Design of a Soft Computing Model for Cost Estimation

[1]Ravneet Preet Singh Bedi, [2] Amardeep Singh
[1]Ph.D.Research Scholar, [2]Professor
[1,2]Department of Computer Science and Engineering,
Punjabi University, Patiala – 147002, Punjab, India

ABSTRACT

Cost estimation of a software development is a significant action taken in the early phases of software design. The input datasets are mainly engaged from the promise source. Data mining and soft computing techniques are used to mark the software program development cost estimation. In this paper, two data mining techniques named Liner Regression and Multi-Layer Perceptron are used to analyze the software estimation. Results are analyzed using weka tool and various performance parameters are analyzed such as correlation coefficient, mean absolute error, root mean squared error, relative absolute error and root relative absolute error. Experimental results are shown in terms of graphs.

KEYWORDS

Soft Computing, PROMISE, MLP, LRC.

I. INTRODUCTION

Software growth is the development of a software product. The keyword "software development" can be referred to the action of computer software design, which is the procedure of writing and maintenance of the source code, but in a wider sense of the time, it comprises all that is involved between the commencement of the expected software through to the final appearance of the software, ideally in a strategic and organized way [1]. The requirement for improved quality control of the software growth development has given rise to the discipline of software engineering, which aims to apply the orderly approach demonstrated in the manufacturing example to the procedure of software development.

Software effort estimation is an essential component of software progress. As the software increases in size and complexity the software effort approximation job becomes complex in order to deal with the complexity occurring subsequently from the last few ages. Many researchers, all over the world try to advance new demonstrating techniques, which could deal with the varying complexity and increased size of software. The skilled approximation is the

dominant strategy when estimating software development effort. Creation of prescribed software effort estimation models has been the main emphasis in this area of research. The primary replicas were typically based on regression examination or mathematically derived philosophies from other areas. Subsequently many other methods [2] have been advanced to raise the productivity of the methods. Refining the exactness of the cost estimation models principals to actual controller of period and efficient throughout software growth.

In this paper, we have presented a few of the new methods which are used for software effort estimation. Soft computing is a series of methods and approaches which contract with real real-world circumstances in the same way as individuals deal with them, on the basis of intellect, common sense, reflection of similarities, approaches, etc. in other words, soft computing is a family of problem-resolution methods headed by approximate reasoning and functional and optimization approximation methods, including search methods [3]. Soft computing is therefore the theoretic basis for the area of intellectual systems and it is manifest that the change between the area of artificial intellect and that of smart systems is that the first is founded on hard figuring and the second on soft computing. Soft Computing is still growing and developing.

II. LITERATURE SURVEY

Bishop C.M. [4] proposes linear regression classifiers to be used for a variety of pattern recognition applications. This technique is also very much suitable for classification of dataset related to software engineering problems.

Heiat A. [5] shows that a technique called multilayer perceptron neural network and regression models can be a promising technique for estimation of software efforts. This technique has a tested drawback that it only works well if the third and fourth generation languages dataset are used. Otherwise, the technique shows not so promising results.

Braga P. L. et al. [6] device a bagging classifier-based technique to estimate software development efforts and are able to show promising results. The result show improvements upon the techniques using linear regression and multilayer perceptron.

Srinivasan et al. [7] uses the decision tree method and produce excellent results by managing the inaccuracy in inputs very well. Also, their quality is sensitive to the data on which they are trained. The research cautions the users to study the model sensitivity before application.

Du et al. [8] combine neuro-fuzzy model with SEER-SEM technique to achieve an accuracy that is claimed to be 18% higher than its counterparts.

III. PROPOSED TECHNIQUE

Linear Regression: It works by estimating coefficients for a line or hyperplane that best fits the training data. It is a very simple regression algorithm, fast to train and can have great performance if the output variable for your data is a linear combination of your inputs. The performance of linear regression can be reduced if your training data has input attributes that are highly correlated. Weka can detect and remove highly correlated input attributes automatically by setting eliminate Collinear Attributes to True.

