Int J Syst Assur Eng Manag (November 2017) 8(Suppl. 2):S1487–S1502
DOI 10.1007/s13198-017-0618-4

ORIGINAL ARTICLE

Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept

Lov Kumar[1] · **Santanu Ku Rath**[1]

Received: 26 September 2016 / Revised: 30 March 2017 / Published online: 21 April 2017

Abstract In present day scenario, majority of software companies use object-oriented concept to develop software systems as it enables effective design, development, testing and maintenance, in addition to the optimal characterization of the software system. With the increase in number of these software systems, their effective maintenance aspect becomes very important day by day. In this study, Neuro-Fuzzy approach: hybrid neural network and fuzzy logic approach has been considered to develop a maintainability model using ten different object-oriented static source code metrics as input. This method is applied on maintainability data of two commercial software products such as UIMS and QUES. Rough set analysis (RSA) and principal component analysis (PCA) are used to select suitable set of metrics from the ten metrics employed to improve performance of maintainability prediction model. From experimental results, it is observed that Neuro-Fuzzy model can effectively predict the maintainability of object-oriented software systems. After implementing parallel computing concept, it is observed that the training time gets reduced to a significant amount when the number of computing nodes were increased. Further it is observed that selected subset of metrics using feature selection techniques i.e., PCA, and RSA was able to predict maintainability with higher accuracy.

✉ Lov Kumar
lovkumar505@gmail.com

Santanu Ku Rath
skrath@nitrkl.ac.in

[1] Department of Computer Science and Engineering, National Institute of Technology, Rourkela, India

Keywords Artificial neural network · Fuzzy Logic · Source code metrics · Maintainability · Parallel computing

1 Introduction

It is often observed that, software developers give more emphasis on object-oriented development methodologies in the present day scenario, because of its inherent advantages of traditional development approach such as cohesion, coupling, inheritance etc. To assess the quality of the software, developed on the basis of object-oriented methodology, different software metrics are used. The usefulness of these metrics lies in their ability to predict the quality of the developed software. Software quality attributes that have been identified by ISO/IEC 9126 (Jung et al. 2004) are efficiency, functionality, portability, maintainability, reliability and usability. In recent years maintainability is considered as important quality parameter for achieving considerable success in software system (Misra 2005; Zhou and Baowen 2008; Malhotra and Chug 2014). Software maintainability means the capability of the software system or component to fix or correct the faults, improve performance, or adapt to changes in environment. A good number of researchers have concluded that the maintainability prediction model can be developed using source code metrics (Misra 2005; Chen and Huang 2009; Chidamber and Kemerer 1994; Li and Henry 1993; Basili et al. 1996; Damaševičius and Štuikys 2010; Misra 2007; Baski and Misra 2011; Misra and Akman 2008; Misra et al. 2011), which are used to measure the internal structure of software system i.e., complexity, coupling, cohesion, inheritance, and size. There are several source code metrics proposed by different researchers such as, Abreu MOOD

metrics suite (Abreu and Carapuca 1994; Kang and Bieman 1995; Briand et al. 2000; Halstead 1977; Henderson-Sellers 1996; Li and Henry 1993; McCabe 1976; Lorenz and Kidd 1994), and CK metrics (Chidamber and Kemerer 1994) suite, to measure the internal structure of software systems. In this work, only those source code metrics are selected which have strong relationships with maintainability of software. So, we have considered CK metrics suite consisting different source code metrics such as depth of inheritance (DIT), weighted methods per class (WMC), number of children (NOC), response for a class (RFC), lack of cohesion in methods (LCOM), and Li and Henry metrics consisting different source code metrics such as data abstraction coupling (DAC), message passing coupling (MPC), and number of local methods (NOM) and size metrics [traditional line of code (SIZE1), and total number of attributes and methods of a class (SIZE2)] to develop a maintainability prediction model for predicting maintainability of object-oriented software systems. These source code metrics mostly emphasize on quality aspect of a class.

Performance of the maintainability prediction model depends on choosing the right set of object-oriented source code metrics. Feature selection is a process of selecting a suitable subset of object-oriented metrics from the available ones. In our work, RSA and PCA are considered in finding the right subset of source code metrics (Kumar and Rath 2015). In RSA, subset of features are identified to collectively improve predictive capability while in PCA, initial set of features data are considered and builds the derived values.

There are a number of mechanisms employed in literature for software maintainability prediction. Some of the extensively employed approaches for maintainability prediction are regression based model, association rule mining, clustering, neural network, Bayesian network, SVM etc. (Malhotra and Chug 2014; Zhou and Leung 2007; Kumar and Rath 2015; Kumar et al. 2015). Development of accurate maintainability prediction model to predict maintainability of class is still a challenging task in software engineering discipline. In this study, hybrid approach of neural network and Fuzzy logic (Neuro-Fuzzy approach) has been considered while developing a model to predict maintainability of object-oriented software. Parallel computing concept is used to accelerate training procedure of the neural network model (Kumar and Rath 2015). Parallel computing algorithms are classified into two subclasses consisting of node and training dataset parallelism. In node parallelism, neurons are mapped into different computing node for pipelining the process. In training dataset parallelism, a complete structure of neural network is assigned to each computing node. Each node conducts entire computation for neural network. In this study training dataset parallelism has been considered to accelerate the neural network training procedure.

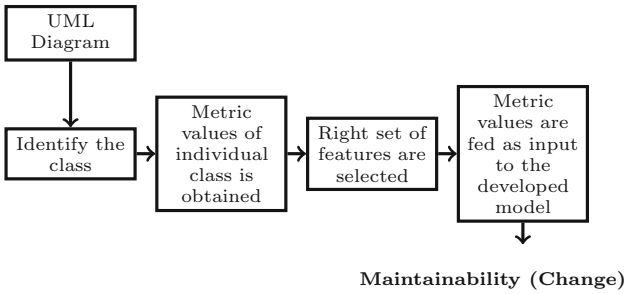

Fig. 1 Flow chart for maintainability prediction

The generic steps followed to predict maintainability of any object-oriented software are shown in Fig. 1. Initially the classes of the respective software are identified from the class diagram and next, the different metric values of a class are extracted using different tools such as CKJM, LOCMetric analyzer etc. as available in literature.[1] Then, different feature selection techniques are considered to find the right sets of features. Further, these metric sets are considered as input to the developed model in order to predict maintainability of all individual classes in a software.

This study intends to focus on:

- Identification of suitable set of source code metrics for maintainability prediction.
- Development of a maintainability prediction model using Neuro-Fuzzy approach.
- Accelerating the Neuro-Fuzzy approach training procedure.

The rest of the paper is organized in this way: Sect. 2 shows the existing literature in the field of maintainability prediction. Section 3 highlights the experimental dataset used to develop the maintainability prediction model. The feature selection techniques and research methodology are described in Sects. 4 and 5, respectively. Section 6 presents the performance parameters used for evaluating the models. Section 7 presents the research framework and also highlights on the results for maintainability prediction, achieved by applying Neuro-Fuzzy approach. In Sect. 8, various threats to validity are discussed and Sect. 9 concludes the paper with scope for future work.

2 Related work

This section presents a review of literature on the use of software metrics and their application in maintainability prediction. In literature, different combination and subsets

[1] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/.

of software metrics have been analyzed and relationship is derived between the object-oriented metrics and maintainability as mentioned in Table 1. In Table 1, the first column indicates the name of the author, and the year in which the work was carried out. The second column indicates the different subsets of object-oriented metrics, considered to develop a model for predicting maintainability of object-oriented software. Last column of the table represents the techniques used to develop a maintainability prediction models.

From Table 1, it is observed that in all the studies made by different authors, independent variables are the different subset of the object-oriented metrics and the dependent variable is the maintainability of object-oriented software. This shows that the performance of maintainability prediction model depends on the software metrics which have been considered as input to develop a model. Selection of right set of feature is an important data preprocessing task in different application of data mining and machine learning (Huang and Chow 2005; Kabir et al. 2010). In this work, two different types of features selection techniques i.e. principal component analysis (PCA), and rough set analysis (RSA) have been considered to find right subset of software metrics (Pawlak 1982). The effectiveness of these feature selection techniques are evaluated using Neuro-Fuzzy approach (Adeli and Hung 1994).

From Table 1, our observations suggest that regression based analysis and their different forms are commonly used by various authors; but very less work has been carried out on using neural network models for maintainability prediction. Neural network models over the years have seen an explosion of interest, and their applicability across a wide range of problem domains. In this paper, Neuro-Fuzzy approach is used to develop the maintainability prediction model (Adeli and Hung 1994).

3 Experimental dataset and setup

The following subsections highlight on model used for prediction of quality parameter i.e., maintainability, considering different case studies. Data are normalized to obtain better accuracy and then dependent, along with independent variables are chosen for maintainability estimation.

