Scholars Bulletin ***(Computer Science)***
An Official Publication of "Scholars Middle East Publishers"
Dubai, United Arab Emirates
Website: www.saudijournals.com
ISSN 2412-9771 (Print)
ISSN 2412-897X (Online)

Software effort estimation using ANN technique

Siva Suryanarayana Ch[1*], Satya Prakash Singh[2]
[1]Department of Computer Science & Engineering, Birla Institute of Technology, Off-shore Campus, Ras Al Khaimah, UAE
[2]Department of Computer Science & Engineering, Birla Institute of Technology, Off- Campus, Noida, India

***Corresponding author**
Siva Suryanarayana Ch

Article History
Received: 28.04.2018
Accepted: 05.05.2018
Published: 30.05.2018

DOI:
10.36348/sb.2018.v04i05.003

Abstract: Software has frolicked an increasing significant role in methods acquisition, engineering, development, mainly for large complex systems. For such systems, accurate estimates of the software costs are a perilous part of effective software management. The preparation of forecasting the cost of the software has evolved, but it is far from perfect. Constructive cost model is the most widely used among all the models available. The proposed model focusses on data set obtained from 91 models nick named as "Maxwell". The projected model is tested and the test results from the neural network are matched with that of COCOMO model. From the experimental results, it is concluded that the amalgamation of the proposed model for the "Maxwell" data has proved better than the existing COCOMO and another model as obtained from literature.

Keywords: Software Cost Estimation (SCE), Software effort Estimation models, Artificial Neural Networks.

INTRODUCTION

SCE calculates the amount of effort and development time required to build software. It is one of the most precarious tasks and it helps the software industries to effectively manage their development process. In this paper, the use of neural network back propagation for SCE is proposed. The model is designed in such a manner that billets the COCOMO model and improves its performance.

It also increases the predictability of the SCE. The model is tested using 4 dataset-Cocomo81, Nasa93, China and Maxwell dataset. The test results are compared with that of the COCOMO model. From the experimental results, it was concluded that the integration of the conventional COCOMO model and the new approach improves the cost estimation accuracy and the estimated cost can be very close to the actual cost.

LITERATURE REVIEW

Software Effort Estimation

It is impractical to expect very precise effort estimations of software development because of the characteristic uncertainty in projects, and the composite and vibrant interaction of factors that impact development effort use. The important issue in the software effort estimation is the accuracy of resource estimation. The accurate estimation can help managers

There are two major types of cost estimation methods: *algorithmic* and *non-algorithmic.*

Algorithmic models contrast commonly in scientific sophistication. Some are based on simple arithmetic formulas using such summary statistics as means and standard deviations [1]. Others are based on regression models [2] and differential equations [3].

Non-algorithmic Methods

- Analogy costing: This technique requires one or more completed projects that are similar to the new project and derives the assessment through reasoning by analogy using the actual costs of previous projects. The forte of this method is that the estimate is based on actual project experience.
- Expert judgment: This technique involves consulting one or more experts. The experts offer estimates using their own approaches and experience.

- Delphi technique: A modification of the Delphi technique proposed by Boehm and Fahquhar [4] seems to be more effective: Before the estimation, a group meeting involving the coordinator and experts is arranged to discuss the estimation issues.
- Price-to-win: The SCE to be the best price to win the project. The estimation is based on the customer's budget instead of the software functionality.

Algorithmic methods

The algorithmic methods are based on mathematical models that produce cost estimate as a function of a number of variables, which are considered to be the major cost factors. Any algorithmic model has the form:

Effort = f(x1, x2, …, xn)

Where,

{x1, x2, …. xn} denote the cost factors. The existing algorithmic methods vary in two aspects: the selection of cost factors, and the form of the function f.

Cost factors

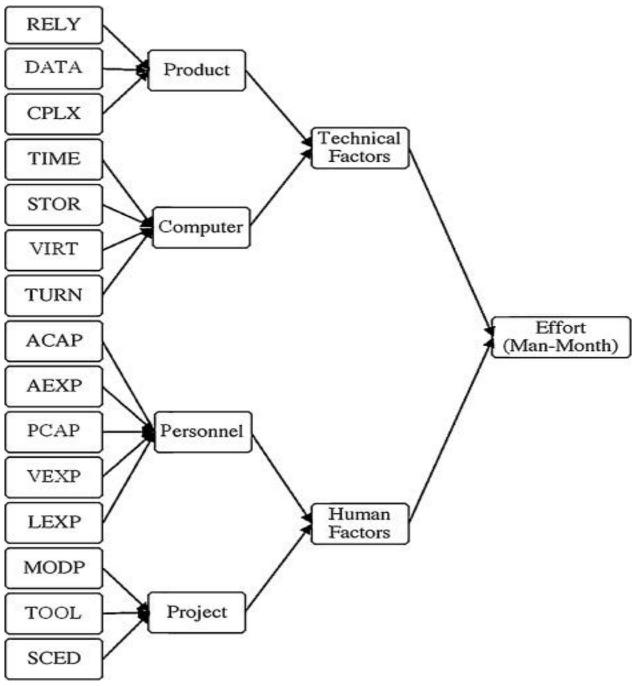
Besides the size, there are many other cost factors. The most widespread set of cost factors are proposed by Boehm in COCOMO II model [5]. These cost factors can be divided into four types:

