Neural Network and Statistical Modeling of Software Development Effort

Ruchi Shukla, Mukul Shukla and Tshilidzi Marwala

Abstract Many modeling studies that aimed at providing an accurate relationship between the software project effort (or cost) and the involved cost drivers have been conducted for effective management of software projects. However, the derived models are only applicable for a specific project and its variables. In this chapter, we present the use of back-propagation neural network (NN) to model the software development (SD) effort of 18 SD NASA projects based on six cost drivers. The performance of the NN model was also compared with a multi-regression model and other models available in the literature.

Keywords Neural network · Software development · Effort estimation · Regression

R. Shukla (✉)
Department of Electrical and Electronic Engineering Science, University of Johannesburg, Johannesburg, South Africa
e-mail: ruchishuklamtech@gmail.com

M. Shukla
Department of Mechanical Engineering Technology, University of Johannesburg, Johannesburg, South Africa

M. Shukla
Department of Mechanical Engineering, MNNIT, Allahabad, UP, India
e-mail: mukulshukla2k@gmail.com

T. Marwala
Faulty of Engineering and Built Environment, University of Johannesburg, Johannesburg, South Africa
e-mail: tmarwala@uj.ac.z

B. V. Babu et al. (eds.), *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28–30, 2012*, Advances in Intelligent Systems and Computing 236, DOI: 10.1007/978-81-322-1602-5_21,

1 Introduction

Software companies today are outsourcing a wide variety of their jobs to offshore organizations, for maximizing returns on investments. Estimating the amount of effort, time, and cost required for developing any information system is a critical project management issue. In view of the above, long-term, credible, and optimum forecast of software project estimates in the early stages of a project's life cycle is an almost intractable problem. Often, key information of real-life projects regarding size, complexity, system documentation, vocabulary, annual change traffic, client attitude, multilocation teams, etc. is unavailable. In spite of the availability of more than 100 estimation tools in the market, experience-based reasoning still remains the commonly applied estimation approach owing to some fundamental estimation issues which software developers have struggled with [1].

2 Literature Review

A review of studies on expert estimation of SD effort was presented by [2, 3]. An exploratory analysis of the state of the practice on schedule estimation and software project success prediction is presented in [4]. It was found that the data collection approach, role of respondents, and analysis type had an important impact on software estimation error [5]. Soft-computing- or artificial intelligence (AI)-based approaches are of late being used for more accurate prediction of software effort/cost. Artificial neural networks (ANN) offer a powerful computing architecture capable of learning from experimental data and representing complex, nonlinear, multivariate relationships [6, 7]. Kumar et al. compared the effectiveness of the variants of wavelet neural network (WNN) with many other techniques to forecast the SD effort [8]. Genetic algorithms (GAs) were used for the estimation of COCOMO model parameters of NASA SD projects in [9] while different fuzzy logic-based studies have been conducted [10–12]. Many hybrid schemes (neuro-GA, neuro-fuzzy, grey-GA, fuzzy-grey, etc) have also been investigated [13–15]. Many studies on software prediction have focused on the development of regression models based on historical data [16, 17].

3 Statistical Modeling

This modeling study is based on the SD effort dataset of Bailey and Basili [18] (Table 1—shown partly for brevity reasons). The six input factors are the total lines of code, new lines of code, developed lines of code (DL) (all in kloc), total methodology (ME), cumulative complexity, and cumulative experience, and the output is effort (in man months). Preliminary statistical analysis of the dataset was conducted

Table 1 SD effort dataset [18]