3.1 Experimental dataset

In this paper, two commercial object-oriented software published by Henry are used as case studies (Li and Henry 1993). Software such as quality evaluation system (QUES), and user interface system (UIMS) are chosen for maintainability prediction. These case studies were chosen mainly because many researchers recently used these case studies to evaluate the performance of their developed maintainability prediction models (Kumar and Rath 2015; Zhou and Leung 2007; Koten and Gray 2006; Elish and Elish 2009; Al-Jamimi et al. 2012; Aljamaan et al. 2013; Kumar and Rath 2015) and hence this study intends to be able to compare performance of proposed model with the performance of published models. QUES, and UIMS software have 71 and 39 number of classes. These softwares are developed using Classic-ADa programing language. Classic-ADa is an object-oriented programing language that adds the capability of object-oriented programing to ADa by providing object-oriented construct in addition to the ADa constructs (Li and Henry 1993).

3.2 Dependent variable: maintainability

Number of definition and metrics for software maintainability are defined by different authors (Li and Henry 1993; Zhou and Leung 2007; Banker et al. 1993). From literature, it is observed that most of the authors have used ‘maintainability index (MI)’ and ‘change metrics’ as the factors to determine maintainability of software. In this work, maintainability is defined as the amount of change (usually a number) brought in the code throughout the maintenance period. A line change can be considered as an ‘addition’ or ‘deletion’ of lines of code in a class during maintenance period (3 year) (Malhotra and Chug 2014; Li and Henry 1993; Zhou and Leung 2007; Aggarwal et al. 2005; Riaz et al. 2009, 1997). Figure 3 shows the data boxplots of maintainability for QUES and UIMS datasets. The line in the middle of box represents the median of the data, lower and upper quartile show the value of 25 and 75% of the data, and lower and upper extreme show the minimum and maximum value of the data. From Fig. 3, it is observed that the median value (middle line of the box plot) of CHANGE in QUES software is higher than those in the UIMS. This suggests that, UIMS software is more maintainable.

3.3 Predictor variables: source code metrics

Number of software metrics have been proposed for different application such as fault prediction, maintenance effort, cost estimation etc. In this paper, ten different object-oriented static source code metrics such as depth of inheritance (DIT), weighted methods per class (WMC), response for a class (RFC), data abstraction coupling (DAC), lack of cohesion in methods (LCOM), number of children (NOC), message passing coupling (MPC), number of local methods (NOM), traditional line of code (SIZE1), and total number of attributes and methods of a class (SIZE2). The detail description of

Table 1 Summary of empirical literature on maintainability

Author	Software metrics used	Prediction technique considered
Li and Henry (1993)	Li and Henry metrics, CK metrics, and size metrics	Regression based models
Oman and Hagemeister (1994)	Mccabe metrics, and Halstead's metrics	Regression based models
Schneberger (1997)	Model complexity metrics	Regression based models
Binkley and Schach (1998)	CDM and design quality metrics	Regression analysis
Kohavi (1999)	Mccabe metrics, Halstead's metrics, LOC	Experts judgments
Bandi et al. (2003)	Interface and design quality metrics	Variance, correlation, and regression analysis
Dagpinar and Jahnke (2003)	Cohesion metrics, coupling metrics, inheritance metrics, and size metrics	Best subset of regression model, correlation, multivariate analysis
Misra (2005)	Britee Abreu, CK metrics, Carapuca metrics, Lorenz and Kidd metrics	Linear regression, correlation and multiple regression
Aggarwal et al. (2005)	Li and Henry and CK metrics	Fuzzy model
Aggarwal et al. (2006)	Li and Henry and CK metrics	Artificial neural network
Koten and Gray (2006)	Li and Henry, CK metrics and size metrics	Bayesian network, backward elimination and stepwise selection and regression tree
Zhou and Leung (2007)	Li and Henry, CK metrics and size metrics	Artificial neural network, multivariate linear regression, multivariate adaptive regression splines, regression tree and support vector regression
Zhou and Baowen (2008)	Design quality metrics	Regression based models
Elish and Elish (2009)	Li and Henry, CK metrics and size metrics	TreeNet
Jin and Liu (2010)	Li and Henry, CK metrics and size metrics	Support vector machine (SVM)
Al-Jamimi et al. (2012)	Li and Henry, CK metrics and size metrics	Fuzzy logic
Chandra (2012)	Li and Henry, CK metrics and size metrics	Support vector machine
Aljamaan et al. (2013)	Li and Henry, CK metrics and size metrics	Multilayer perceptron (MLP), radial basis function network (RBF), support vector machines (SVMs), and M5 model tree (M5P)
Malhotra and Chug (2014)	Li and Henry, CK metrics and size metrics	Feed forward 3-layer back propagation network (FF3LBPN) and general regression neural network (GRNN)
Kumar and Rath (2014)	Li and Henry, CK metrics and size metrics	ANN, Neuro-GA, and Neuro-PSO
Kumar and Rath (2015)	CK metrics	ANN, and Neuro-GA
Kumar et al. (2015)	CK metrics, Li and Henry metrics, and size metrics	Neuro-GA
Kumar and Rath (2015)	Li and Henry, CK metrics and size metrics	Neuro-GA with parallel computing

these metrics are described in Li and Henry (1993). Figure 2 shows the data boxplots of source code metrics for QUES and UIMS datasets. Table 2 displays the descriptive statistics such as: Min, 25%, Max, Mean, Median, 75% and Standard deviation for all the metrics across both projects.

From Fig. 2 and Table 2, observations made are:

1. In both the system, DIT metric of classes have low value of median. This low value shows that, both systems have used limited inheritance.
2. The value of 'NOC' in QUES software product, has all classes with NOC values to be zero. This indicates that there are no immediate sub-classes and hence NOC is not considered in computing maintenance effort.

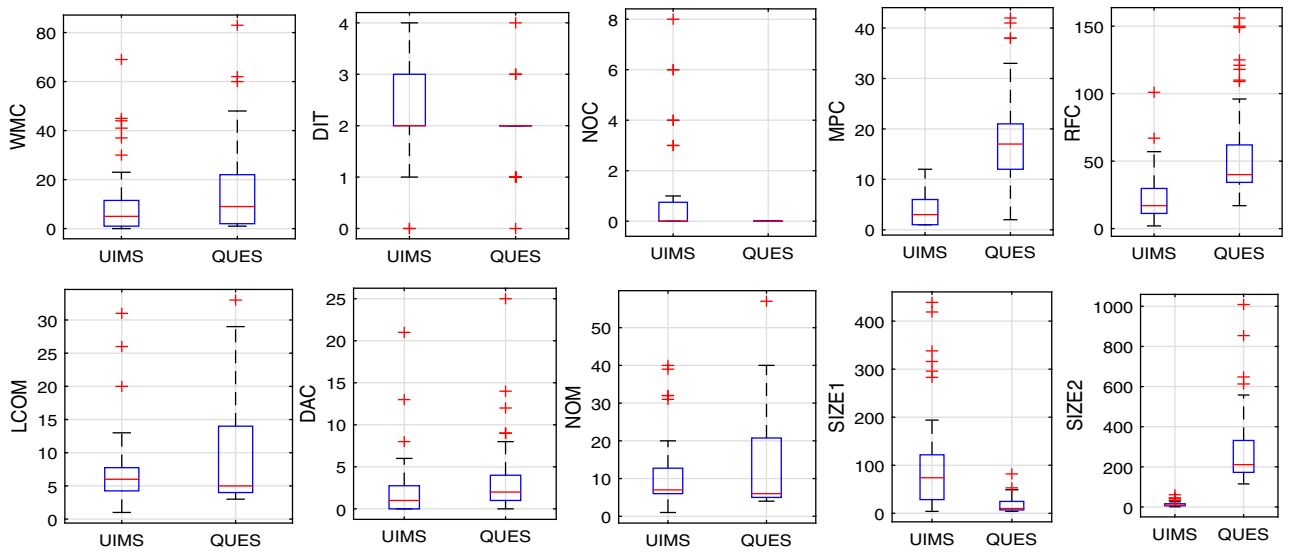

Fig. 2 Source code metrics

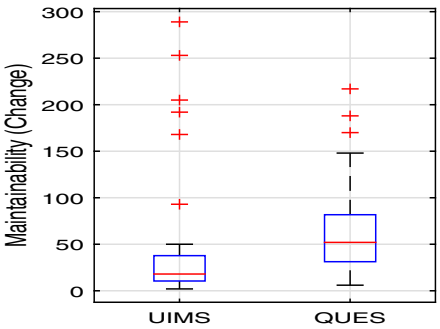

Fig. 3 Descriptive statics of maintainability

3. The median value of coupling metrics (RFC, and MPC) in QUES are higher than those in the UIMS. This suggests that QUES software have high coupling between the classes.
4. The median value of cohesion metrics i.e., LOCM are similar for both system. This suggests that, both systems have similar cohesion.
5. Similar median value of NOM and SIZE2 are found in the both software, suggesting that both systems are similar class size at the design level. However, both software have significant difference in SIZE1.

