COMPARATIVE ANALYSIS OF MACHINE LEARNING TECHNIQUES FOR SOFTWARE RELIABILITY PREDICTION

BONTHU KOTAIAH, T. ARUNDHATHI, PATHAN MEHRA JAHAN

Abstract: In our paper, we try to discuss the models which are developed to accurately forecast software reliability. Various intelligent techniques (back propagation trained neural network, dynamic evolving neuro-fuzzy inference system and TreeNet) based models are discussed and presented. Three linear ensembles and one non-linear ensemble are designed and tested. Based on the experiments performed on the software reliability data obtained from literature, it is observed that the non-linear ensemble outperformed all the other ensembles and also the constituent statistical and intelligent techniques for the effective assessment of software reliability.

Keywords: Software reliability, Assessment and forecasting; Operational risk; Assemble forecasting model; Machine Learning Techniques; Soft computing; Neural Networks, Fuzzy Systems.

Introduction: Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment (ANSI definition). Software reliability modeling has gained a lot of importance in the recent years. Criticality of software in many of the present day applications has led to a tremendous increase in the amount of work being carried out in this area. The use of intelligent neural network and hybrid techniques in place of the traditional statistical techniques have shown a remarkable improvement in the prediction of software reliability in the recent years. Among the intelligent and the statistical techniques it is not easy to identify the best one since their performance varies with the change in data.

In this paper, an ensemble-based approach is followed in predicting software reliability. Specifically, a non-linear ensemble trained using back propagation neural network (BPNN) is proposed. The proposed approach takes the advantage of all the techniques' prediction capabilities towards

the data and appropriately assigns weights to each of the techniques based upon their performance.

The rest of the paper is organized in the following manner. In Section 1, a brief review of the works carried out in the area of software reliability prediction in research is presented. In Section 2, the various stand-alone intelligent methods that are applied in this paper are described briefly. In Section 3, the four ensembles that are developed are presented. Section 4 presents the experimental methodology; discussion of the results is presented in Section 5. Finally, Section 6 concludes the paper.

Literature survey: In the last few years many research studies has been carried out in this area of software reliability modeling and forecasting. They included the application of neural networks, fuzzy logic models; Genetic algorithms (GA) based neural networks, recurrent neural networks, Bayesian neural networks, and support vector machine (SVM) based techniques, to name a few. Cai et al. (1991) advocated the development of fuzzy software reliability models in place of probabilistic software reliability models (PSRMs). Their argument was based on the proof that software reliability is fuzzy in nature. A demonstration of how to develop a fuzzy model to characterize software reliability was also presented. Karunanithi et al. (1992) carried out a detailed study to explain the use of connectionist models in software reliability growth prediction. It was shown through empirical results that the connectionist models adapt well across different datasets and exhibit better predictive accuracy than the well-known analytical software reliability growth models. Sitte (1999) made a comparative study of neural networks and parametric-recalibration models in software reliability prediction and found neural networks to be much simpler to use and also to be better predictors. Also, through empirical results it was shown that the neural network models are better trend predictors. Ho et al. (2003) performed a comprehensive study of con- nectionist models and their applicability to software reli-ability prediction and found them to be better and more flexible than the traditional models. A comparaitive study was performed between their proposed modified Elman recurrent neural network, with the more popular feed forward neural network, the Jordan recurrent model, and some traditional software reliability growth models. Numerical results show that the proposed network architecture performed better than the other models in terms of predictions. Despite of the recent advancements in the software reliability growth models, it was observed that different

models have different predictive capabilities and also no single model is suitable under all circumstances.