Project no.	Project attributes						Response
	Total lines (kloc)	New lines (kloc)	Developed lines (kloc)	Total methodology	Cumulative complexity	Cumulative experience	Effort (man months)
1	111.9	84.7	90.2	30	21	16	115.8
2	55.2	44	46.2	20	21	14	96
–	–	–	–	–	–	–	–
17	14.8	11.9	12.5	27	23	18	23.9
18	110.3	98.4	100.8	34	33	16	138.3
Correlation Coefficient	0.94	0.97	0.96	0.03	0.65	-0.02	
Covariance	1593.7	1319.3	1456.3	6.98	134.4	−2.82	
Kurtosis	−1.25	−0.38	−0.73	−0.83	−0.27	3.55	−1.26
R-Square	0.88	0.95	0.92	0.00	0.42	0.00	

beforehand including the following: (1) correlation coefficient, (2) covariance, (3) kurtosis, and (4) R-square as presented in Table 1. Initially, from Minitab [19]-based ANOVA, a multivariable linear regression model (Eq. 1) has been fitted. The goodness of this developed model is validated with two other models (Eqs. 2 and 3) given by Sheta and Al-Afeef [15] in Table 2. Based on the high T (or low P) values, the following ranking (in a decreasing order) of the 6 effort drivers has been established: (1) methodology, (2) new LoC, (3) total LoC, (4) cumulative experience, (5) developed LoC, and (6) cumulative complexity. The high R-squared value of 98.3 % and R-Sq(adjusted) values of 97.4 % justify the correctness of the ANOVA.

$$\begin{aligned} \text{Effort} = 41.6 + 0.314\ Tot\_LoC + 0.986\ New\_LoC + 0.116\ Develop\_LoC \\ - 1.57\ Meth - 0.112\ Cum\_Complex + 0.376\ Cum\_Exper \end{aligned} \tag{1}$$

$$E = 1.75992 \times DL - 4.56 \times 10^{-3} \times DL^2 \tag{2}$$

$$E = 2 \times DL - 0.59 \times 10^{-3} ME^2 \times DL \tag{3}$$

The main effect plots for the 6 effort drivers are shown in Fig. 1.

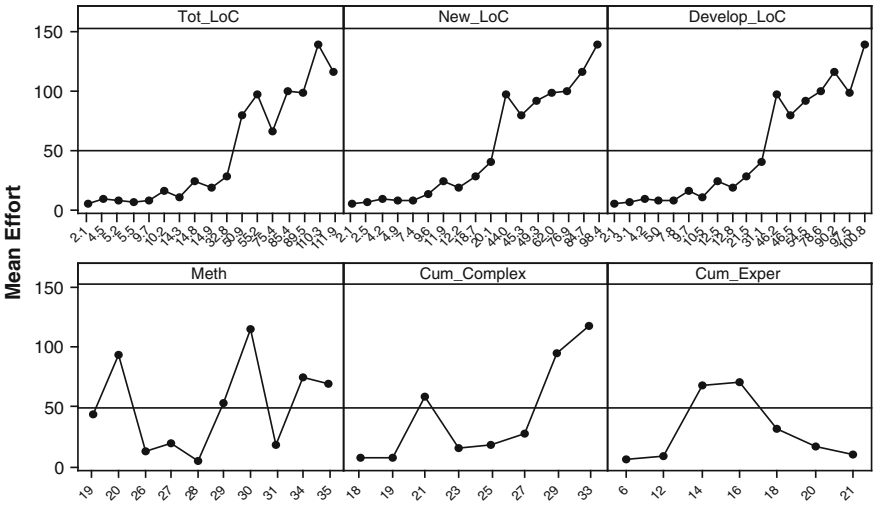

Fig. 1 Main effects plot

4 Neural Network Modeling

Back-propagation (BP) NN modeling for effort estimation has been carried out in this work using the MATLAB (2007b) NN toolbox options. Initially, a simple two-layer BP (6-6-1) NN was employed. The number of hidden nodes in the hidden layer was kept equal to the number of inputs (6 here). The number of hidden neurons was then suitably increased in an orderly hit and trial manner, to decide the final structure of the NN by keeping a check on the convergence rate of training, testing, and validation errors as well as the average percentage error. The learning rate and momentum can also be adjusted for the above purpose (although not varied in the present work).

