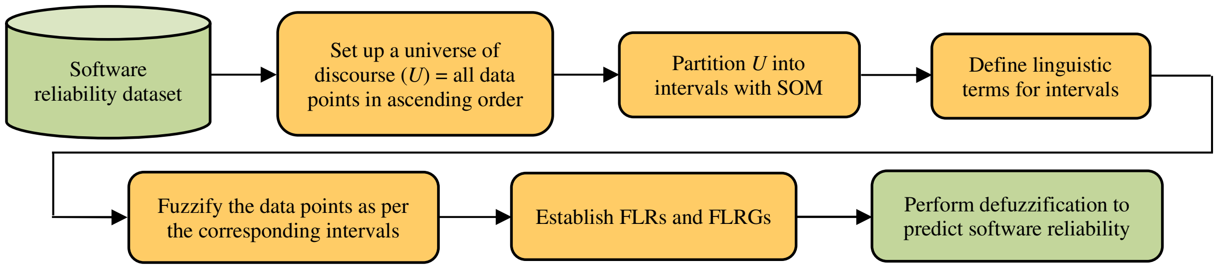
Software reliability dataset
Set up a universe of discourse (U) = all data points in ascending order
Partition U into intervals with SOM
Define linguistic terms for intervals
Fuzzify the data points as per the corresponding intervals
Establish FLRs and FLRGs
Perform defuzzification to predict software reliability

Fig. 1 Brief description of the proposed software reliability prediction model

In the context of this study, the SOM technique is integrated with FTS to predict software reliability. The SOM technique is applied to effectively fuzzify the time between failure (TBF) datasets of software systems. In this study, SOM is implemented on MATLAB 2021a version 9.10. The Euclidean distance is used as the distance measure for feature selection, and an array of size 10 × 10 is taken as the size of the output neurons. Values of additional parameters such as the number of epochs, learning radius, and learning rate are taken as 200, 3, and 0.5 respectively. The hybrid SOM-FTS technique can be well understood in Fig. 1.

The detailed stepwise process is explained as follows.

Step 1: TBF series dataset is taken as the input for SOM-FTS.

Step 2: The minimum and maximum values are obtained from the software reliability dataset. Also, a universe of discourse $U = [p, q]$ is set up, where p is the minimum value of the dataset and q is the maximum value of the dataset.

Step 3: The SOM procedure given in the work of Singh [17] is used to divide U into intervals. SOM is applied as an unsupervised learning algorithm for clustering on the universe of discourse U to obtain the number of partitions of U, which can be considered n number of intervals of different lengths as f_0, f_1, f_2, ……, and f_{n-1}. Now all the time series values of the software reliability dataset are assigned to their respective intervals. Each interval centroid can be calculated by averaging all values which belong to the corresponding interval.

Step 4: Linguistic terms are defined for the intervals obtained from Step 3 to make them fuzzy sets. n linguistic terms F_0, F_1, F_2,……, and F_{n-1} are defined for n number of intervals f_0, f_1, f_2, ……, and f_{n-1} respectively. Fuzzy sets for these linguistic terms are defined as follows:

$$\begin{aligned}
&F_0 = 1/f_0 + 0.5/f_1 + 0/f_2 + ... + 0/f_{n-3} + 0/f_{n-2} + 0/f_{n-1}, \\
&F_1 = 0.5/f_0 + 1/f_1 + 0.5/f_2 + ... + 0/f_{n-3} + 0/f_{n-2} + 0/f_{n-1}, \\
&F_2 = 0/f_0 + 0.5/f_1 + 1/f_2 + ... + 0/f_{n-3} + 0/f_{n-2} + 0/f_{n-1}, \\
&. \\
&. \\
&F_{n-1} = 0/f_0 + 0/f_1 + 0/f_2 + ... + 0/f_{n-3} + 0.5/f_{n-2} + 1/f_{n-1}
\end{aligned} \tag{1}$$

where the interval f_i has the maximum degree of membership of the fuzzy set F_i and $0 \le i \le n$.

Step 5: The fuzzified values are assigned to the data points of the software reliability dataset. The triangular membership function is used to fuzzify the software reliability dataset. In Eq. (1), for example, the value of membership degree of interval f_0 in fuzzy sets F_0 and F_1 are 1 and 0.5, respectively, and the membership value is 0 for the remaining fuzzy sets. The degree of membership values of each fuzzy set is considered 0, 0.5, or 1 for easy calculation.

Step 6: Fuzzy logical relationships (FLRs) are established for the software reliability dataset. FLRs can be established between two consecutive fuzzified values of software reliability datasets by using the definition of FLR given in the work of Song et al. [18]. Considering that $F(t\text{-}1) = F_i$ and $F(t) = F_j$ are two consecutive fuzzy values, the FLR between these two consecutive fuzzy values can be represented as follows:

$$F_i \to F_j \tag{2}$$

where F_i and F_j are the previous state and current state of FLR.