Tian and Noore (2005a) proposed an on-line adaptive software reliability prediction model using evolutionary connectionist approach based on multiple-delayed-input single-output architecture. The proposed approach, as shown by their results, had a better performance with respect to next-step predictability compared to existing neural network model for failure time prediction. Tian and Noore (2005b) proposed an evolutionary neural network modeling approach for software cumulative failure time prediction. Their results were found to be better than the existing neural network models. It was also shown that the neural network architecture has a great impact on the performance of the network. According to Bai et al. (2005) Bayesian networks show a strong ability to adapt in problems involving complex variant factors. They developed a software prediction model based on Markov Bayesian networks, and a method to solve the network model was proposed. Reformat (2005) proposed an approach leading to a multitechnique knowledge extraction and development of a comprehensive meta-model prediction system in the area of corrective maintenance of software. The system was based on evidence theory and a number of fuzzy-based models. In addition they carried out a detailed case study for estimating the number of defects in a medical imaging system using the proposed approach. Pai and Hong (2006) have applied support vector machines (SVMs) for forecasting software reliability where simulated annealing (SA) algorithm was used to select the parameters of the SVM model. The experimental results show that the proposed model gave better predictions than the other compared methods. Su and Huang (2006) showed how to apply neural networks to predict software reliability. Further they made use of the neural network approach to build a dynamic weighted combinational model (DWCM) and experimental results show that the proposed model gave significantly better predictions. Also recently, neural networks were applied for predicting faults in object-oriented software (Kanmani et al., 2007). The study showed neural network models to be performing much better than the statistical methods.

Application of intelligent techniques in place of the statistical techniques has increased by leaps and bounds in the recent years. Application of Soft Computing techniques in software reliability engineering has come up

recently (Madsen et al., 2006). Despite the recent advancements in the software reliability growth models, it was observed that different models have different predictive capabilities and also no single model is suitable under all circumstances. An ensemble uses the output obtained from the individual constituents as inputs to it and the data is processed according to the design of the arbitrator lying at the heart of the ensemble.

Overview of the techniques applied: The following techniques are applied to predict software reliability (i) generalized regression neural network (GRNN), (ii) dynamic evolving neuro-fuzzy inference system (DENFIS). All these machine learning techniques are very popular in effectively assessing the Software Reliability. All these two constituents of the ensembles are described briefly in the subsequent subsections.

Generalized regression neural network (GRNN): Specht(1991) introduced GRNN. It can be thought of as a normalized radial basis function (RBF) network in which there is a hidden unit centered at every training case. These RBF units are called "kernels" and are usually probability density functions such as the Gaussian. The hidden- to-output weights are just the target values, so the output is simply a weighted average of the target values of training cases close to the given input case. The only weights that need to be learned are the widths of the RBF units. These widths (often a single width is used) are called "smoothing parameters" or "bandwidths" and are usually chosen by cross-validation or by more esoteric methods that are not well known in the neural net literature; gradient descent is not used. GRNN is a universal approximator for smooth functions, so it should be able to solve any smooth function-approximation problem given enough data. The main drawback of GRNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. GRNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. GRNN available in MAT- LAB 6.5 was used in the paper.

Dynamic evolving neuro-fuzzy inference system (DENFIS): DENFIS was introduced by Kasabov (2002). DENFIS evolve through incremental, hybrid (supervised/unsupervised) learning, and accommodate new input data, including new features, new classes, etc., through local element tuning. New fuzzy rules are created and updated during the operation of the system. At each time moment, the output of DENFIS is calculated through a fuzzy inference system based on most activated fuzzy rules, which are dynamically chosen from a fuzzy rule set. A set of fuzzy rules

can be inserted into DENFIS before or during its learning process. Fuzzy rules can also be extracted during or after the learning process.

Ensemble forecasting models: The idea behind ensemble systems is to exploit each constituent model's unique features to capture different patterns that exist in the dataset. Both theoretical and empirical works indicate that ensembling can be an effective and efficient way to improve accuracies. Bates and Granger (1969) in their seminal work showed that a linear combination of different techniques would give a smaller error variance than any of the individual techniques working in stand-alone mode. Since then, many researchers worked on ensembling or combined forecasts. Makridakis et al. (1982) reported that combining several single models has become common practice in improving forecasting accuracy. Then, Pelikan et al. (1992) proposed combining several feed-forward neural networks to improve time series forecasting accuracy. Some of the ensemble techniques for prediction problems with continuous dependent variable include linear ensemble (e.g., simple average; Bene- diktsson et al., 1997), weighted average (Perrone and Cooper, 1993) and stacked regression (Breiman, 1996) and non-linear ensemble (e.g., neural-network-based nonlinear ensemble (Yu et al., 2005)). Hansen et al. (1992) reported that the generalization ability of a neural network system could be significantly improved by using an ensemble of a number of neural networks. The purpose is to achieve improved overall accuracy on the production data. In general, for classification problems, an ensemble system combines individual classification decisions in some way, typically by a majority voting to classify new examples. The basic idea is to train a set of models (experts) and allow them to vote. In majority voting scheme, all the individual models are given equal importance. Another way of combining the models is via weighted voting, wherein the individual models are treated as unequally important.This is achieved by attaching some weights to the prediction given by the individual models and then combine them. Olmeda and Fernandez (1997) presented a genetic algorithm based ensemble system, where a GA determines the optimal combination of the individual models so that the accuracy is maximized. Zhou et al. (2002) carried out a detailed study on ensembling neural networks and proposed that using a set of neural networks to form an ensemble is better than to use all the neural networks. They proposed an approach that can be used to select the neural networks to become part of the ensemble from the available set of neural networks. Genetic algorithm