Product factors: reliability; product complexity; database size used; required reusability; documentation match to life-cycle needs;

Computer factors: implementation time constraint; main storage constraint; computer dispatch constraints; platform volatility;

Personnel factors: forecaster capability; application experience; programming capability; platform experience; language and tool experience; personnel continuity.

Project factors: multisite development; use of software tool; required development schedule. The above factors are not necessarily independent, and most of them are hard to quantify. In many models, some of the factors appear in combined form and some are simply ignored. Also, some factors take discrete values, resulting in an estimation function with a piece-wise form.



Accurate estimation of the size is important to suitable estimation of effort, time duration, and cost of software. The Project size is a measure of the problem complexity in terms of effort and time required to develop the product.

New approach

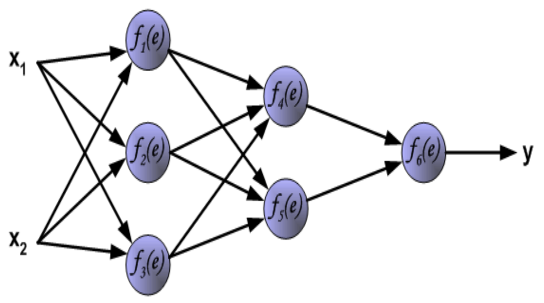
Most of the SCE models are constructed on using data from projects of a particular organization. Using such data has well known benefits such as ease of understanding and controlling of collected data. But different researchers have recounted inconsistent results using different SCE techniques. It is recounted that it is still difficult to generalize many of the obtained results [6]. This is due to the characteristics of the data sets being used and data sets being small. Alternative among the best popular models is the neural networks. These are non-linear modeling techniques inspired by the functioning of the human brain. Observations which are the most commonly used type of neural networks that are based upon a network of neurons agreed in an input layer, one or more of the hidden layers and an output layer in a strictly feed forwarded manner. Each neuron processes its inputs and generates one output value via a transmission function, which is transferred to the neurons in the subsequent layer. The output of the hidden neuron i is computed by processing the weighted inputs and its bias term b_i as follows:

H_i = f(b_i + summation (j=1 to n W_{ij} x_j)

W_{ij} denotes the weight connecting input I to hidden unit j [7].

Neural networks have learning ability and are good at modeling complex non – linear relationships. In addition it also provides more flexibility to integrate expert knowledge into the model. The architecture of neural networks which is used in SCE is the back propagation trained feed forward networks.

Three layer neural network with two inputs and one output, which is shown in the picture below, is used:



In this research, we present a model to estimate the software effort (person-month) based on back propagation neural network built from fifteen COCOMO components and the software size.

PROPOSED MODEL

Neural network performance depends on its architecture and their parameters. There are many parameters prevailing the architecture of the neural network counting the number of layers, the number of nodes in each layer, the transfer function in each node, learning algorithm parameters and the weights which determine the connectivity between nodes. There is no rule which regulates the ultimate parameter settings, even a slight parameter changes can cause key variations in the behaviour of almost all networks [8]. This property of the neural network is took in this work

The model presented in this paper is inspired by Ch. Satyananda Reddy and KVSVN Raju [8]. It reconnoitres the influence of variations of activation functions on SCE. It lodges the COCOMO II stake architecture model given by "(1)". The proposed model uses the uniqueness function at the input layer which is defined by f(x)=x It uses unipolar sigmoid function defined by f(x)=(1/(1+(e^-x))) at the hidden and the output layer respectively.