Step 7: Fuzzy logical relationship groups (FLRGs) are constructed for the software reliability dataset. According to the definition of FLRGs given by Chen [19], FLRs having the same previous state can be grouped together in the same FLRG. The following FLRs are considered.

$$\begin{aligned} &F_i \to F_{k1} \\ &F_i \to F_{k2} \\ &F_i \to F_{k3} \\ &. \\ &. \\ &F_i \to F_{km} \end{aligned} \tag{3}$$

Now, these FLRs can be grouped together in the same FLRG as:

$$F_i \to F_{k1},\ F_{k2},\ F_{k3},\ \ldots\ldots,\ F_{km} \tag{4}$$

Step 8: The defuzzification technique used in the work of Chen [19] is applied to predict the software reliability from the fuzzified values of the software reliability dataset. In this step, the defuzzification technique is applied on the FLRGs obtained from Step 7 for the defuzzification of the fuzzified value of the software reliability dataset and is finally used to predict software reliability. For the prediction of software reliability at the time t, the fuzzified value at the time (t-1) is required. The defuzzification is done as follows.

Case 1: This case is applicable when the current state has more than one fuzzified value. A detailed stepwise explanation of this case is given below:

Step i: The $Y(t\text{-}1)$ fuzzified software reliability value is obtained at the time (t-1) as F_i.

Step ii: The FLRG of the form "$F_i \to F_{j0}, F_{j1}, \ldots\ldots, F_{jp}$" is obtained.

Step iii: The intervals $f_{j0}, f_{j1}, \ldots\ldots, f_{jp}$ are obtained for the maximum membership value of fuzzy sets $F_{j0}, F_{j1}, \ldots\ldots, F_{jp}$ respectively, and the centroids $c_{j0}, c_{j1}, \ldots\ldots, c_{jp}$ are obtained for these intervals.

Step iv: *Fpredict* is computed by using the following formula.

$$Fprecdict = \frac{c_{j0} + c_{j1} + \ldots\ldots + c_{jp}}{p} \tag{5}$$

where p denotes the total number of fuzzy sets in the current state of FLRG.

Case 2: This case is applicable when the current state has only one fuzzified value. A detailed stepwise explanation of this case is given below:

Step i: The $Y(t\text{-}1)$ fuzzified software reliability value is obtained at the time (t-1) as F_i.

Step ii: The FLRG of the form "$F_i \to F_j$" is obtained.

Step iii: Software reliability is predicted using the following formula.

$$Fprecdict = C_j \tag{6}$$

where C_j denotes the centroid of interval f_j corresponding to the maximum membership value of fuzzy set F_j.

Case 3: This case is applicable when the previous state fuzzified value is not present in FLRGs. A detailed explanation of this case is given below:

Step i: The $Y(t\text{-}1)$ fuzzified software reliability value is obtained at the time (t-1) as F_i.

Step ii: As there is no rule in FLRGs corresponding to F_i as the previous state, the centroid of interval f_i is taken as the predicted value.

3.2. *Other software reliability modeling techniques*

In this study, other eleven modeling techniques are also evaluated in addition to the proposed SOM-FTS model for a fair comparison with a wide range of modeling techniques. Among these eleven techniques, two are classical techniques (i.e., ARIMA and multiple linear regression (MLR)), six are based on machine learning (i.e., multilayer perceptron (MLP), SVR, additive regression (AR), bagging, regression by discretization (RegbyDisc), and decision table (DT)), and the rest are the FTS forecasting models proposed by Yu [20], Cheng et al. [21], and Efendi et al. [22].

3.3. *Performance measures*

To evaluate the twelve software reliability modeling techniques, four performance measures are used as evaluation criteria. Two are cost criteria, and two are beneficial criteria. For cost criteria, minimization is desired. For beneficial criteria, maximization is desired. The performance measures are described as follows.

(1) Cost criteria:

The cost criteria include normalized root mean square error (NRMSE) and standard deviation of absolute residual error (SdARE). NRMSE is expressed as Eq. (7) and the absolute residual error (ARE) is given by Eq. (8). u_i and v_i are the actual value and predicted value, and n is the number of data points.

$$NRMSE = \sqrt{\frac{\sum_{i=1}^{n}(u_i - v_i)^2}{\sum_{i=1}^{n}u_i^2}} \tag{7}$$

$$ARE = \|u_i - v_i\| \tag{8}$$

(2) Beneficial criteria:

The beneficial criteria include correlation coefficient (r) and Pred (0.20). Correlation coefficient (r) gives the strength of the relationship between two variables (i.e., the actual value and predicted value in this study). Pearson's correlation coefficient is used and can be calculated using the following equation.

$$r = \frac{n\left(\sum uv\right) - \left(\sum u\right)\left(\sum v\right)}{\sqrt{\left[n\sum u^2 - \left(\sum u\right)^2\right]\left[n\sum v^2 - \left(\sum v\right)^2\right]}} \tag{9}$$

where u and v are the actual and predicted values and n is the number of observations.