was used to assign weights to the constituent networks.

It is generally the case that for a given dataset one kind of intelligent technique outperforms the other and the results can be entirely opposite when a different dataset is used. In order not to lose any generality and also to combine the advantages of the intelligent techniques, an ensemble uses the outputs of all the stand-alone intelligent techniques with each being assigned a certain priority level and provides the output with the help of an arbitrator.

An ensemble uses the output obtained from the individual constituents as inputs to it and the data is processed according to the design of the arbitrator. Four different variants of ensembles are designed and employed as shown in Figs. 1 and 2. These include (i) linear ensemble based on average, (ii) linear ensemble based on weighted mean, (iii) linear ensemble based on weighted median, and finally (iv) a non-linear ensemble based on BPNN. These ensembles are described briefly below.

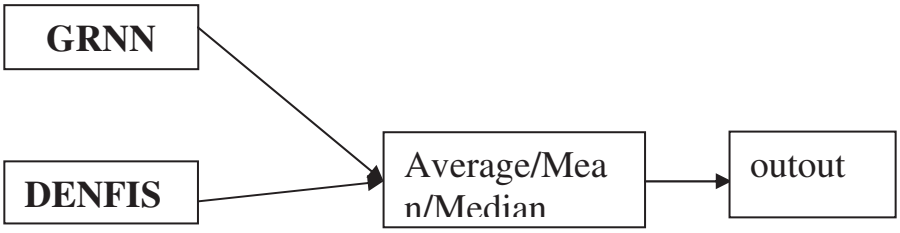

Fig.1: Generic design of Linear ensemble.

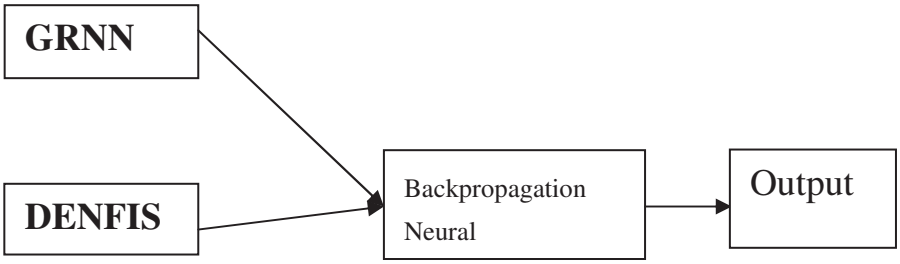

Fig.2: Generic design of Non Linear ensemble.

Linear ensemble based on average: For each observation, the output

values of the individual components are taken as the input to the ensemble and the average of these values is output by the ensemble. This is the simplest kind of ensemble one can imagine.

Linear ensemble based on weighted mean: In this ensemble, the individual output values are not taken as they are but are given weights based upon certain criteria set by the user. In this case, the criteria of setting the weightages is based on the mean of the normalized root mean square error (NRMSE) values over the individual lags on the test data. The lower the mean the higher the weight age with the condition that the sum of all the weights is equal to one. This helps in setting the priority towards a technique based on its performance.

Linear ensemble based on weighted median: It is similar to the linear ensemble based on weighted mean, except that the median of the NRMSE values of the individual techniques on the test data is considered in assigning the weight ages instead of the mean of the values.

Neural network based non-linear ensemble: Here, no assumptions are made about the input that is given to the ensemble. The output values of the individual techniques are fed into an arbitrator, which is a back propagation neural network (BPNN) which when trained, assigns the weights accordingly.