3.4 Cross correlation analysis

In this study, Pearson's correlations (*r* Coefficient of correlation) is used to measure the linear relation between different source code metrics. Pearson's correlations is used to measure the direction and strength of the linear relationship between two attributes. Figure 4 shows the Pearson's correlation among all source code metrics for UIMS and QUES software system. The sign of the correlation coefficient defines the direction of the relationship, i.e., −ve or +ve. A +ve value of *r* indicates that the independent and dependent variables grow linearly (when independent variables increases, the dependent metrics also increases, and vice versa). While in case of −ve value of r, the independent variable is inversely proportional to dependent variable (when independent variables increases, the dependent metrics decreases, and vice versa) For the sake of simplicity, the graphs are represented using four different symbols as described below:

- Black circle (●): r value between 0.7 and 1.0 indicate a strong positive linear relationship.
- White circle (○): r value between 0.3 and 0.7 indicate a weak positive linear relationship.
- Black square (■): r value between −1 and −0.7 indicate a strong negative linear relationship.
- White square (□): r value between −0.7 and −0.3 indicate a weak negative linear relationship.
- Blank circle: no linear relationship.

From Fig. 4, it is observed that there is a strong positive linear relationship between some metrics such as WMC is highly correlated with MPC, RFC, LCOM, NOM, SIZE1, and Change. This high correlation value between pair of metrics show that, although metrics measure different features of class design, there is a significant statistical reason to believe that classes with low (or high) metric values also have low (or high) values of other highly correlated metrics. In this paper, Principal component analysis (PCA) has been considered to extract new set of metrics from original metrics set that have low correlation value.

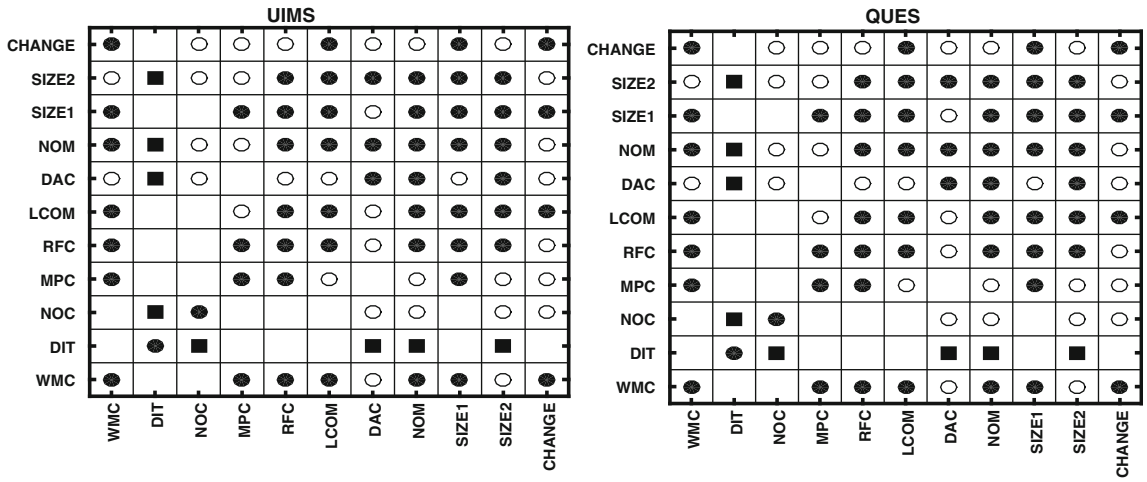

Fig. 4 Correlation between source code metrics

Table 2 Descriptive statistics of classes for UIMS and QUES

	WMC	DIT	NOC	MPC	RFC	LCOM	DAC	NOM	SIZE1	SIZE2
UIMS										
Min	1.00	0.00	0.00	2.00	17.00	3.00	0.00	4.00	4.00	115.00
25%	2.00	2.00	0.00	12.00	34.25	4.00	1.00	5.00	7.00	172.50
Mean	14.96	1.92	0.00	17.75	54.38	9.18	3.44	13.41	18.03	275.58
Median	9.00	2.00	0.00	17.00	40.00	5.00	2.00	6.00	10.00	211.00
75%	22.00	2.00	0.00	21.00	62.00	14.00	4.00	20.75	24.75	331.50
Max	83.00	4.00	0.00	42.00	156.00	33.00	25.00	57.00	82.00	1009.00
SD	17.06	0.53	0.00	8.33	32.67	7.31	3.91	12.00	15.21	171.60
QUES										
Min	0.00	0.00	0.00	1.00	2.00	1.00	0.00	1.00	4.00	1.00
25%	1.00	2.00	0.00	1.00	11.25	4.25	0.00	6.00	28.50	6.00
Mean	11.38	2.15	0.95	4.33	23.21	7.49	2.41	11.38	106.44	14.00
Median	5.00	2.00	0.00	3.00	17.00	6.00	1.00	7.00	74.00	9.00
75%	11.50	3.00	0.75	6.00	29.75	7.75	2.75	12.75	121.75	16.00
Max	69.00	4.00	8.00	12.00	101.00	31.00	21.00	40.00	439.00	61.00
SD	15.90	0.90	2.01	3.41	20.19	6.11	4.00	10.21	114.65	13.47

3.5 Effectiveness of metrics

In this study, three different set of metrics (one containing all metrics, selected metrics using RSA, extracted feature using PCA) are considered as input to develop maintainability prediction model. The dependent and independent variables of the model are shown in Table 3.

4 Feature extraction and selection using PCA and RSA

Since the performance of the maintainability prediction model is highly influenced by the quality of the maintainability dataset, consisting of software metrics and the maintainability information, the selection of the right set of software metrics becomes an important step of the maintainability prediction process. In this study, two different types of features selection techniques have been considered to select right subset of object-oriented metrics out of total available object-oriented metrics which are able to predict maintainability of object-oriented software with higher accuracy.

4.1 Principal component analysis (PCA)

Principal component analysis (PCA) concept was first develop by Karl Pearson in 1901 (Kumar and Rath 2015). It is a statistical technique used for transfer to feature space of lower dimension having the most significant features

Table 3 Effectiveness of metrics

Analysis	Dependent variable	Independent variable
A1	Maintainability	DIT, WMC, RFC, NOC, DAC, LCOM, MPC, NOM, SIZE2, SIZE1
A2	Maintainability	Reduced feature attributes using RSA
A3	Maintainability	Extracted feature attributes using PCA

Fig. 5 Framework of PCA calculation

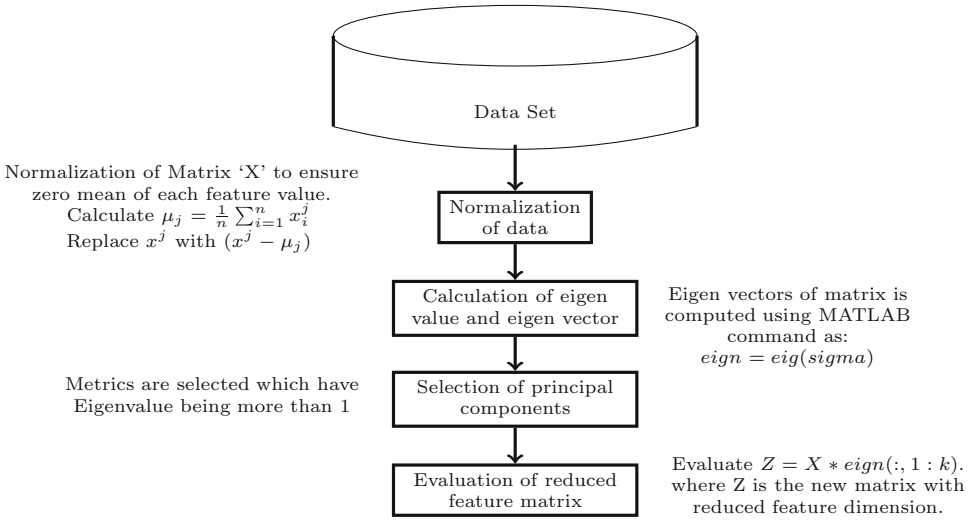

Table 4 Rotated principal component

Metrics	UIMS		QUES		
	PC1	PC2	PC1	PC3	PC3
DIT	−0.368	**0.745**	0.062	0.027	**0.976**
WMC	**0.919**	0.189	**0.830**	0.261	−0.272
RFC	**0.957**	0.182	**0.874**	0.339	0.048
NOC	0.349	**−0.792**	–	–	–
MPC	**0.693**	0.389	−0.026	**0.966**	0.037
LCOM	**0.819**	0.220	**0.870**	−0.153	0.058
NOM	**0.936**	0.020	**0.972**	−0.129	0.094
DAC	**0.707**	−0.311	**0.797**	0.028	0.425
SIZE1	**0.933**	0.234	**0.973**	−0.090	0.187
SIZE2	**0.788**	−0.357	**0.808**	0.481	−0.090
Eigenvalues	5.332	2.351	5.375	1.397	1.246
Cumulative % variance	53.315	76.887	59.725	75.244	89.086
% variance	53.315	23.57	59.725	15.519	13.842

from data space of high dimension. The detail steps of PCA is described in Fig. 5.