Pred (0.20) is the number of predicted values for which the magnitude of relative error (MRE) ≤ 0.20 is divided by the number of observations. MRE is given by Eq. (10).

$$MRE = \|u_i - v_i\| / u_i \tag{10}$$

3.4. MCDM-based evaluation of software reliability modeling techniques

This section describes the MCDM methods used for the evaluation of twelve models, including the proposed model SOM-FTS, as described in section 3.1 and section 3.2. These twelve modeling techniques are applied on the three software reliability datasets, Musa-1 (DS1) [23], Musa-2 (DS2) [23], and Iyer and lee (DS3) [24], to obtain the values of four performance measures. Fig. 2 demonstrates the entire approach followed for the MCDM-based evaluation of software reliability modeling techniques.

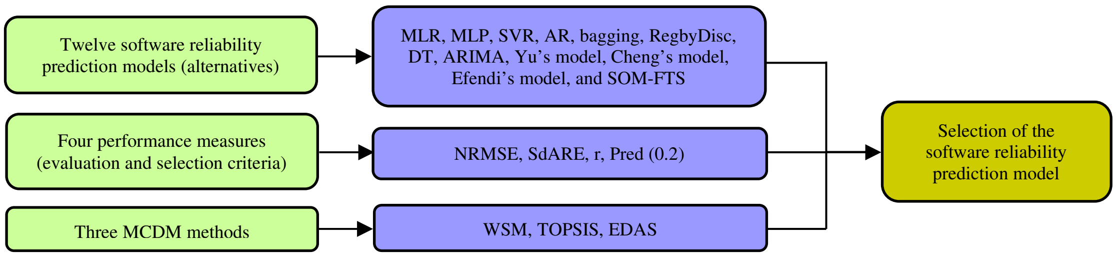
Twelve software reliability prediction models (alternatives)
MLR, MLP, SVR, AR, bagging, RegbyDisc, DT, ARIMA, Yu's model, Cheng's model, Efendi's model, and SOM-FTS
Four performance measures (evaluation and selection criteria)
NRMSE, SdARE, r, Pred (0.2)
Three MCDM methods
WSM, TOPSIS, EDAS
Selection of the software reliability prediction model

Fig. 2 MCDM approach for evaluating the software reliability prediction models

This study chooses three MCDM methods, namely the weighted sum model (WSM) [25], technique for order preference by similarity to ideal solution (TOPSIS) [26], and evaluation based on distance from average solution (EDAS) [27], for the evaluation of software reliability prediction models. These MCDM methods use a decision matrix, say $D_{a\times c}$, as an input, where a represents the number of alternatives (software reliability prediction models) and c represents the number of criteria (performance measures). In this matrix, each entry D_{ij} represents the value of a performance measure j for the corresponding software reliability prediction model i. Further detailed procedure of three MCDM methods is explained as follows.

3.4.1. Weighted sum model (WSM)

Step 1: The normalized decision matrix $ND_{a\times c}$ is calculated by using the following formula.

$$ND_{a\times c} = \frac{D_{ij}}{\sqrt{\sum_{i=1}^{a} D_{ij}^{2}}} \tag{11}$$

Step 2: The total benefit and total cost of twelve software reliability prediction models (alternatives) are calculated using the following equations.

$$A_i^{\text{total benifit}} = \sum_{j=1}^{m} w_j ND_{ij} \tag{12}$$

$$A_i^{\text{total cost}} = \sum_{j=1}^{n} w_j ND_{ij} \tag{13}$$

where w_j, m, and n represent the weight of criteria j, the number of beneficial criteria, and the number of cost criteria, respectively. In this study, the value of m and n is two. Correlation coefficient (r) and Pred (0.2) are beneficial criteria. NRMSE and SdARE are cost criteria.

Step 3: The following equation is used for calculating the WSM score.

$$A_i^{\text{wsm-score}} = A_i^{\text{total benifit}} - A_i^{\text{total cost}} \tag{14}$$

Step 4: The software reliability prediction models are ranked based on the WSM score. The model with the highest score is considered the best.

3.4.2. Technique for order preference by similarity to ideal solution (TOPSIS)

Step 1: The weighted normalized decision matrix is calculated by multiplying all the values in each column of the normalized decision matrix $ND_{a\times c}$ (obtained from Eq. (11) as described in section 3.4.1) by the corresponding weight of each criterion.

Step 2: The positive ideal value and negative ideal value are obtained for each performance measure (criterion). In the case of negative criteria (also called cost criteria), the minimum value will be the positive ideal value, and the maximum value will be the negative ideal value. In the case of positive criteria (also called beneficial criteria), the maximum value will be the positive ideal value, and the minimum value will be the negative ideal value.