Experimental design: Because software reliability forecasting has only one dependent variable and no explanatory variables in the strict sense and since we have a time-series, we followed the general time series forecasting model in conducting our experiments, which is represented in the following form (as shown in Eq. (1)):

$X_t = f(X')$ .....(1)

where *X'* is vector of lagged variables $\{x_{t-1}, x_{t-2}, \ldots, x_{t-p}\}$. Hence the key to finding the solution to the forecasting problems is to approximate the function 'f'. This can be done by iteratively adjusting the weights in the modeling process.

An illustration of how training patterns can be designed in the neural network modeling process is provided in Fig. 3 (Xu et al. (2003)). In this figure, 'p denotes the number of lagged variables and (t-p*)* denotes the total number of training samples.In this representation, '*X*' is a set of (t-p*)* vectors of dimension 'p and '*Y* is a vector of dimension *(t-p*). Thus, in the transformed data set, '*X*' and '*Y*' represent the vector of explanatory variables and dependent variable, respectively.

In this study, the software failure data, presented in Table 1, is obtained

from Musa (1979). It is used to demonstrate the forecasting performance of the proposed ensembles.

X				**Y**
x_1	x_2	...	x_p	x_{p+1}
x_2	x_3	...	x_{p+1}	x_{p+2}
x_3	x_4	...	x_{p+2}	x_{p+3}
.	.	...	.	.
.	.	...	.	.
.	.	...	.	.
x_{t-p}	x_{t-p+1}	....	x_{t-1}	x_t

Fig. 3. Design of the training patterns.

The data contains 101 observations of the pair (t, Yt) pertaining to software failure. Here Yt represents the time to failure of the software after the t th modification has been made. SPSS 14.0 obtained from (http:// www.spss.com) was used to find the optimal lag for the given time-series data. We performed the tests of 'auto correlation function' and 'partial auto correlation function' as prescribed by Box-Jenkins methodology in Time series forecasting using SPSS 14.0 software on the data set and found that lag 1 was sufficient for the data set. However, we wanted to investigate whether NRMSE values would improve further when we go for higher lags and we tested up to lag 5. In view of the foregoing discussion on generating lagged data sets out of the original time series such as this, we created five datasets corresponding to lag # 1, 2, 3, 4 and 5, respectively.

Table 1 Data of software failures					
t	Yt	t	Yt	t	Yt
0	5.7683	34	10.6301	68	12.5982
1	9.5743	35	8.3333	69	12.0859
2	9.105	36	11.315	70	12.2766
3	7.9655	37	9.4871	71	11.9602
4	8.6482	38	8.1391	72	12.0246
5	9.9887	39	8.6713	73	9.2873
6	10.1962	40	6.4615	74	12.495
7	11.6399	41	6.4615	75	14.5569
8	11.6275	42	7.6955	76	13.3279
9	6.4922	43	4.7005	77	8.9464
10	7.901	44	10.0024	78	14.7824
11	10.2679	45	11.0129	79	14.8969
12	7.6839	46	10.8621	80	12.1399
13	8.8905	47	9.4372	81	9.7981
14	9.2933	48	6.6644	82	12.0907
15	8.3499	49	9.2294	83	13.0977
16	9.0431	50	8.9671	84	13.368
17	9.6027	51	10.3534	85	12.7206
18	9.3736	52	10.0998	86	14.192
19	8.5869	53	12.6078	87	11.3704
20	8.7877	54	7.1546	88	12.2021
21	8.7794	55	10.0033	89	12.2793
22	8.0469	56	9.8601	90	11.3667
23	10.8459	57	7.8675	91	11.3923
24	8.7416	58	10.5757	92	14.4113
25	7.5443	59	10.9294	93	8.3333
26	8.5941	60	10.6604	94	8.0709
27	11.0399	61	12.4972	95	12.2021
28	10.1196	62	11.3745	96	12.7831
29	10.1786	63	11.9158	97	13.1585
30	5.8944	64	9.575	98	12.753
31	9.546	65	10.4504	99	10.3533
32	9.6197	66	10.5866	100	12.4897
33	10.3852	67	12.7201		

Since it is a time-series data, performing 10-fold cross validation does not

make sense, as it involves randomly choosing samples into the folds and then the time aspect of the data gets obscured and overlooked. 10-fold cross validation is extremely powerful and useful in assessing the performance of a model, provided we do not deal with time series or spatial series data. Hence, we carried out hold-out method of testing viz., splitting the data set into 80% and 20%, respectively for training and testing. In fact, this check is included in many popular commercial data mining/statistical tools. The training data is used to identify the optimal parameters for the model that satisfy the given error criteria and those parameters are the used to forecast values on the test set. The value of normalized root mean square error (NRMSE) is used as the measurement criteria.