In this to estimate person months in "(1)" requires 24 input nodes in the input layer which corresponds to 17 EM's, 5 SF's and 2 bias values. The COCOMO model which is a nonlinear model is transformed into a linear model using natural logarithms as shown in "(2)".

ln(PM) =
$\ln(A) + \sum_{i=1}^{i=17} \ln(EM_i)$+[1.01+SF1+..+SF5]*ln(size) (2)

The above equation becomes:

CPM = [b1+x1*z1+x2*z2+……+x17*z17] + [b2+z18+……..+z22]*[yi+ln(size)] (3)

where , CPM=ln (PM);

z1=ln(EM1); z2=ln(EM2) ;……; z17=ln(EM17);

z18= SF1 ;……..;

z22= SF5;

b1 and b2 are the biases and the coefficients xi and yi are weights from the input layer to the hidden layer. The COCOMO II model as given by "(3)" is used in our proposed work and is shown in "Fig.1". This network consists of two hidden layer nodes CEM and CSF that take into account the contribution of effort multipliers and scale factors. CPM is the node of the output layer where we get the value of ln(PM) which is the desired output of the model. In the above network all the original EMi and SFi values of COCOMO II are pre-processed to ln(EMi) and ln(SFi) and used as input nodes. The two bias values are denoted by b1 and b2, which are ln(A) and 1.01 respectively. The size of the product is not considered as one of the inputs to the network but as a cofactor for the initial weights for scale factors (SF). The weights associated to the input nodes connected to the hidden layer are denoted by xi for 1<=i<=17for each input ln(EMi) and b1. On the other hand, the weights associated to the hidden layer for each ln (SFi) input nodes and b2 are yi+ln(size) for 18<=i<=22. These weights are initialized as xi=1and yi=0. The weights from the hidden layer to the output layer are denoted by p and q and initialized as p=q=1. As mentioned before, the activation function in the input layer is the identity function. The activation function in the hidden layer and the output layer is the sigmoidal function.

TRAINING ALGORITHM

The feed forward back propagation method is used to train the network by iteratively handling a set of training samples and associating the network's estimate with the actual value. For each training sample, the weights are modified so as to minimize the error between the networks expected value and the actual value. The following algorithm is used for training the proposed network and for calculating the new set of weights:

Step1: weights and learning rate α ($0< \alpha<=1$) are Initialized
Step2: Implement steps 3-10 when stopping condition is false.
Step3: Execute steps 4-9 for each training pair.
Step4: Each input component receives input signal and sends it to the hidden component.
Step5: Each hidden component CEM and CSF sums its weighted input signals to calculate net input given by:
$CEM = b1+ z_i*x_i$ for i=1 to 17
$CSF = b2+ z_i*(y_i + \ln(size))$ for i=18 to 22

Apply sigmoidal activation function over CEM and CSF and send the output signal from the hidden component to the input of output layer components.

Step6: The output component CPM, calculates the net input given by:
$CPM =CEM*p+CSF*q$

Apply sigmoidal activation function over CPM to compute the output signal E_{est}.

Step7: Calculate the error correction term as:
$\delta=E_{act} -E_{est}$, where E_{act} is the actual effort from the dataset and E_{est} is the estimated effort from step6.

Step 8: Update the weights between hidden and the output layer as:
$p(new)=p(old)+ \alpha* \delta* CEM$
$q(new)=q(old)+ \alpha* \delta* CSF$
Step 9: Update the weights and bias between input and hidden layers as:
$xi(new)=xi(old)+ \alpha* \delta EM*zi$ for i=1 to 17
$yi(new)=yi(old)+ \alpha* \delta SF*zi$ for i=18 to 22
$b1(new)=b1(old)+ \alpha* \delta EM$
$b2(new)=b2(old)+ \alpha* \delta SF$
The error is calculated as
$\delta EM= \delta*p; \delta SF= \delta*q$;

EVALUATION AND RESULTS

The experimented with the proposed model by taking the original projects with "Maxwell data" 91 data sets. The evaluation contains in comparing the accuracy of the estimated effort with the actual effort. The estimate accuracy is measured based on standard metrics such as Magnitude of relative error (MRE). MRE is defined as

RE (Relative Error)=(Estimated PM- Actual PM)/(Actual PM) (1)

MRE (Magnitude of Relative Error) = |Estimated PM-Actual PM |/ (Actual PM) (2)