Step 3: The Euclidean distance of each alternative is calculated from the positive ideal solution and the negative ideal solution.

Step 4: In this step, a score of performance is calculated for each software reliability modeling technique (alternative). The score is expressed as a ratio of the distance of each alternative from the negative ideal solution to the difference between the distance from the negative ideal solution and the distance from the positive ideal solution.

Step 5: Finally, each software reliability prediction model (alternative) is ranked according to its performance score, where the highest score is ranked as the best.

3.4.3. Evaluation based on distance from average solution (EDAS)

In this method, the best alternative is selected on the basis of distance from the average solution (AVG). Two measures, namely positive distance from average solution (PDAVG) and negative distance from average solution (NDAVG), play a key role in selecting the best alternative. An alternative with a higher value of PDAVG and a lower value of NDAVG is considered a superior solution than the AVG. The detailed procedure is described as follows.

Step 1: The AVG for performance measure j is calculated by using the following equation.

$$AVG_j = \frac{\sum_{i=1}^{a} D_{ij}}{a} \tag{15}$$

where D_{ij} represents the value of performance measure j for the corresponding software reliability prediction model i, and a represents the number of software reliability prediction models. In this study, the value of a is twelve.

Step 2: PDAVG is calculated by using the following equation.

$$PDAVG_{ij} = \frac{\max(0,\ D_{ij} - AVG_j)}{AVG_j} \text{ for benificial criteria} \tag{16}$$

$$PDAVG_{ij} = \frac{\max(0,\ AVG_j - D_{ij})}{AVG_j} \text{ for cost criteria} \tag{17}$$

where $PDAVG_{ij}$ represents the positive distance for software reliability prediction model (alternative) i from the AVG for jth performance measure.

Step 3: The weighted sum of PDAVG is calculated by using the following equation.

$$WSP_i = \sum_{j=1}^{c} w_j \times PDAVG_{ij} \tag{18}$$

where WSP_i represents the weighted sum of PDAVG of i^{th} alternative (software reliability prediction model), and c represents the number of performance measures. In this study, the value of c is four and $w_{ij} = 0.25$.

Step 4: NDAVG is calculated by using the following equation.

$$NDAVG_{ij} = \frac{\max(0,\ AVG_j - D_{ij})}{AVG_j} \text{ for benificial criteria} \tag{19}$$

$$NDAVG_{ij} = \frac{\max(0,\ D_{ij} - AVG_j)}{AVG_j} \text{ for cost criteria} \tag{20}$$

where $NDAVG_{ij}$ represents the negative distance for software reliability prediction model (alternative) *i* from the AVG for *j*th performance measure.

Step 5: The weighted sum of NDAVG is calculated by using the following equation.

$$WSN_i = \sum_{j=1}^{c} w_j \times NDAVG_{ij} \tag{21}$$

where WSN_i represents the weighted sum of NDAVG of *i*th alternative (software reliability prediction model), and *c* has the same meaning as in Step 3.

Step 6: The weighted sums of PDAVG and NDAVG are normalized using the following equations.

$$NWSP_i = \frac{WSP_i}{\max(WSP_i)} \tag{22}$$

$$NWSN_i = 1 - \frac{WSN_i}{\max(WSN_i)} \tag{23}$$

For the *i*th alternative (software reliability prediction model), $NWSP_i$ and $NWSN_i$ represent the normalized value of the weighted sum of PDAVG and the weighted sum of NDAVG, respectively.

Step 7: The evaluation factor (EF_i) for the *i*th alternative (software reliability prediction model) is calculated by using the following formula.

$$EF_i = \frac{(NWSP_i + NWSN_i)}{2} \tag{24}$$

Step 8: The software reliability prediction models (alternatives) are ranked based on the value of the evaluation factor, where the highest value gets the first rank.

4. Software Reliability Datasets

4.1. Overview of datasets

For the validation of the proposed approach, this study chooses three software failure datasets, namely Musa-1 (DS1), Musa-2 (DS2), and Iyer and Lee (DS3). All three datasets are the TBF datasets, where the time unit is the CPU time (in seconds). The detailed description of datasets is given in Table 1.

Table 1 Description of datasets

Name of dataset	Failure count	Testing time (s)	Type of project	Development phase
DS1	101	1035.2	Military system	System test operations
DS2	163	1742.6	Military system	System test operations
DS3	191	2238.7	Iyer et al. [24]	System test operations

4.2. Hypothesis testing

As discussed in section 4.1., all the datasets are software failure datasets with TBF. These datasets can be treated as univariate time-series datasets for software reliability prediction. For time series forecasting, a stationarity check is an essential step. A dataset is said to be stationary if it has a constant mean and variance over the entire time period.