Root mean square error (RMSE): The Root Mean Square Error (**RMSE**) (also called the root mean square deviation, RMSD) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled. These individual differences are also called residuals, and the RMSE serves to aggregate them into a single measure of predictive power. The RMSE of a model prediction with respect to the estimated variable X_{model} is defined as the square root of the mean squared error:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (X_{obs,i} - X_{model,i})^2}{n}}$$

where X_{obs} is observed values and X_{model} is modelled values at time/place *i*. The calculated RMSE values will have units, and RMSE for phosphorus concentrations can for this reason not be directly compared to RMSE values for chlorophyll *a* concentrations etc. However, the RMSE values can be used to distinguish model performance in a calibration period with that of a validation period as well as to compare the individual model performance to that of other predictive models.

Normalized root mean square error (NRMSE): Non-dimensional forms of the RMSE are useful because often one wants to compare RMSE with different units. There are two approaches: normalize the RMSE to the range of the observed data, or normalize to the mean of the observed data.

$$NRMSE = \frac{RMSE}{X_{obs,\max} - X_{obs,\min}}$$

$$NRMSE = \frac{RMSE}{\overline{X_{obs}}}$$

(the latter one is also called C_V,RMSE for the resemblance with calculating the coefficient of variance).

Results and discussion: For each technique, the appropriate parameters, as specified by the algorithm, are tweaked to get the most optimal results. Table 2 illustrates the NRMSE values of different lags of data obtained over different techniques. The parameters are tweaked until the least NRMSE values computed using Eq. (2) could be obtained and the best values are presented in Table 2. For a given lag, the test results obtained from these individual techniques are presented to different ensembles. The NRMSE values obtained from the ensembles for different lags are presented in Table 3.

The parameters over which GRNN gave the best results over different lags are summarized in Tables 4-7, respectively. Also, parameters over which the non-linear ensemble trained with BPNN gave the best results are presented in Table 8. These values are obtained by trial and error. In selecting the constituents for the ensemble, the performance of the individual techniques over all the lags (Tables 2 and 3) is considered and accordingly the best five among the techniques - GRNN and DENFIS are selected to become part of the ensemble. Interesting observations can be drawn from Tables 2 and 3. First, there seems to be a correlation between the lag numbers and the corresponding NRMSE value. We noticed that as the lag increases the NRMSE value decreases. Second, for the individual lags, GRNN seemed to outperform all the other techniques in the stand-alone mode, although other techniques such as DENFIS performed consistently well over all the lags.

Third, the ensembles yielded better results than any of the individual techniques with some exceptions. For instance, for lag1, GRNN is found to be better than the three linear ensembles and for lag2, GRNN outperformed the linear ensembles. Finally, the non-linear ensemble built using GRNN as the arbitrator, outperformed all the other constituent techniques in the stand-alone mode and all linear ensembles over all the

lags. Within the non-linear ensemble, the least NRMSE value is obtained for lag1 and also the difference of the NRMSE values over all other lags is very minimal.