From Fig. 4, it is clear that many pairs of source code metrics have high (>0.7) correlation value. PCA is used to extract metrics from raw metrics that have low correlation value, we call the new PCA. In PCA, only those metrics are selected which have Eigenvalue being more than 1. Table 4 shows the rotated component metrics of raw data. In this paper, the rotation is performed to maximizes the sum of the variances of the squared coefficients within each eigenvector. The value greater then 0.7 (shown bold in Table 4) are the metrics, which are used to interpret the principal component. Table 4 also shows the eigenvalue, variance percentage and cumulative percentage.

The interpretations of principal component for QUES software (similar for UIMS software) are given as follows:

1. PC1: WMC, RFC, LOCM, NOM, DAC, SIZE1, and SIZE2 are size, coupling, and cohesion metrics. PC1 contains the size, cohesion, and coupling metrics.
2. PC2: PC2 contains coupling metrics i.e., MPC that counts the number of send statements in class.

3. PC3: PC3 contains inheritance metrics i.e., DIT that measure the depth of inheritance tree of a class.

4.2 Rough set analysis

Pawlak described rough set analysis as a formal approximation of a conventional (CRISP) set (Pawlak 1982). Lowe and upper bound of the raw data are used to represent this formal approximation. The application of this formal approximation is to analysis of various data types, especially when dealing with inexact, vague and uncertain data. Figure 6 shows the steps followed to obtain reduced attribute set.

In this study, RSA is used as feature selection technique to select right set of metrics for improving performance of maintainability prediction model. In RSA, data needs to be classified before applying RSA. In this work, K-means clustering was used to classify the data. The approach followed in K-means clustering is shown in Fig. 7. The distance from each object to the centroid is computed using the euclidean distance concept.

Equation 4 shows the function for computing the euclidean distance.

$$d(x,y) = \sum_{i=1}^{p} |x_i - y_i| \tag{4}$$

The cluster center's of each source code metrics for our case studies are shown in Table 5.

After computing the value of cluster center, each source code metrics are categorized into three different groups. Table 6 shows the range of source code metrics in each group. The selected set of source code metrics are tabulated in Table 7. The reduced set of source code metrics obtained using rough set analysis are considered as input to develop maintainability prediction model. The selected set of source code metrics are tabulated in Table 7.

5 Research methodology: Neuro-Fuzzy approach

Neuro-Fuzzy approach helps to build hybrid architecture of neural network and fuzzy logic which maps fuzzy inputs to a crisp output (Adeli and Hung 1994). In Neuro-Fuzzy approach, Fuzzy back propagation (Fuzzy BP) architecture is used to train the network as shown in Fig. 8.

In Fig. 8, the input vector I is represented as: $I = (I_0, I_1, I_2, I_3, \ldots, I_n)$ and weight W vector is represented as $W = (W_0, W_1, W_2, W_3, \ldots, W_n)$. In Neuro-Fuzzy approach, output O is computed as:

$$O = f(NET) = f\left(CE\left(\sum_{i=0}^{n} W_i I_i\right)\right) \tag{5}$$

where n is the no of inputs, I and W represent the input and weight vector respectively. NET is computed as:

$$NET = CE(net) \tag{6}$$

where *net* is defined as:

$$net = \sum_{i=0}^{n} W_i I_i \tag{7}$$

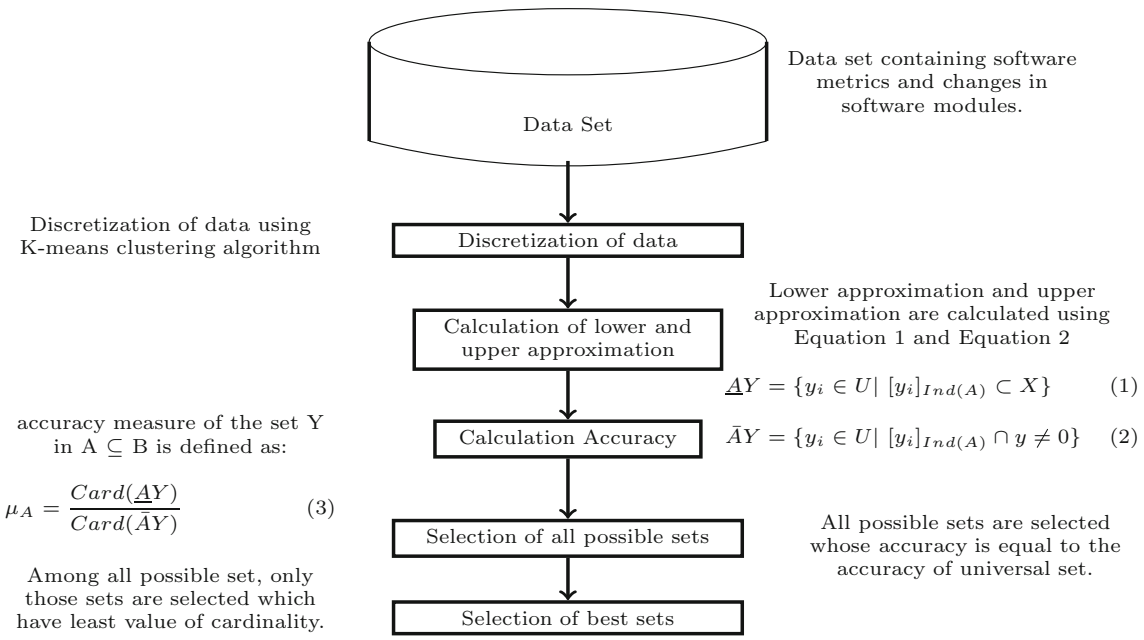

Fig. 6 Framework of rough set theory

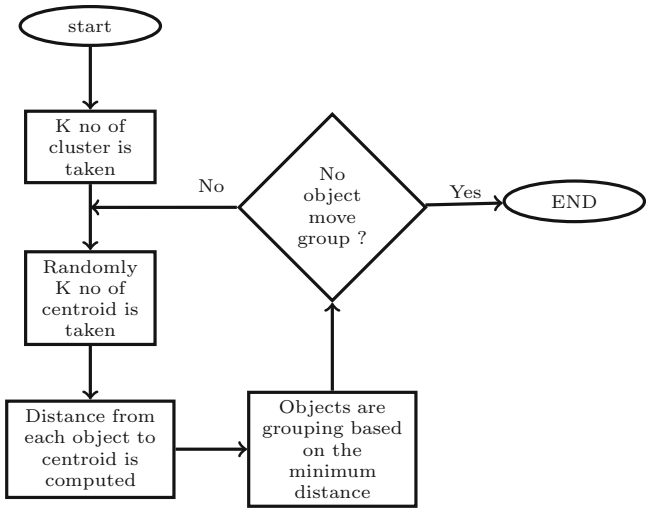

Fig. 7 Flow chart K-means clustering algorithm

Table 5 Cluster Center of source code metrics

	UIMS			QUES		
	C1	C2	C3	C1	C2	C3
DIT	0.75	1	3.06	0.90	2	3.25
WMC	1.60	11.23	44.33	6.01	32.35	68.33
RFC	11.80	33.10	69.75	35.47	70.05	129.75
NOC	0	1.80	5.60	–	–	–
MPC	1.13	4.56	9.87	8.90	18.68	34.11
LCOM	3.75	7.75	25.66	4.40	14.16	24.87
NOM	5.047	12.61	34.80	5.55	21.29	37.77
DAC	0.45	3.33	17.00	1.79	7.62	25.00
SIZE1	30.88	100.266	348.50	176.32	309.29	611.45
SIZE2	6.44	18.55	43.40	8.73	28.80	52.14
CHANGE	21.14	188.33	271	33.95	84.15	173.80

Table 6 Group range

Group	Range
High	$[\frac{C2+C3}{2}$, MAX(metrics)]
Low	[MIN(metrics), $\frac{C1+C2}{2}$]
Medium	$[\frac{C1+C2}{2}, \frac{C2+C3}{2}]$

Table 7 Reduciced attribute

Project	Metrics
UIMS	Size2, DAC, MPC
QUES	Size1, DAC, LCOM

The *CE* function is the centroid of the triangular fuzzy number. It maps fuzzy weighted summation to a crisp value which is called defuzzification operation. If $net = (net_m, net_\alpha, net_\beta)$, then $CE(net)$ is defined as:

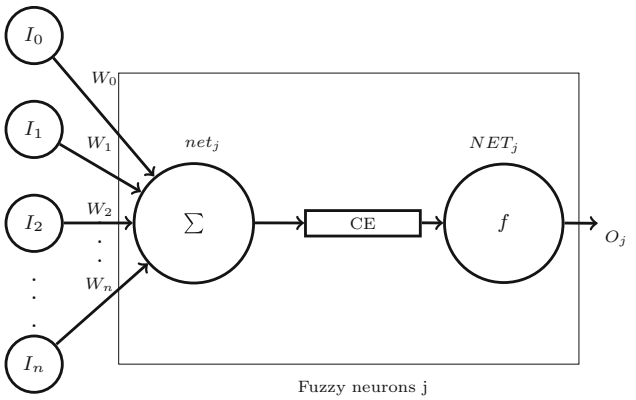

Fig. 8 Neuro-Fuzzy architecture

$$CE(net) = net_m + \frac{1}{3} \times (net_\alpha - net_\beta) = NET \tag{8}$$

In this paper, sigmoidal function is used as output function, which maps non-linearly input to output. Accordingly Eq. 5 can be represented as:

$$Output = fun(NET) = \frac{1}{1 + e^{(-NET)}} \tag{9}$$

In fuzzy neurons, both input and weight vector are represented by triangular left-right fuzzy numbers (LR-type). LR-type fuzzy numbers are type of representation for fuzzy numbers, proposed by Dubois and Prade in 1979's (Dubois and Prade 1979). A fuzzy number M is of LR-type if there exist reference functions L (for left), R (for right) and scalars, $\alpha > 0, \beta > 0$ with

$$\mu_m = \begin{cases} L\left(\frac{m-x}{\alpha}\right) & for \quad x \leq m \\ R\left(\frac{x-m}{\beta}\right) & for \quad x > m \end{cases} \tag{10}$$

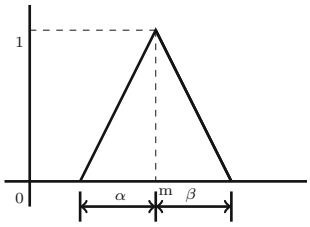

Fig. 9 Symmetric triangular LR-type fuzzy number

where m is the mean value of fuzzy number 'M'. α and β are refereed to as left and right spreads as shown in Fig. 9.