For ensuring that the datasets are stationary, the following hypothesis is tested:

(1) H_0: Given dataset is non-stationary.

(2) H_1: Given dataset is stationary.

This is done by applying a well-known statistical significance test called augmented Dickey-Fuller (ADF) test on each dataset [28]. Results of the hypothesis test on each dataset are listed in Table 2. From Table 2, it can be observed that for all three datasets, there is no strong evidence to reject H_0 as p-value > 0.05. Log transformation, followed by first difference transformation, is applied to make all datasets stationary. Next, the ADF test is applied on all datasets after transformation. Now, there is strong evidence to reject H_0 and accept H_1 as p-value < 0.05. Therefore, all datasets after transformation are stationary and can be used further for software reliability prediction. However, in the case of fuzzy-based software reliability prediction models, this assumption of stationarity is not necessary [29].

Table 2 Results of hypothesis testing

Dataset/p-value (ADF test)	Original dataset	Dataset after transformation
DS1	0.7626 (> 0.05)	3.887e-08 (< 0.05)
DS2	0.1809 (> 0.05)	6.214e-16 (< 0.05)
DS3	0.6312 (> 0.05)	6.231e-17 (< 0.05)

5. Experimental Study and Results

5.1. Experimental study

An overview of the experimental study is shown in Fig. 3. As can be seen from the figure, the experimental study can be divided into two phases. The detailed explanation is provided as follows.

Phase 1: Twelve modeling techniques are applied on three software reliability datasets to obtain the results of four performance measures. The motivation for selecting the models is to make a fair comparison of the proposed SOM-FTS technique with existing techniques. The results are compared with distinct types of statistical techniques (ARIMA and RegbyDisc), tree-based models (decision tree), various categories of neural networks (MLP and SVR), and ensemble learning techniques (bagging), and are stored in a 12 × 4 matrix for each dataset. The SOM-FTS technique is implemented on MATLAB R2021 version 9.10. MLR, MLP, SVR, AR, bagging, RegbyDisc, and DT-based models are implemented using WEKA version 3.8.3. The open-source package R version 4.0.2 is used to implement the ARIMA-based software reliability model. The open-source package PyFTS is used to implement the FTS forecasting models proposed by Yu [20], Cheng et al. [21], and Efendi et al. [22].

Phase 2: Twelve software reliability prediction models are evaluated using MCDM. The 12 × 4 matrix obtained from Phase 1 is used as the input for MCDM methods for selecting the most suitable software reliability prediction model taking all four performance measures into consideration.

To justify the best software reliability prediction model, the software reliability prediction models are ranked by applying three MCDM methods, i.e., WSM, TOPSIS, and EDAS, as described in section 3.4. At last, the ranking index for software reliability prediction models in the form of a 12 × 1 matrix is obtained as the output of three MCDM methods for three datasets used in this study.

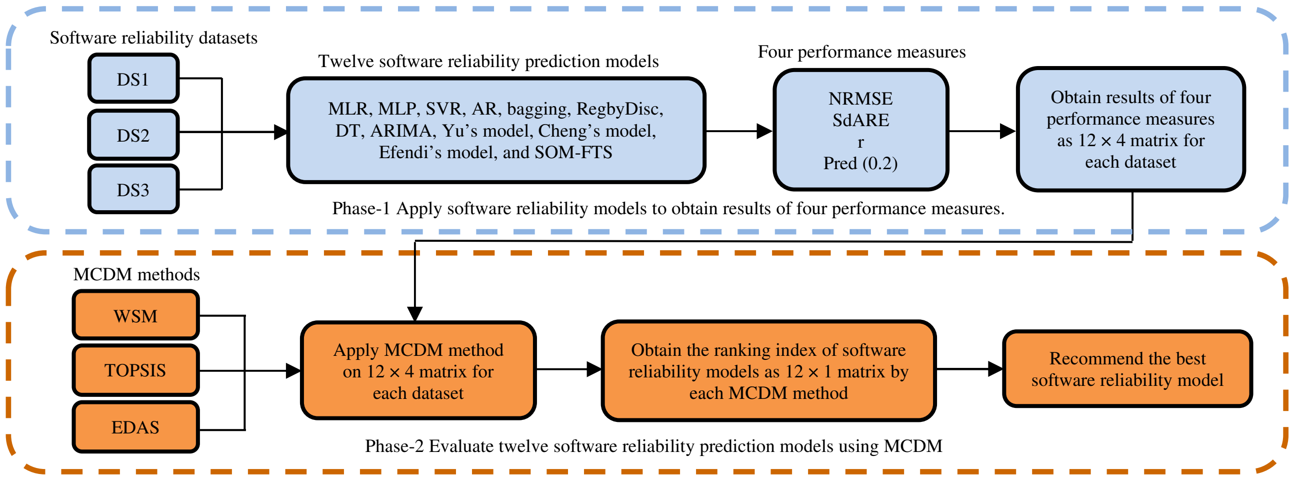
Software reliability datasets
DS1
DS2
DS3
Twelve software reliability prediction models
MLR, MLP, SVR, AR, bagging, RegbyDisc, DT, ARIMA, Yu's model, Cheng's model, Efendi's model, and SOM-FTS
Four performance measures
NRMSE SdARE r Pred (0.2)
Obtain results of four performance measures as 12 × 4 matrix for each dataset
Phase-1 Apply software reliability models to obtain results of four performance measures.
MCDM methods
WSM
TOPSIS
EDAS
Apply MCDM method on 12 × 4 matrix for each dataset
Obtain the ranking index of software reliability models as 12 × 1 matrix by each MCDM method
Recommend the best software reliability model
Phase-2 Evaluate twelve software reliability prediction models using MCDM