Straight-line regression examination includes a reaction variable i.e. gy, and include solitary indicator variable i.e. gx. It is the least complex type of relapse, and models' gy as a straight capacity of gx. That is,

gy=b + w_{gx}+ …… Eq. (1)

where the difference of y is thought to be consistent, and b and w are relapse coefficients indicating the Y-catch and slant of the line, individually. The relapse coefficients, w and b, can likewise be thought of as weights, with the goal that we can identically compose,

gy=w_0+$w1_{gx}$+. ……Eq. (2)

These coefficients can be explained for by the strategy for minimum squares, which appraises the best-fitting straight line as the one that limits the mistake between the genuine information and the gauge of the line. Give D a chance to be a preparation set comprising of estimations of indicator variable, gx, for some populace and their related esteems for reaction variable, y. The preparation set contains |D| information purposes of the frame (gx_1, gy_1) ,(gx_2, gy_2),……($gx_{|D|}, gy_{|D|}$)).

The regression coefficients may be envisioned the usage of this technique with the following equations:

$$w1 = \frac{\sum_{i=1}^{|D|}(gx_i - g\bar{x})\ (gy_i - \bar{gy})}{\sum_{i=1}^{|D|}\ (gx_i - g\bar{x})^2} \qquad \text{..... Eq. (3)}$$

Multilayer Perceptron

Artificial neural networks are the part of artificial intelligence and MLP is a type of neural network. Multi-Layer Perceptron (MLP) is a feed-forward network that maps the organization of inputs to their corresponding outputs. Figure 1 demonstrates a feed-forward multi-layer perceptron neural community [9]. MLP is made of easy neurons termed as a perceptron. Neural community generates records by way of allowing input perceptron's consisting the values labelled on them. The activation function of neurons is calculated through the components referred to under in the output layer:

$$a_i = \sigma(\sum_j W_{ij}\, O_j) \qquad \text{..... Eq. (4)}$$

Where a_i represent level of activation for ith neurons; j is the set of neurons of the preceding layer; W_{ij} is the load of the connection between neurons i and j; O_j represents the output of jth neuron and $\sigma(x)$ is the transfer function.

$$\sigma(x) = \frac{1}{1+e^{x}} \qquad \text{..... Eq. (5)}$$

A back-propagation strategy based on delta rule is used to train multilayer perceptron network. MLP consist of various neurons divided into various layers, as follows:

- **Input Layer:** This layer generates the input for the network. The number of neurons depends upon the number of input given to the network.
- **Hidden Layer:** The layer that maps the input to the corresponding output is named as a hidden layer. Hidden layers vary in number.
- **Output Layer:** The layer from where the resultant can be seen. The number of neurons in output layer depends upon the learning of the kind of problem.

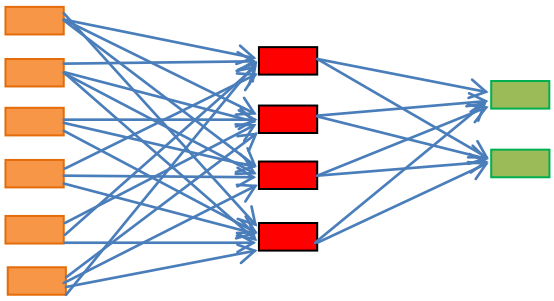

Figure 1: Multi-Layer Neural Network Architecture

The type of relationship between the input and output vectors in MLP is a non-linear relationship. This is done by interconnecting the neurons in the antecedent and succeeding layers. Outputs are achieved by multiplying them with weight coefficients. In the training phase, the neural network is given the information regarding training only. Later on, the weights of the network are tuned between [-0.5, 0.5] to minimize the error rate between the expected and observed outputs and to enhance the frequency of training to a predicted level. A sequence untrained inputs are applied to the input to formalize the training. These input set are different from the inputs that are used for the training of the network. Training of the neural network is highly complex due to a large number of variables. MLP holds lots of advantages over other algorithms even if a correct relationship is not induced between the input and output or if the essential and exact information is not achieved. Non-Linear Activation Function of MLP makes MLP different from other networks.