Table 2 NRMSE values on test data for the selected 2 different techniques in standalone mode					
	Lag1	Lag2	Lag3	Lag4	Lag5
GRNN	0.210247	0.211408	0.176769	0.179869	0.166883
DENFIS	0.170907	0.267306	0.18425	0.189379	0.293461

Table 3 NRMSE values on Test data for the various ensembles					
	Lag1	Lag2	Lag3	Lag4	Lag5
Linear ensemble based on	0.168961	0.166962	0.147629	0.145939	0.143424
Linear ensemble based on	0.170045	0.166926	0.147439	0.146003	0.143463
Linear ensemble based on	0.170037	0.166901	0.147187	0.145898	0.143399
Non-linear ensemble based on BPNN	0.130723	0.136737	0.132911	0.136644	0.136328

Table 4 Details of GRNN structure and parameters over different lags					
	Lag1	Lag2	Lag3	Lag4	Lag5
Number of input nodes	1	2	3	4	5
Number of hidden nodes	4	4	8	10	8
Learning rate	0.1	0.1	0.1	0.1	0.1
Momentum rate	0.1	0.2	0.2	0.016	0.11

Table 5:Details of TANN structure and values for diff.					
	Lag1	Lag2	Lag3	Lag4	Lag5
Number of input	1	2	3	4	5
Number of hidden	4	7	7	7	7
Value of Pindex	29	33	25	27	27
Value of Epsilon	0.004	0.008	0.03	0.009	0.025

Table 6: Details of PSN structure and parameters for different					
	Lag1	Lag2	Lag3	Lag4	Lag5
Number of input nodes	1	2	3	4	5
Number of pattern nodes	80	79	78	77	76
Smoothing parameter	1.01	1.77	0.55	1.99	2.46

Table 7:Details of GRNN Structure and parameters over different lags					
	Lag1	Lag2	Lag3	Lag4	Lag5
Number of input nodes	1	2	3	4	5
Number of pattern nodes	80	79	78	77	76
Smoothing parameter	1.01	1.77	0.55	1.99	2.46

Table 8 Details of BPNN structure and parameters over different lags for the ensemble data					
	Lag1	*Lag2*	*Lag3*	*Lag4*	*Lag5*
Number of input nodes	5	5	5	5	5
Number of hidden	1	1	1	2	5
Learning rate	0.08	0.02	0.01	0.09	0.1
Momentum rate	0.44	0.36	0.39	0.22	0.15

lag2, where some techniques are better, the linear ensembles of all kinds showed better performance than the individual stand-alone techniques. Non-Linear ensemble is better than any other technique or ensemble over all kinds of data. Amongst the linear ensembles, the weighted mean and weighted median based ensembles yielded similar NRMSE values for all lags.

In this connection, we observe that ensembling is more time consuming than using intelligent methods in their stand-alone mode. However, it is believed that the gains accrued in the bargain in the form of improved accuracy more than offset the time lost. Further, we point out that, when

reliability prediction is to be made accurately in an offline manner, then time is no constraint and non-linear ensemble should be preferred. However,when time is a constraint, then, on-line methods like DENFIS should be preferred, as they need only one-pass or one-iteration to give predictions.

Further, we observe that Pai and Hong (2006) also used the same data set to test the efficacy of their support vector machine simulated annealing (SVMSA) method. However, since they did not use the lagged data in their experimentation our results cannot be compared with theirs. Further, they divided the data set of 101 observations into training (33 observations), validation (8 observations) and test (60 observations) sets. Since it is a non-standard method of splitting the data set for experimentation, we chose not to compare our results with theirs. The NRMSE value obtained on the test set by their experiments was 0.1562, which is not as good as the results of the proposed model.

Conclusion: In the paper, ensemble models are developed to forecast software reliability efficiently. Three linear ensembles and one non-linear ensemble are developed and tested to forecast software reliability. Various statistical and intelligent techniques constitute the ensembles. They are back propagation trained neural network (BPNN), dynamic evolving neuro-fuzzy inference system (DENFIS). Based on the numerical experiments conducted by us on the software reliability data obtained from literature, we noticed that the non-linear ensemble outperformed all the other ensembles and also the constituent statistical and intelligent techniques. Further, we noticed that the linear ensembles also outperformed the constituent techniques from lag3 onwards. In conclusion, the ensembles developed here can be used as viable alternatives to the existing methods for software reliability prediction.