Fig. 3 An overview of experimental study

5.2. *Results and discussion*

This section is further divided into two parts. The first part reports the experimental results with respect to various performance measures mentioned in section 3.3. for twelve software reliability prediction models for three datasets. The second part presents the MCDM ranking of all software reliability models.

5.2.1. *Results of software reliability prediction*

Tables 3-5 report the experimentally obtained values of the four performance measures used in this study, namely NRMSE, SDARE, correlation coefficient (r), and Pred (0.2) as described in section 3.3, for twelve modeling techniques for three datasets (DS1, DS2, and DS3). Observations from the experimental results of Tables 3-5 are as follows:

DS1: The proposed modeling technique SOM-FTS is the most accurate technique when considering NRMSE and correlation coefficient (r), whereas RegbyDisc is the most accurate technique when considering SdARE and Pred (0.2).

DS2: The proposed modeling technique SOM-FTS is the most accurate technique when considering NRMSE, SdARE, and correlation coefficient (r), whereas RegbyDisc is the most accurate technique when considering Pred (0.2).

DS3: The proposed modeling technique SOM-FTS performs best in terms of NRMSE and correlation coefficient (r), whereas RegbyDisc performs best in terms of SdARE. In terms of Pred (0.2), Yu's model has the best performance [18].

From the above observations, it is evident that no software reliability prediction modeling technique achieves the best performance across all measures and across all three datasets. This motivates the authors to use MCDM methods for selecting the optimal software reliability prediction modeling technique in the presence of varying performance across different measures.

Table 3 Results of software reliability prediction for DS1

Model/performance measures	NRMSE	SdARE	r	Pred (0.2)
MLR	0.1585	1.0613	0.6062	0.8100
MLP	0.1506	1.0142	0.6576	0.8100
SVR	0.1593	1.1349	0.6158	0.7900
AR	0.1318	0.8604	0.7498	0.8500
Bagging	0.1394	0.9170	0.7198	0.8400
RegbyDisc	0.1106	0.8068	0.8320	0.8900
DT	0.1267	0.8886	0.7721	0.8600
ARIMA	0.1676	0.9949	0.5851	0.7700
Yu [20]	0.1685	1.1872	0.5520	0.7700
Cheng et al. [21]	0.1700	1.1801	0.5356	0.7900
Efendi et al. [22]	0.1622	1.1123	0.5803	0.7900
SOM-FTS (proposed model)	0.1034	0.8241	0.8546	0.8800

Table 4 Results of software reliability prediction for DS2

Model/performance measures	NRMSE	SdARE	r	Pred (0.2)
MLR	0.1371	0.9405	0.6614	0.8642
MLP	0.1451	0.9189	0.6753	0.8395
SVR	0.1378	0.9786	0.6643	0.8704
AR	0.1211	0.8247	0.7500	0.8889
Bagging	0.1163	0.8244	0.7762	0.9074
RegbyDisc	0.1095	0.8614	0.8005	0.9383
DT	0.1320	0.9068	0.6918	0.8642
ARIMA	0.1455	0.9411	0.6413	0.8272
Yu [20]	0.1393	1.0289	0.6589	0.8580
Cheng et al. [21]	0.1378	0.9984	0.6577	0.8519
Efendi et al. [22]	0.1351	0.9560	0.6733	0.8580
SOM-FTS (proposed model)	0.0906	0.7642	0.8685	0.9259

Table 5 Results of software reliability prediction for DS3

Model/performance measures	NRMSE	SdARE	r	Pred (0.2)
MLR	0.1599	1.4429	0.6953	0.8421
MLP	0.1593	1.5554	0.7172	0.8368
SVR	0.1731	1.8204	0.6846	0.8421
AR	0.1434	1.3053	0.7658	0.8737
Bagging	0.1343	1.2252	0.7999	0.8842
RegbyDisc	0.1459	1.3438	0.7549	0.8737
DT	0.1547	1.4004	0.7184	0.8526
ARIMA	0.1670	1.3740	0.6782	0.8368
Yu [20]	0.1482	1.3860	0.7462	0.8895
Cheng et al. [21]	0.1483	1.5008	0.7526	0.8632
Efendi et al. [22]	0.1450	1.3700	0.7583	0.8842
SOM-FTS (proposed model)	0.1230	1.2672	0.8344	0.8632