Before the network is made ready to make estimates, we input the combinations of data inputs and outputs [18] through the network for training (60 %), validation (20 %), and testing (20 %). In our case, the activation functions of both the hidden and output layers were initially chosen to be tan-sigmoid. The same was later changed to the purelin(ear) function in the output layer. We used the two most popular training algorithms i.e., the Levenberg-Marquardt (LM) and the Bayesian regularization (BR) algorithms. The training performance and linear regression analysis (between the network outputs and the corresponding targets) are shown in Figs. 2 and 3. For the LM algorithm, the output tracks the targets reasonably well, and the regression coefficient (R) value is over 0.97 mostly. Similarly, for the BR algorithm-based training with purelin output function, the R values are over 0.99 in nearly all the cases (Fig. 3).

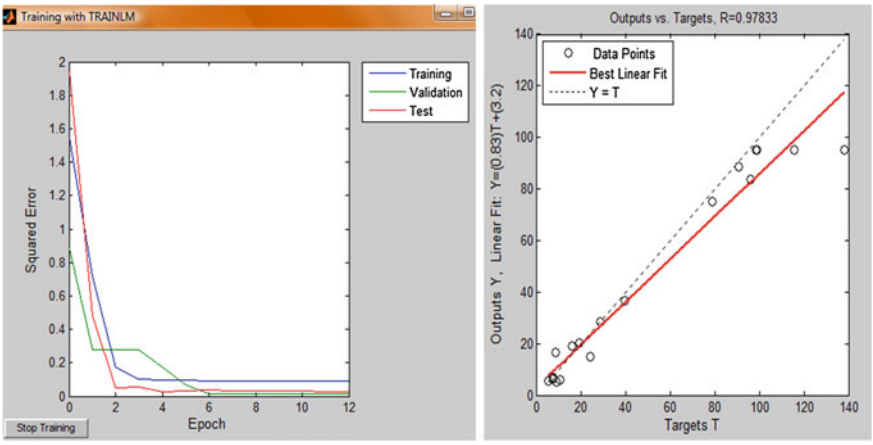

Fig. 2 Levenberg-Marquardt training with tansigmoid function in output layer

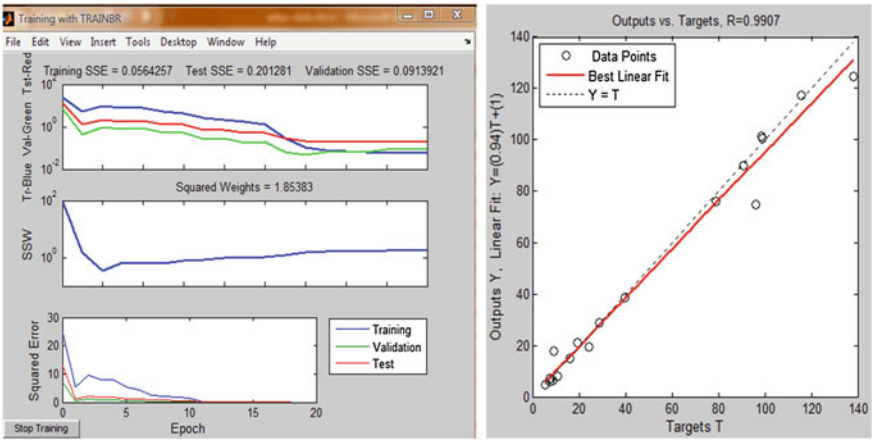

Fig. 3 Bayesian regularization training with purelin(ear) function in output layer

4.1 NN Modeling Tips

Listed underneath are some practical tips for efficient NN modeling.