Considering μ_M as the fuzzy number 'M' membership function. An LR-type of Fuzzy number 'M' can be defined as (m, α, β). Thus for input vector $I = (I_0, I_1, I_2, I_3, \ldots, I_n)$, single input I_i is represented as $I_i = (I_{mi}, I_{\alpha i}, I_{\beta i})$. Similarly for weight vector $W = (W_0, W_1, W_2, W_3, \ldots, W_n)$, single weight W_i is represented as $W_i = (W_{mi}, W_{\alpha i}, W_{\beta i})$.

Fuzzy back propagation architecture Fuzzy BP is a three layered feed-forward architecture i.e., input, hidden, and output layer. The execution of Fuzzy BP model is a two stage process such as:

1. *Learning or training* The learning procedure of fuzzy BP follows the gradient descent method of minimizing error. Here, mean square error (MSE) for patterns p is represented as follow:

$$E_p = \sum_{p=1}^{n} \frac{1}{2} \left(O_p'' - O_p \right)^2 \tag{11}$$

where O and O$''$ are the actual and expected output respectively.
During learning phase, the weights are updated using following equation:

$$W_{k+1} = W_k + \Delta W_k \tag{12}$$

where ΔW_k is computed using the following equation:

$$\Delta W_k = -\eta \bigtriangledown E_{ik} + \alpha W_k \tag{13}$$

where $\bigtriangledown E_{ik}$ is given by

$$\bigtriangledown E_{ik} = \frac{\partial E_{ik}}{\partial W} = \left(\frac{\partial E_{ik}}{\partial W_m}, \frac{\partial E_{ik}}{\partial W_\alpha}, \frac{\partial E_{ik}}{\partial W_\beta} \right) \tag{14}$$

and the weight vector W is represented as: $W = (W_m, W_\alpha, W_\beta)$.

2. *Inference* After fuzzy BP execution, the model is trained using training data set and then it is ready for inference. Considering a set of patterns $\bar{F_p}$ to be inferred, where $\bar{F_p} = (\bar{F_{p1}}, \bar{F_{p2}}, \bar{F_{p3}}, \ldots, \bar{F_{pn}})$ and $\bar{F_{pi}}$ is an LR-type fuzzy number given by $\bar{F_{pi}} = (\bar{F_{pmi}}, \bar{F_{p\alpha i}}, \bar{F_{p\beta i}})$. The objective is to obtain output O_p for the corresponding patterns F_p. Algorithm 1 is used in inference of fuzzy BP.

Algorithm 1 Algorithm for inference of fuzzy BP: Inference()

Input: $\bar{F} = (F_1, F_2, F_3, \ldots, F_n)$, W_{ih}, and W_{ho}
Output: $\bar{O} = (O_1, O_2, O_3, \ldots, O_n)$
where $\bar{F}$, $\bar{O}$ represent the input and output pattern pairs of the l-m-n configuration of neural network. W_{ih}, W_{ho} are the weight sets obtained after training using fuzzy BP.
Step 1: P Initialize as 1
Step 2: Patterns F_p are fetched.
Step 3: Output for input neurons is computed using following equation:

$$\bar{O_{pi}} = \bar{F_{pi}} \tag{15}$$

Step 4: Output for hidden neurons is computed using following equation:

$$O'_{pj} = f(NET_{pj}) = \frac{1}{1 + e^{-(NET_{pj})}} \tag{16}$$

where j is the no of hidden neurons and NET_{pj} is computed using Equation 8.
Step 5: Output for the output neurons is computed using following equation:

$$O''_{pk} = f(NET_{pk}) = \frac{1}{1 + e^{-(NET_{pk})}} \tag{17}$$

where k is the no of output neurons and NET_{pk} is computed using Equation 8.
Step 6: P incremented by 1.
If p<n, loop back to Step 2.

6 Performance evaluation parameters

Software maintainability estimation accuracy for a model designed by using AI techniques is determined by using four different performance parameters such as Mean Absolute Error (MAE), Mean magnitude Relative Error (MMRE), and Standard Error of the Mean (SEM) (Menzies et al. 2006). They are represented as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} \left(|X_i' - X_i| \right) \tag{18}$$

$$MMRE = \frac{1}{n} \sum_{i=1}^{n} \frac{|X_i - X_i'|}{X_i + 0.05} \tag{19}$$

In Eq. 19, a numerical value of 0.05 is added in the denominator in order to avoid numerical overflow (division by zero). In Eqs. 18, 19, X_i' and X_i show the estimated and actual value respectively.

$$SEM = \frac{SD}{\sqrt{n}} \tag{20}$$

where SD is the sample standard deviation, and n is the number of samples.

7 Research framework and experimental results

Figure 10 shows our research framework consisting of various steps. The first step in the process is to compute the source code metrics and maintainability for UIMS and QUES software system. The source code metrics serve as the predictor or independent variables.

As shown in Fig. 10, we conduct experiments with three different sets of source code metrics. One set of metrics consist of all the ten source code metrics. Another set consist of experiment with principal component analysis (PCA) as a preprocessing step for dimensionality reduction. PCA reduces dimensionality by selecting a subset of variables that preserves as much information present in the original set of variables. We also apply another technique using rough set theory for feature selection. We use rough set analysis to remove features with little and no effect on the dependent variable. These three sets of source code metrics are validated using hybrid approach of neural network and fuzzy logic i.e., Neuro-Fuzzy approach. Before applying Neuro-Fuzzy approach, data are normalized over the range between 0 to 1 i.e., [0 1] using Min–Max normalization technique (Kumar and Rath 2014). Normalization of data are required to adjust the defined range of input attribute and avoid the saturation of neurons. In this study, data are normalized using Min–Max normalization technique (Kaur et al. 2010). We also use fivefold cross validation to create different partitions of training and testing data and generalize the result of our analysis. Cross-validation method is based on statistical learning concept. It is used to compare and evaluate the models by separating the data into two groups. One group of separated data is used for learning or training the model and other data is used for validation of model. The basic form of cross-validation method is K-fold cross-validation method. In this method, data are separated into K equal or nearly equal size group or folds. For each model, $K-1$ group or folds data are used for learning or training the