5.2.2. MCDM ranking

In this section, three MCDM methods, i.e., WSM, TOPSIS, and EDAS, are applied to select the best model from twelve software reliability models, optimizing four conflicting performance measures as selection criteria and considering them together. Table 6 shows the ranking index for software reliability prediction models for all three datasets. From Table 6, it is clear that the proposed SOM-FTS model is ranked as the best among twelve software reliability prediction models for all three datasets by all three MCDM methods.

Table 6 Ranking index of software reliability prediction models using MCDM methods

Model/ranking	Ranking for DS1			Ranking for DS2			Ranking for DS3		
	WSM	TOPSIS	EDAS	WSM	TOPSIS	EDAS	WSM	TOPSIS	EDAS
MLR	7	8	7	7	7	7	9	10	9
MLP	6	6	6	6	8	9	11	11	11
SVR	9	9	9	9	9	8	12	12	12
AR	4	4	4	4	4	4	3	3	3
Bagging	5	5	5	3	3	3	2	2	2
RegbyDisc	2	2	2	2	2	2	5	4	5
DT	3	3	3	5	5	5	8	7	8
ARIMA	8	7	8	10	10	12	10	9	10
Yu [20]	11	11	11	12	12	11	6	6	6
Cheng et al. [21]	12	12	12	11	11	10	7	8	7
Efendi et al. [22]	10	10	10	8	6	6	4	5	4
SOM-FTS (proposed model)	1	1	1	1	1	1	1	1	1

6. Conclusions

This study proposes a hybrid neural-FTS (SOM-FTS) modeling technique to predict software reliability. The SOM-FTS technique is applied on three benchmark software reliability datasets. Also, an MCDM-based approach is applied to compare the SOM-FTS modeling technique with other modeling techniques in terms of various divergent performance measures. Based on the experimental study, the following conclusions are made:

(1) SOM-FTS is ranked as the best software reliability modeling technique across four conflicting performance measures through all the three MCDM methods (WSM, TOPSIS, and EDAS).

(2) This work will be helpful for software quality practitioners in two ways. Firstly, the SOM-FTS technique can be used by the software quality assurance teams as an efficient tool for software reliability prediction. Secondly, the proposed MCDM-based approach can be used for the selection of an appropriate software reliability model among various available software reliability prediction models considering various divergent performance measures.

(3) Although there are no silver bullets to predict software reliability for all conditions, the MCDM approach is an aid to handle this issue and select a prediction model taking into account multiple performance measures. MCDM handles this issue by selecting the most appropriate model among different available models for software reliability prediction by optimizing all performance measures.

(4) MCDM approach has many advantages. One of the most important key strengths of the MCDM method is that it provides a systematic way to select the best alternative against a set of decision criteria. Moreover, the MCDM approach can consider a wide range of criteria taking into account the relative importance of the different criteria. On the other hand, the potential weakness of the MCDM approach is that the subjectivity can be high in deciding the relative importance of different criteria.

In the future, it is suggested applying the SOM-FTS modeling technique on a large number of reliability datasets of industrial and open-source software systems. For the datasets and software systems considered in this study, it is assumed that historical datasets are available. The design and development of techniques to handle sparse software failure datasets can be an interesting extension of this work.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] A. Iannino, et al., “Software Reliability,” Advances in Computers, vol. 30, pp. 85-170, 1990.
[2] H. Pham, System Software Reliability, London: Springer, 2006.
[3] M. Bisi, et al., “Software Development Efforts Prediction Using Artificial Neural Network,” IET Software, vol. 10, no. 3, pp. 63-71, June 2016.
[4] M. Bisi, et al., “Prediction of Software Inter-Failure Times Using Artificial Neural Network and Particle Swarm Optimisation Models,” International Journal of Software Engineering, Technology, and Applications, vol. 1, no. 2-4, pp. 222-244, 2015.
[5] P. Roy, et al., “Forecasting of Software Reliability Using Neighborhood Fuzzy Particle Swarm Optimization Based Novel Neural Network,” IEEE/CAA Journal of Automatica Sinica, vol. 6, no. 6, pp. 1365-1383, November 2019.
[6] P. Rani, et al., “A Neuro-Particle Swarm Optimization Logistic Model Fitting Algorithm for Software Reliability Analysis,” Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, vol. 233, no. 6, pp. 958-971, 2019.
[7] J. Wang, et al., “Software Reliability Prediction Using a Deep Learning Model Based on the RNN Encoder-Decoder,” Reliability Engineering and System Safety, vol. 170, pp. 73-82, February 2018.