- NNs are rather sensitive to the number of neurons in the hidden layers. Too few neurons often lead to underfitting, while too many neurons can contribute to overfitting. In this case, inspite of all the training points being well fitted, the fitted curve oscillates largely between these points [20].
- The NN dataset is generally divided in the following ratios: training (50–60), validation (20–25), and testing (20–25).
- Learning rate (alpha) represents how quickly an NN learns ranges from 0 to 1 and is initialized randomly. As with linear networks, a learning rate that is too large leads to unstable learning. Contrarily, a too small learning rate results in much longer training times. Typical values are 0.01–0.05.
- Momentum is a variable, which helps NN to break out of local minima. It may range from 0 to 1. Typical values are around 0.5.
- The threshold function (logsig, tansig, purelin, etc) selection is critical and determines when a node fires propagating a value further through the network. The choice is essentially based on the range and sign of inputs/outputs.
- The BR algorithm which is a modification of the LM algorithm is often used, as it generalizes well and reduces the difficulty of determining the optimum network architecture.
- LM training would normally be used for small- and medium-size networks, if enough memory is available. If memory is a problem, then there are a variety of other fast algorithms available. For large networks, one would probably want to use trainscg (conjugate gradient) or trainrp (resilient BP) algorithms.

- Overfitting is one of the most common problems that occurs in NN training. The training set error becomes a very small value, but the error turns to a large value when new data are presented to the network. An attempt at collecting more data and increasing the size of the training set must be made to prevent the situation of overfitting [20].
- One suggested method to improve network generalization is the use of a just large enough network that provides an adequate fit. Larger is the network used, more complex can be the functions the network can create. A small enough network will not have enough power to overfit the data. The two methods for improving generalization and implemented in MATLAB NN toolbox are regularization and early stopping [20].

5 Results and Discussion

The degree to which a model's estimated effort (MM_{est}) matches the actual or target effort (MM_{act}) is estimated by a percentage relative error. Magnitude of relative error (MRE), which accounts for under and overestimates along with its mean magnitude of relative error (MMRE) is often used in effort estimation analysis.

$$MRE = \left| \frac{MM_{act} - MM_{est}}{MM_{act}} \right| \tag{4}$$

Table 2 (in brief) presents a comparison of the empirical models (Eqs. 1–3) fitted effort and NN effort (for different configurations) with the target effort of [19]. It can be concluded that the present NN framework is able to successfully model the dataset with nearly the following percentage relative error and percentage mean relative error:

1. −10.58 to 8.36 and 4.68 %, respectively, for trainlm with tansig function in output layer and hidden neurons varied from 6 to 20.
2. −12.5 to −9.62 and −10.3 %, respectively, for trainbr with tansig function in output layer and hidden neurons varied from 6 to 20.
3. 0.65 to −3.12 and 0.79 %, respectively, for trainbr with purelin function in output layer and hidden neurons varied from 6 to 20.

The relative error obtained from the developed multi-regression model (Eq. 1) is comparable to other models (Eqs. 2 and 3). A comparison between the mean relative error of the developed NN and regression models and the MMRE of Halstead, Walston-Felix, Bailey-Basili, and Doty models are shown in Table 3 [16].

Table 2 Comparison of empirical model fitted and NN effort with target effort

Project number	Target effort [18]	Predicted effort (Eq. 1)	Percentage error = (1-C3/C2) *100	Predicted effort (Eq. 2)	% error = (1-C5/C2) *100	Percentage error (Predicted effort from Eq. 3)	Percentage error NN output		
							LM	BR tansig	purelin
1	115.8	127.28	−9.91	121.64	−5.05	−14.42	17.8	19.2	−1.1
—									
6	98.4	98.63	−0.23	128.24	−30.33	−49.01	9.0	−54.2	−2.8
—									
18	138.3	133.89	3.19	131.07	5.23	3.94	31.2	31.9	10.0
		Percentage mean error	−5.41		−8.66	−2.76	4.68	−10.3	0.79

Table 3 Comparison of different models

Model name	Model equation	MMRE
Halstead	$E = 5.2(\mathrm{DL})1.50$	0.1479
Walston-Felix	$E = 0.7(\mathrm{DL})0.91$	0.0822
Bailey-Basili	$E = 5.5 + 0.73(\mathrm{DL})1.16$	0.0095
Doty (for DL > 9)	$E = 5.288(\mathrm{DL})1.0$	0.1848
		Mean relative error
Present work—(1) LM-based BPNN	–	0.0468
(2) BR-based BPNN – 1		−0.1030
(3) BR-based BPNN – 2		0.0079
(4) Multilinear regression	Eq. 1	−0.0541