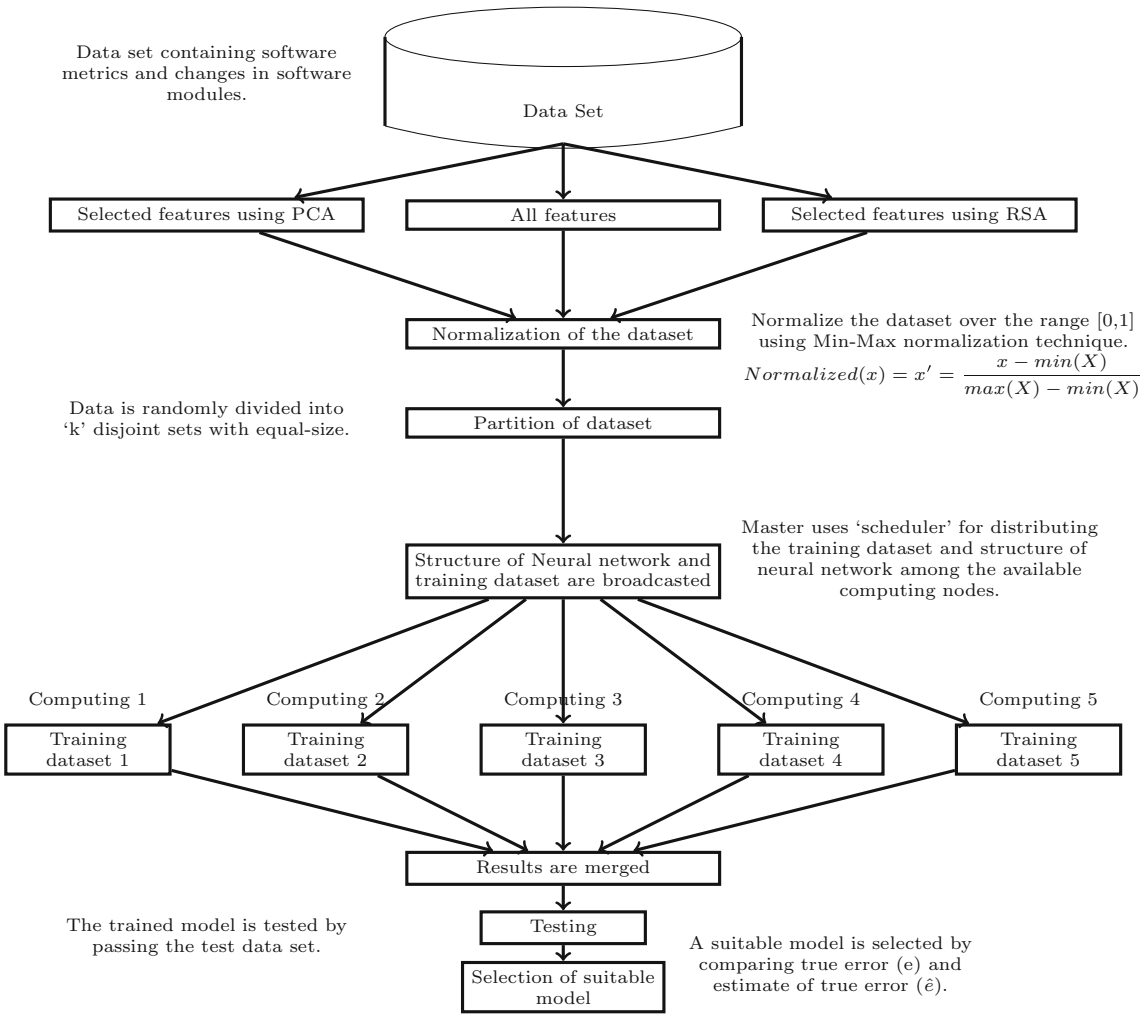

Fig. 10 Framework of proposed work

model and one group data are used for validation of model. The objective of K-fold cross-validation method is to use each data point for both training and validation.

In this study, different subsets of static source code metrics are considered as input to develop a maintainability prediction model using Neuro-Fuzzy approach. Mean absolute error (MAE), mean absolute relative error (MARE), and standard error of the mean (SEM) are taken as a performance parameter to compare the models. Parameters like True error (e) and Estimate of true error ($\hat{e}$) are being used for evaluating models involving cross validation approach.

7.1 Performance evaluation

In this paper, a three layered architecture of Fuzzy BP is considered. The software metric was used as input data to train the network using Fuzzy BP. This input data is represented using LR type Fuzzy number. Figure 11 shows the triangular membership function in which C_1, C_2, C_3 are the cluster centers which are found by using K-mean clustering algorithm. Table 9 contains the cluster centers C_1, C_2, C_3 of software metrics suite for UIMS and QUES.

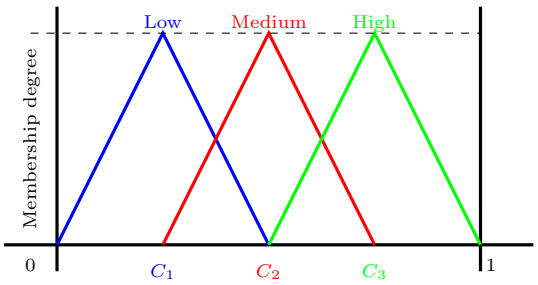

Fig. 11 Symmetric triangular LR-type fuzzy number

Table 8 shows the LR-type fuzzy number equivalents of fuzzy terms for UIMS and QUES respectively. In this paper, fivefold cross-validation concept has been considered for both QUES and UIMS for comparing the models. Table 10 shows the obtained performance metrics for UIMS and QUES software products.

From Table 10, it may be concluded that the performance in estimating software maintainability is better when PCA in UIMS and QUES (A2 analysis) is considered. Figure 12 shows the variance of mean square error and iteration number of UIMS and QUES.

In this work, Pearson residual boxplots are used for visual comparison between all developed models. Figure 13 shows the Pearson residual boxplots for all developed models. Boxplot diagrams help to observe performance of all developed maintainability prediction based on a single diagram. The top and bottom of each box represent the 75 and 25% of Pearson residual. The line in the middle of each box represents the median of the Pearson residual. The 1st and 2nd sub-figure of Fig. 13 show the Pearson residual for UIMS and QUES case studies. From Fig. 13, it may be observed that, in both case studies i.e., UIMS and QUES, analysis A2 has narrowest box and the smallest whiskers, as well as the few number of outliers. Based on these boxplots, it is evident that analysis A2 shows best estimation accuracy as compared to other two analysis i.e., A1 and A3. Hence it is observed that the model developed by considering extracted set of features using PCA feature selection technique, yields better maintenance accuracy value.

7.2 Parallel computing concepts

Core i5 processor with 4GB RAM and storage capacity of 250GB hard disk are the hardware items used in this study. Here computing node is processed that runs on a physical

Table 8 LR-Type fuzzy no. equivalent for fuzzy sets associated with software metrics

Metrics	UIMS			QUES		
	Low	Medium	High	Low	Medium	High
DIT	(0.0, 0.00, 0.25)	(0.25, 0.25, 0.37)	(0.62, 0.37, 0.37)	(0.00, 0.00, 0.25)	(0.25, 0.25, 0.27)	(0.52, 0.27, 0.47)
WMC	(0.02, 0.02, 0.13)	(0.16, 0.13, 0.47)	(0.64, 0.47, 0.35)	(0.06, 0.06, 0.29)	(0.35, 0.29, 0.36)	(0.71, 0.36, 0.28)
RFC	(0.09, 0.09, 0.17)	(0.26, 0.17, 0.34)	(0.60, 0.34, 0.39)	(0.13, 0.13, 0.22)	(0.35, 0.22, 0.30)	(0.76, 0.30, 0.23)
NOC	(0.00, 0.00, 0.12)	(0.12, 0.12, 0.48)	(0.60, 0.48, 0.39)	–	–	–
MPC	(0.02, 0.02, 0.22)	(0.23, 0.22, 0.42)	(0.66, 0.42, 0.33)	(0.17, 0.17, 0.24)	(0.41, 0.24, 0.38)	(0.80, 0.38, 0.19)
LCOM	(0.09, 0.09, 0.13)	(0.22, 0.13, 0.59)	(0.82, 0.59, 0.17)	(0.04, 0.04, 0.32)	(0.37, 0.32, 0.35)	(0.72, 0.35, 0.27)
NOM	(0.10, 0.10, 0.19)	(0.29, 0.19, 0.56)	(0.86, 0.56, 0.13)	(0.02, 0.02, 0.22)	(0.30, 0.22, 0.31)	(0.62, 0.31, 0.37)
DAC	(0.02, 0.02, 0.13)	(0.15, 0.13, 0.65)	(0.80, 0.65, 0.19)	(0.04, 0.04, 0.12)	(0.16, 0.12, 0.29)	(0.45, 0.29, 0.54)
SIZE2	(0.02, 0.02, 0.10)	(0.13, 0.10, 0.63)	(0.76, 0.63, 0.23)	(0.06, 0.06, 0.14)	(0.21, 0.14, 0.33)	(0.55, 0.33, 0.44)
SIZE1	(0.06, 0.06, 0.15)	(0.22, 0.15, 0.57)	(0.79, 0.57, 0.20)	(0.05, 0.05, 0.24)	(0.30, 0.24, 0.31)	(0.61, 0.31, 0.38)

Table 9 Cluster Center of software metrics

	UIMS			QUES		
	C1	C2	C3	C1	C2	C3
DIT	0.000	0.2500	0.6290	0.000	0.2500	0.5205
WMC	0.0232	0.1628	0.6425	0.0612	0.3585	0.7170
RFC	0.0913	0.2626	0.6044	0.1330	0.3573	0.7618
NOC	0.000	0.1250	0.6071	–	–	–
MPC	0.0121	0.2364	0.6623	0.1726	0.4171	0.8028
LCOM	0.0917	0.2250	0.82222	0.0466	0.3722	0.7291
NOM	0.1038	0.2978	0.8667	0.0262	0.3085	0.6208
DAC	0.0216	0.1587	0.8095	0.0432	0.1632	0.4533
SIZE2	0.0275	0.1300	0.7645	0.0686	0.2173	0.5553
SIZE1	0.0618	0.2213	0.7920	0.0554	0.3006	0.6172

Table 10 Performance matrix

	MAE	MARE	SEM
QUES			
A1 (FULL)	0.1490	0.3449	0.0257
A2 (PCA)	0.1518	0.3375	0.0095
A3 (RST)	0.0947	0.3377	0.228
UIMS			
A1 (FULL)	0.0711	0.4018	0.0887
A2 (PCA)	0.0631	0.2826	0.0672
A3 (RST)	0.1461	0.3366	0.0479

core. In this study, various number of computing nodes are considered ranging from one to five. fivefold cross validation techniques are used for both case studies, with the following steps:

- In case of five computing nodes, each fold data is assigned to each and every computing node.
- In case of four computing nodes, two folds data are assigned to first computing node and onefold of data to the remaining computing nodes.
- In case of three computing nodes, two folds data are assigned to first two computing nodes and onefold of data to the remaining computing nodes.
- In case of two computing nodes, first three folds of data set are assigned to first computing nodes and reaming two folds data to the second computing nodes.
- Finally, in case of single computing node, all the fivefolds data are assigned to the existing single computing node.