[8] G. Jabeen, et al., “An Improved Software Reliability Prediction Model by Using High Precision Error Iterative Analysis Method,” Software Testing, Verification, and Reliability, vol. 29, no. 6-7, Article no. e1710, 2019.
[9] L. Zhen, et al., “Parameter Estimation of Software Reliability Model and Prediction Based on Hybrid Wolf Pack Algorithm and Particle Swarm Optimization,” IEEE Access, vol. 8, pp. 29354-29369, 2020.
[10] S. Kassaymeh, et al., “Salp Swarm Optimizer for Modeling Software Reliability Prediction Problems,” Neural Processing Letters, vol. 53, no. 6, pp. 4451-4487, December 2021.
[11] P. F. Pai, et al., “Software Reliability Forecasting by Support Vector Machines with Simulated Annealing Algorithms,” Journal of Systems and Software, vol. 79, no. 6, pp. 747-755, June 2006.
[12] J. Lou, et al., “Software Reliability Prediction via Relevance Vector Regression,” Neurocomputing, vol. 186, pp. 66-73, April 2016.
[13] H. Zhang, et al., “Applying Software Metrics to RNN for Early Reliability Evaluation,” Journal of Control Science and Engineering, vol. 2020, Article no. 8814394, 2020.
[14] B. Mohammed, et al., “Failure Prediction Using Machine Learning in a Virtualised HPC System and Application,” Cluster Computing, vol. 22, no. 2, pp. 471-485, 2019.
[15] A. Kumar, et al., “A Hybrid SOM-Fuzzy Time Series (SOMFTS) Technique for Future Forecasting of COVID-19 Cases and MCDM Based Evaluation of COVID-19 Forecasting Models,” International Conference on Computing, Communication, and Intelligent Systems, pp. 612-617, February 2021.
[16] A. Kaur, et al., “Statistical Comparison of Modelling Methods for Software Maintainability Prediction,” International Journal of Software Engineering and Knowledge Engineering, vol. 23, no. 6, pp. 743-774, August 2013.
[17] P. Singh, “Rainfall and Financial Forecasting Using Fuzzy Time Series and Neural Networks Based Model,” International Journal of Machine Learning and Cybernetics, vol. 9, no. 3, pp. 491-506, 2018.
[18] Q. Song, et al., “Forecasting Enrollments with Fuzzy Time Series—Part I,” Fuzzy Sets and Systems, vol. 54, no. 1, pp. 1-9, 1993.
[19] S. M. Chen, “Forecasting Enrollments Based on Fuzzy Time Series,” Fuzzy Sets and Systems, vol. 81, no. 3, pp. 311-319, 1996.
[20] H. K. Yu, “Weighted Fuzzy Time Series Models for TAIEX Forecasting,” Physica A: Statistical Mechanics and Its Applications, vol. 349, no. 3-4, pp. 609-624, April 2005.
[21] C. H. Cheng, et al., “Forecasting Innovation Diffusion of Products Using Trend-Weighted Fuzzy Time-Series Model,” Expert Systems with Applications, vol. 36, no. 2, pp. 1826-1832, March 2009.
[22] R. Efendi, et al., “Improved Weight Fuzzy Time Series as Used in the Exchange Rates Forecasting of US Dollar to Ringgit Malaysia,” International Journal of Computational Intelligence and Applications, vol. 12, no. 1, Article no. 1350005, March 2013.
[23] W. C. S. Hong, Principal Concepts in Applied Evolutionary Computation: Emerging Trends, Hershey: IGI Global, 2012.
[24] R. K. Iyer, et al., “Measurement-Based Analysis of Software Reliability,” https://www.cse.cuhk.edu.hk/~lyu/book/reliability/pdf/Chap_8.pdf, 1996.
[25] P. C. Fishburn, “Additive Utilities with Incomplete Product Sets: Application to Priorities and Assignments,” Operations Research, vol. 15, no. 3, pp. 537-542, 1967.
[26] C. L. Hwang, et al., Multiple Attribute Decision Making, Berlin: Springer Berlin Heidelberg, 1981.
[27] M. K. Ghorabaee, et al., “Multi-Criteria Inventory Classification Using a New Method of Evaluation Based on Distance from Average Solution (EDAS),” Informatica, vol. 26, no. 3, pp. 435-451, Januray 2015.
[28] D. A. Dickey, et al., “Distribution of the Estimators for Autoregressive Time Series with a Unit Root,” Journal of the American Statistical Association, vol. 74, no. 366, pp. 427-431, June 1979.
[29] C. Kocak, et al., “A New Fuzzy Time Series Method Based on an ARMA-Type Recurrent Pi-Sigma Artificial Neural Network,” Soft Computing, vol. 24, no. 11, pp. 8243-8252, June 2020.