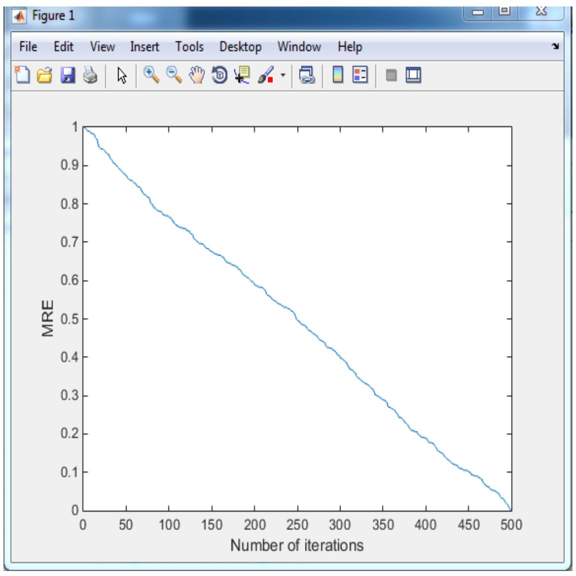


Fig-1: Comparison of the three models used in the study

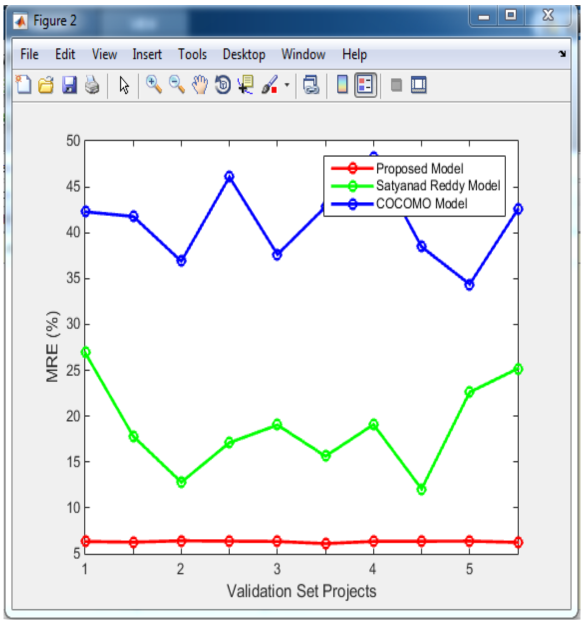


Fig-2: Graph of error correlation with number of iterations.

If the value of MRE is large, then the model over-estimates the cost, while a larger negative value would indicate, that the model under-estimates the software cost.

In evaluating the proposed model, the MRE values are calculated using the equation (1). These values are then compared using the COCOMO model as proposed by [9] and as compared with [10].

Table-1: Result and comparison MRE

S No	COCOMO model	Satyananda Reddy Model	Proposed Model
1	39.24	20.08	4.218
2	43.31	11.21	4.215
3	32.22	30.03	4.213
4	40.22	30.10	4.210

MMRE (Mean Magnitude of Relative Error) = Σ MRE/N (3)

Table-2: Result and comparison MMRE

COCOMO model	Satyananda Reddy Model	Proposed Model
41.10	21.11	4.22

The figure-1 and 2 is the graphical representation of MRE for the three models. MRE values were plotted for each of the project in the validation set. It is observed that there is a decrement of relative error using the proposed model. Table-1 obviously depicts that the proposed model has shown lower values when compared to the models as proposed by the counterparts. The results thus obtained suggest that the proposed architecture can be applied for accurately predicting the software costs.

CONCLUSIONS AND FUTURE WORK

Software development gradually costly to develop and is a major cost factor in any information system budget. The accuracy of assessment of software project cost has direct and important effect on the quality of the firm's decisions. Organization sensibly considers the costs and benefits of the software before promising the required resources to that bidding for a contract.

This paper uses resilient back propagation method of neural network which records the Maxwell data. This model includes the data set and is subsequently used to train the network. Based on the tests performed, it is observed that the proposed model outscored the existing two models as discussed earlier.

Future research can confirm this estimation with other member functions or hybrid with new functions for software cost estimation.

REFERENCES

1. Tausworthe, R. C. (1981). Deep Space Network Estimation Model. *Jet Propulsion Report*, *817*.
2. Boehm, B. W. (1981). *Software engineering economics* (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.
3. Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, *68*(9), 1060-1076.
4. Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches—A survey. *Annals of software engineering*, *10*(1-4), 177-205.
5. Deshpande, M. V., & Bhirud, S. G. (2010). Analysis of combining software estimation techniques. *International Journal of Computer Applications (0975–8887)*.
6. Rao, B. T., Sameet, B., Swathi, G. K., Gupta, K. V., RaviTeja, C., & Sumana, S. (2009). A novel neural network approach for software cost estimation using Functional Link Artificial Neural Network (FLANN). *International Journal of Computer Science and Network Security*, *9*(6), 126-131.
7. Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Using neural networks in reliability prediction. *IEEE Software*, *9*(4), 53-59.
8. http://promise.site.uottawa.ca/SERepository.
9. Khoshgoftaar, T. M., Allen, E. B., & Xu, Z. (2000). Predicting testability of program modules using a neural network. In *Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on* (pp. 57-62). IEEE.
10. Kaushik, A., Soni, A. K., & Soni, R. (2012, November). An adaptive learning approach to software cost estimation. In *Computing and Communication Systems (NCCCS), 2012 National Conference on* IEEE. (pp. 1-6).