Figure 14 shows the time taken for training the model based on number of computing nodes. From Fig. 14, it can be inferred that normal process take less time as compared to single node on parallel computing process. Further, it is observed the time taken for training the model is reduced on an average 21.14% in QUES and 18.70% in UIMS, when the number of computing nodes are increased.

7.3 Comparison of models

From literature, it is observed that some authors such as Zhou and Leung (2007), Koten and Gray (2006), Elish and Elish (2009), Al-Jamimi et al. (2012), Aljamaan et al. (2013), Kumar and Rath (2015) and Kumar and Rath (2015) used same case studies for maintainability prediction using methods. They had considered performance parameter 'MMRE' to compare the performance of maintainability prediction models. Table 11 shows the MMRE value of the proposed work and the work done by Zhou and Leung (2007), Koten and Gray (2006), Elish and Elish (2009), Al-Jamimi et al. (2012), Aljamaan et al. (2013), Kumar and Rath (2015). From Table 11, it can be observed that, MMRE value is almost same in case of QUES, but our proposed model obtained better performance for maintainability prediction in case of UIMS. The proposed technique reduces computational time of Kumar and Rath (2015) by approx 50%.

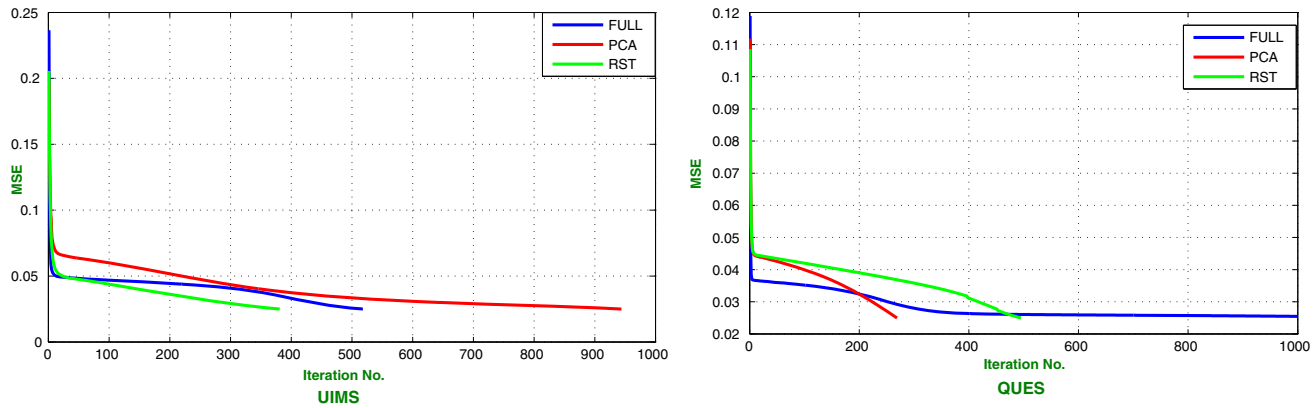

Fig. 12 MSE versus number of iterations (epoch)

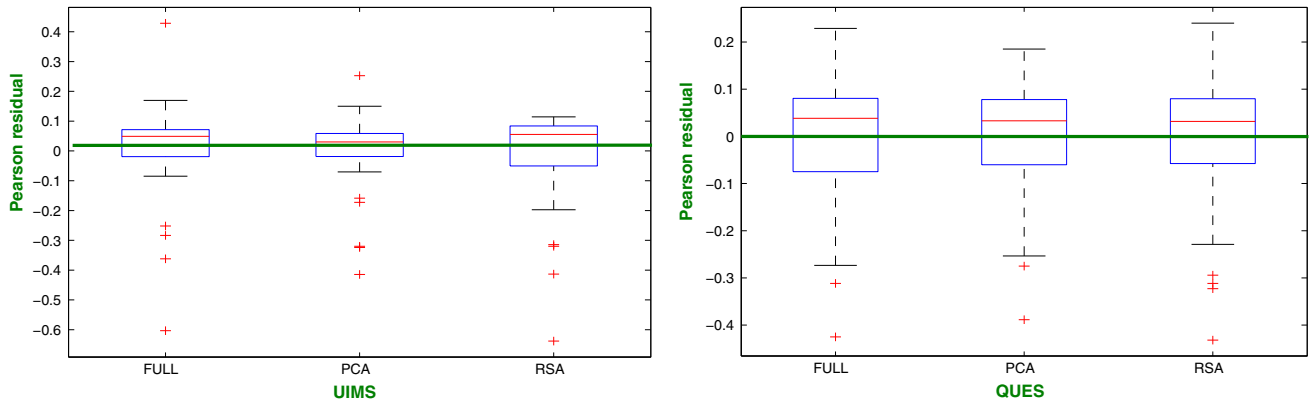

Fig. 13 Residual boxplot for UIMS and QUES

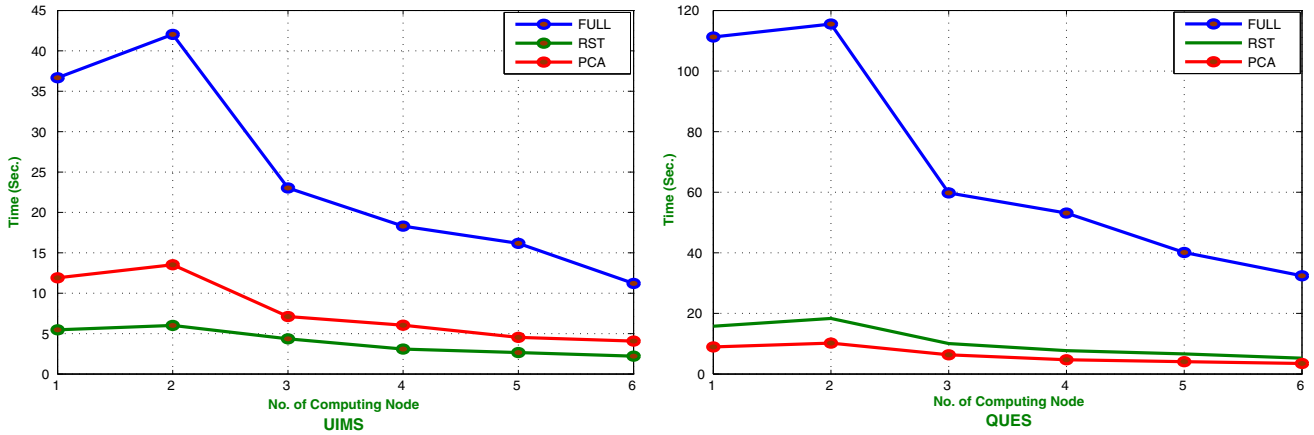

Fig. 14 Training time versus number of computing node

8 Threats to validity

For the sake of completeness, it is necessary to indicate some of the existing threats to validity of the proposed work. In literature, some of these limitations are very common when any empirical study is conducted (Malhotra and Chug 2014; Zhou and Leung 2007; Koten and Gray 2006; Elish and Elish 2009; Al-Jamimi et al. 2012; Chandra 2012; Aljamaan et al. 2013). Like most of the existing works, the proposed work also suffers from following threats:

1. All case studies are designed in ADa language. However the models designed in this study are likely to be valid for other object-oriented programing languages. Further research can be extended to design a model for other programing paradigms too.
2. In this work, only eleven different source code metrics are considered for development of maintainability prediction models. Some of the static source code object-oriented metrics which are also used for object-oriented software can be further considered for maintainability prediction.
3. Number of psychological factors also affect the reliability of software. But in this study, factors such as history of development of the system, stakeholders of the system, different level of expertise for developers, standards in which software is developed are not considered.

9 Conclusion

In this paper, an effort has been made to design a model for maintainability prediction of object-oriented system by considering software metrics as input. Neuro-Fuzzy approach: Hybrid neural network and fuzzy logic approach is used to develop maintainability prediction model for two commercial object-oriented software. Training data parallelism concept was used to accelerate the training or learning procedure of Neuro-Fuzzy approach. The concept involved in usage of varying number of computing nodes was explored in this analysis. The performance of Neuro-Fuzzy model was assessed and compared with those of the work carried out by several researchers (Kumar and Rath 2015; Zhou and Leung 2007; Koten and Gray 2006; Elish and Elish 2009; Al-Jamimi et al. 2012; Aljamaan et al.