References:

1. M.L., 1991. A critical review on software reliability modeling. Reliability Engineering and System Safety 32 (3), 357-371.
2. Dueck, G., Scheuer, T., 1990. Threshold accepting: a general-purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics 90, 161-175.
3. Ho, S.L., Xie, M., Goh, T.N., 2003. A study of connectionist models for software reliability prediction. Computers and Mathematics with

Applications 46 (7), 1037-1045.
4. Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V., Thambidurai, P., 2007. Object-oriented software failure fault prediction using neural networks. Information and Software Technology 49, 483-492.
5. Karunanithi, N., Whitley, D., Maliya, Y.K., 1992. Prediction of software reliability using connectionist models. IEEE Transactions on Software Engineering 18, 563-574.
6. Kasabov, N.K., 2002. DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. IEEE Transactions on fuzzy systems 10 (2).
7. Madsen, H., Thyregod, P., Burtschy, B., Albeanu, G., Popentiu, F., 2006. On using soft computing techniques in software reliability engineering. International Journal of Reliability, Quality and Safety Engineering 13 (1), 61-72.
8. Makridakis, S., Anderson, A., Carbone, R., Fildes, R., Hibdon, M., Lewandowski, R., Newton, J., Parzen, E., Winkler, R., 1982. The accuracy of extrapolation (time series) methods: results of a forecasting competition. Journal of Forecasting 1, 111-153.
9. Musa, J.D., 1979. Software reliability data. IEEE Computer Society - Repository.
10. Olmeda, I., Fernandez, E., 1997. Hybrid classifiers for financial multicriteria decision making: the case of bankruptcy prediction. Computational Economics 10, 317-335.
11. Perrone, M.P., Cooper, L.N., 1993. When networks disagree: ensemble methods for hybrid neural netwoks. In: Mammone, R.J. (Ed.), Neural Networks for speech and Image processing. Chapman Hall, pp. 126142.
12. Ravi, V., Zimmermann, H.J., 2001. A neural network and fuzzy rule base hybrid for pattern classification. Soft Computing 5 (2), 152-159.
13. Ravi, V., Zimmermann, H.J., 2003. Optimization of neural networks via threshold accepting: a comparison with backpropagation algorithm. In: Conference on Neuro-Computing and Evolving Intelligence, Auckland, New Zealand.
14. Ravi, V., Murty, B.S.N., Dutt, N.V.K., 2005. Forecasting the properties of carboxylic acids by a threshold accepting-trained neural network. Indian Chemical Engineer 47 (3), 147-151.
15. Reformat, M., 2005. A fuzzy-based multimodel system for reasoning about the number of software defects. International Journal of Intelligent Systems 20 (11), 1093-1115.

16. Shin, Y., Ghosh, J., 1991. The Pi-Sigma networks: an efficient higher-order neural network for pattern classification and function approximation. In: Proceedings of International Joint Conference on Neural Networks, 1, Seattle, Washington, pp. 13-18.
17. Sitte, R., 1999. Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration. IEEE Transactions on Reliability 48 (3), 285-291.
18. Specht, D.F., 1991. A general regression neural network. IEEE Transactions on Neural Networks 2 (6), 568-576.
19. Su, Y.-S., Huang, C.-Y., 2006. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. Journal of Systems and Software 80 (4), 606-615.
20. Tian, L., Noore, A., 2005a. On-line prediction of software reliability using an evolutionary connectionist model. The Journal of Systems and Software 77, 173-180.
21. Tian, L., Noore, A., 2005b. Evolutionary neural network modeling for software cumulative failure time prediction. Reliability Engineering and System Safety 87, 45-51.
22. Xu, K., Xie, M., Tang, L.C., Ho, S.L., 2003. Application of neural networks in forecasting engine systems reliability. Applied Soft Computing 2, 255-268.
23. Yu, L., Wang, S.Y., Lai, K.K., 2005. A novel non-linear ensemble forecasting model incorporating GLAR and ANN for foreign exchange rates. Computers and Operations Research 32 (10), 25232541.
24. Zhou, Z-H., Wu, J., Tang, W., 2002. Ensembling neural networks: many could be better than all. Artificial Intelligence 137, 239-263.

***

Bonthu Kotaiah /Research Scholar, Babasaheb Bhimrao Ambedkar University, Lucknow, India
T. Arundhathi /Assistant Professor, Maulana Azad National Urdu University, Hyderabad, India
Pathan Mehra Jahan/Assistant Professor, Maulana Azad National Urdu University, Hyderabad, India
kotaiah_bonthuklce@yahoo.com, arundathi.tunga21@gmail.com, pathan_mehra@yahoo.com