6 Conclusions

Effort estimation is a complex task, and research studies indicate that results in general vary a lot. The market potential for SD and maintenance is huge and constantly growing mainly for financial and online applications. In this work, a twofold approach based on NN and multilinear regression has been carried out for more accurate SD effort estimation

Acknowledgments The authors would like to acknowledge the financial support extended by the Faulty of Engineering and Built Environment, University of Johannesburg.

References

1. Shukla, R. Misra, A.K.: Estimating software maintenance effort - a neural network approach. 1st India, Software Engineering Conference. 107–112 (2008).
2. Jorgensen, M.: A review of studies on expert estimation of software development effort. J Syst. Software. **70**(1–2), 37–60 (2004)
3. Jorgensen, M., Shepperd, M.: A systematic review of software development cost estimation studies. IEEE T. Software Eng. **33**(1), 33–53 (2007)
4. Verner, J.M., Evanco, W.M., Cerpa, N.: State of the practice: an exploratory analysis of schedule estimation and software project success prediction. Inform. Software Tech. **49**(2), 181–193 (2007)
5. Jorgensen, M., Ostvold, K.M.: Reasons for software effort estimation error: impact of respondent role, information collection approach, and data analysis method. IEEE T. Software Eng. **30**(12), 993–1007 (2004)
6. Huang, X., Ho, D., Ren, J., Capretz, L.F.: A soft computing framework for software effort estimation. Soft Comput. **10**, 170–177 (2006)
7. Tronto, I.F.B., Silva, J.D.S., Anna, N.S.: An investigation of artificial neural networks based prediction systems in software project management. J Syst. Software. **81**(3), 356–367 (2008)
8. Kumar, K.V., Ravi, V., Carr, M., Kiran, N.R.: Software development cost estimation using wavelet neural networks. J Syst. Software. **81**(11), 1853–1860 (2008)
9. Sheta, A.: Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects. J. Comput. Sci. **2**(2), 118–123 (2006)

10. Xu, Z.W., Khoshgoftaar, T.M.: Identification of fuzzy models of software cost estimation. Fuzzy Set. Syst. **145**(1), 141–163 (2004)
11. Ahmed, M., Saliu, M.O., Alghamdi, J.: Adaptive fuzzy logic-based framework for software development effort prediction. Inform. Software Tech. **47**(1), 31–48 (2005)
12. Kazemifard, M., Zaeri, A., Ghasem-Aghaee, N., Nematbakhsh, M.A., Mardukhi, F.: Fuzzy emotional COCOMO II software cost estimation (FECSCE) using multi-agent systems. Appl. Soft Comput. **11**(2), 2260–2270 (2011)
13. Shukla, K.K.: Neuro-genetic prediction of software development effort. Inform. Software Tech. **42**, 701–713 (2000)
14. Huang, S.J., Chiu, N.H.: Optimization of analogy weights by genetic algorithm for software effort estimation. Inform. Software Tech. **48**, 1034–1045 (2006)
15. Sheta, A. F., Al-Afeef, A.: A GP effort estimation model utilizing line of code and methodology for NASA software projects. 10th International Conference on Intelligent Systems Design and Applications. 290–295 (2010).
16. Jorgensen, M.: Regression models of software development effort estimation accuracy and bias. Empir. Softw. Eng. **9**, 297–314 (2004)
17. Shukla, R., Misra, A.K.: Software maintenance effort estimation-neural network vs regression modeling approach. Int. J. Comput. Applic. **1**(29), 83–89 (2010)
18. Bailey, J.W., Basili, V.R.: A metamodel for software development resource expenditures. 5th IEEE International Conference on, Software Engineering. 107–116 (1981).
19. www.minitab.com (2012).
20. www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf (2012).