Table 11 Performance based on MMRE for UIMS and QUES

MMRE			
Author	Technique	UIMS	QUES
Koten and Gray (2006)	Bayesian network	0.972	0.452
	Regression tree	1.538	0.493
	Backward elimination	2.586	0.403
	Stepwise selection	2.473	0.392
Zhou and Leung (2007)	Multivariate linear regression	2.70	0.42
	Artificial neural network	1.95	0.59
	Regression tree	4.95	0.58
	SVR	1.68	0.43
	MARS	1.86	0.32
Elish and Elish (2009)	TreeNet	1.57	0.42
Al-Jamimi et al. (2012)	Fuzzy logic	0.89	0.36
Chandra (2012)	SVM (Radial Kernel)	1.636	0.36
Aljamaan et al. (2013)	MLP	1.39	0.71
	RBF	3.23	0.96
	SVM	1.64	0.44
Kumar and Rath (2015)	ANN (CK metrics)	0.6931	0.4384
	Neuro-GA (CK metrics)	0.5332	0.4180
Kumar and Rath (2015)	Neuro-GA	0.1883	0.3536
Proposed work	Neuro-Fuzzy approach	0.2826	0.3375

2013; Kumar and Rath 2015). The result shows that, Neuro-Fuzzy model can effectively predict the maintainability of object-oriented software systems. From this analysis, it is also observed that, the training time gets reduced on an average 21.14% in QUES and 18.70% in UIMS when the number of computing nodes are increased. Our result also suggested that, with the selected subset of metrics using feature selection techniques i.e., PCA, and RSA, predict maintainability with higher accuracy is more suitable.

In this work, analysis has been made, based on two commercial software datasets i.e., UIMS and QUES considered as case study, being developed in single language i.e., ADa. Future work of this study for other development paradigms, may be planned subsequently. Another interesting work in the future is to improve the performance of prediction models by coupling neural network with other techniques such as Clonal selection algorithm, particle swarm optimization etc.

References

Abreu FBE, Carapuca R (1994) Object-oriented software engineering: measuring and controlling the development process. In: Proceedings of the 4th international conference on software quality, vol 186, pp 1–8

Adeli H, Hung S-L (1994) Machine learning: neural networks, genetic algorithms, and fuzzy systems. Wiley, Hoboken

Aggarwal KK, Singh Y, Chandra P, Puri M (2005) Measurement of software maintainability using a fuzzy model. J Comput Sci 1(4):538–542

Aggarwal KK, Singh Y, Kaur A, Malhotra R (2006) Application of artificial neural network for predicting maintainability using object-oriented metrics. Trans Eng Comput Technol 15:285–289

Al-Jamimi H, Ahmed M et al (2012) Prediction of software maintainability using fuzzy logic. In: 3rd international conference on software engineering and service science (ICSESS), pp 702–705

Aljamaan H, Elish MO, Ahmad I (2013) An ensemble of computational intelligence models for software maintenance effort prediction. In: Advances in computational intelligence, pp 592–603

Bandi RK, Vaishnavi VK, Turk DE (2003) Predicting maintenance performance using object-oriented design complexity metrics. IEEE Trans Softw Eng 29(1):77–87

Banker RD, Datar SM, Kemerer CF, Zweig D (1993) Software complexity and maintenance costs. Commun ACM 36(11):81–94

Basili VR, Briand LC, Melo WL (1996) A validation of object-oriented design metrics as quality indicators. IEEE Trans Softw Eng 22(10):751–761

Baski D, Misra S (2011) Metrics suite for maintainability of extensible markup language web services. IET Softw 5(3):320–341

Binkley AB, Schach SR (1998) Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. In: Proceedings of the 20th international conference on software engineering, pp 452–455. IEEE Computer Society

Briand LC, Wüst J, Daly JW, Porter DV (2000) Exploring the relationships between design measures and software quality in object-oriented systems. J Syst Softw 51(3):245–273

Chandra D (2012) Support vector approach by using radial kernel function for prediction of software maintenance effort on the basis of multivariate approach. Int J Comput Appl 54(4):21–25

Chen J-C, Huang S-J (2009) An empirical analysis of the impact of software development problem factors on software maintainability. J Syst Softw 82(6):981–992
Chidamber SR, Kemerer CF (1994) A metrics suite for object-oriented design. IEEE Trans Softw Eng 20(6):476–493
Dagpinar M, Jahnke JH (2003) Predicting maintainability with object-oriented metrics—an empirical comparison. In: 2013 20th working conference on reverse engineering (WCRE), pp 155–164
Damaševičius R, Štuikys V (2010) Metrics for evaluation of metaprogram complexity. Comput Sci Inf Syst 7(4):769–787
Dubois D, Prade H (1979) Fuzzy real algebra: some results. IEEE Trans Softw Eng 2(4):327–348
Elish MO, Elish KO (2009) Application of TreeNet in predicting object-oriented software maintainability: a comparative study. In: 13th European conference on software maintenance and reengineering, 2009. CSMR'09, pp 69–78
Halstead MH (1977) Elements of software science. Elsevier Science, New York
Henderson-Sellers B (1996) Software metrics. Prentice-Hall, Englewood
Huang D, Chow TWS (2005) Effective feature selection scheme using mutual information. Neurocomputing 63:325–343
Jin C, Liu J-A (2010) Applications of support vector mathine and unsupervised learning for predicting maintainability using object-oriented metrics. In: Second international conference on multimedia and information technology (MMIT), 2010, pp 24–27
Jung H-W, Kim S-G, Chung C-S (2004) Measuring software product quality: a survey of ISO/IEC 9126. IEEE Softw 21(5):88–92
Kabir MM, Islam MM, Murase K (2010) A new wrapper feature selection approach using neural network. Neurocomputing 73(16):3273–3283
Kang BK, Bieman JM (1995) Cohesion and reuse in an object-oriented system. In: Proceedings of the ACM SIGSOFT symposium on software reuseability, pp 259–262. Seattle
Kaur J, Singh S, Kahlon KS, Bassi P (2010) Neural network—a novel technique for software effort estimation. Int J Comput Theory Eng 2(1):17–19
Kohavi R (1999) Relation between software metrics and maintainability. In: Proceedings of the FESMA99 international conference, federation of European Software Measurement Associations, Amsterdam, The Netherlands, pp 465–476
Van Koten C, Gray AR (2006) An application of bayesian network for predicting object-oriented software maintainability. J Mater Process Technol 48(1):59–67
Kumar L, Naik DK, Rath SK (2015) Validating the effectiveness of object-oriented metrics for predicting maintainability. Proc Comput Sci 57:798–806
Kumar L, Rath SK (2014) Hybrid neural network approach for predicting maintainability of object-oriented software. INFOCOMP J Comput Sci 13(2):10–21
Kumar L, Rath SK (2015) Neuro-genetic approach for predicting maintainability using Chidamber and Kemerer software metrics suite. Recent Adv Inf Commun Technol 2015:31–40
Kumar L, Rath SK (2015) Predicting object-oriented software maintainability using hybrid neural network with parallel computing concept. In: Proceedings of the 8th India software engineering conference, pp 100–109
Li W, Henry S (1993) Maintenance metrics for the Object-Oriented paradigm. In: Proceedings of first international software metrics symposium, pp 52–60
Lorenz M, Kidd J (1994) Object-oriented software metrics. Prentice-Hall, Englewood
Malhotra R, Chug A (2014) Application of group method of data handling model for software maintainability prediction using object oriented systems. Int J Syst Assur Eng Manag 5(2):165–173
McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 2(4):308–320
Menzies T, Chen Z, Hihn J, Lum K (2006) Selecting best practices for effort estimation. IEEE Trans Softw Eng 32(11):883–895
Misra SC (2005) Modeling design/coding factors that drive maintainability of software systems. Softw Qual J 13(3):297–320
Misra S, Akman I (2008) Applicability of Weyuker's properties on oo metrics: some misunderstandings. Comput Sci Inf Syst 5(1):17–23
Misra S, Akman I, Koyuncu M (2011) An inheritance complexity metric for object-oriented code: a cognitive approach. Sadhana 36(3):317
Misra S (2007) Cognitive program complexity measure. In: Cognitive informatics, 6th IEEE international conference on, pp 120–125. IEEE
Oman P, Hagemeister J (1994) Construction and testing of polynomials predicting software maintainability. J Syst Softw 24(3):251–266
Pawlak Z (1982) Rough sets. Int J Comput Inform Sci 11(5):341–356
Riaz M, Mendes E, Tempero E (1997) Predicting maintenance effort with function points. Int Conf Softw Maint 1997:32–39
Riaz M, Mendes E, Tempero E (2009) A systematic review of software maintainability prediction and metrics. In: Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement, pp 367–377
Schneberger SL (1997) Distributed computing environments: effects on software maintenance difficulty. J Syst Softw 37(2):101–116
Zhou Y, Baowen X (2008) Predicting the maintainability of open source software using design metrics. Wuhan Univ J Nat Sci 13(1):14–20
Zhou Y, Leung H (2007) Predicting object-oriented software maintainability using multivariate adaptive regression splines. J Syst Softw 80(8):1349–1361

Lov Kumar

S. K. Rath