Let a dataset D, consist of training samples and their target values, L be the rate of learning by the network to generate a trained network:

1. Initialize the weights and the biases of the layers using small random values.
2. Compute the weighted sum of the inputs, where

$$O_j = I_j \quad \text{..... Eq. (6)}$$

The output of the inputs is the true input values.

3. Compute the activation functions of hidden layers, where

$$I_j = \sum_j W_{ij} O_j + \theta_j \quad \ldots\ldots\ldots\ldots\ldots \text{Eq. (7)}$$

Compute the net input of j with respect to i (previous layer).

4. Compute the output of the layers, where

$$O_j = \frac{1}{1+e_j^{-1}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{Eq. (8)}$$

5. Compute the error rate by Back-Propagation, Error for output layer:

$$Error_j = O_j(1 - O_j)(T_j - O_j) \quad \ldots\ldots\ldots\ldots \text{Eq. (9)}$$

Error calculation of next hidden layer, h:

$$Error_j = O_j(1 - O_j) \sum_h Error_h w_{jh} \quad \ldots\ldots\ldots \text{Eq. (10)}$$

Weight update:

$$w_{ij} = w_{ij} + \Delta w_{ij} \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq. (11)}$$

Bias update:

$$\theta_j = \theta_j + \Delta\theta_j \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq. (12)}$$

Where Δw_{ij} and $\Delta\theta_j$ are the change in weight and bias

IV. EXPERIMENTAL RESULTS AND ANALYSIS

Performance Evaluation Criteria

- **Correlation Coefficient:** Two factors correlation coefficient in an informational index equivalents to their covariance independent by the outcome of their individual
- **Mean Absolute Error:** MAE measures the common significance of the errors in a set of predictions, without thinking about their direction. It's the common over the check pattern

of the absolute differences among prediction and actual commentary where all individual differences have equal weight.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \ldots\ldots\ldots\ldots \text{Eq (13)}$$

- **Root mean squared error (RMSE)**: RMSE is a quadratic scoring decide that likewise measures the normal extent of the blunder. It's the square foundation of the normal of squared contrasts amongst forecast and genuine perception.

$$MAE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2} \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq (4)}$$

- **Relative Absolute Error**: The relative absolute error takes the aggregate total mistake and standardizes it by separating by the aggregate total mistake of the straightforward indicator.

  Scientifically, the relative total blunder E_i of an individual program i is assessed by the condition:

$$E_i = \frac{\sum_{j=1}^{n}|P_{ij} - T|}{\sum_{j=1}^{n}|T_j - \bar{T}|} \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq (14)}$$

  Where

  $P_{(ij)}$: Value predicted by the individual program *i* for sample case *j*

  T_j : Target value for sample case *j*; and $\overline{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n}\sum_{j=1}^{n} T_j \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq (15)}$$

- **Root Relative Squared Error:** The relative squared blunders takes the aggregate squared mistake and standardizes it by setting apart by the aggregate squared blunders of the straightforward indicator. Mathematically, the root relative squared errors E_i of an man or woman program *i* is evaluated with the aid of the equation

$$E_i = \sqrt{\frac{\sum_{j=1}^{n}(P_{ij}-T)^2}{\sum_{j=1}^{n}(T_j-T)^2}} \quad \text{..... Eq (16)}$$

Table 1: Classification Results Using Cross Validation Model Taking 10 Folds (Prediction Parameter - Months)

Name	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error	Relative absolute Error	Root Relative Square Error
Linear Regression	0.9182	3.7087	5.51	38.25%	42.39%
Multilayer Perceptron	0.8577	5.4907	7.16	56.63%	55.06%

From table 1, it is clearly shown that the results have been compared with the various classification method such as linear regression classifier, multilayer perceptron neural network classifier.

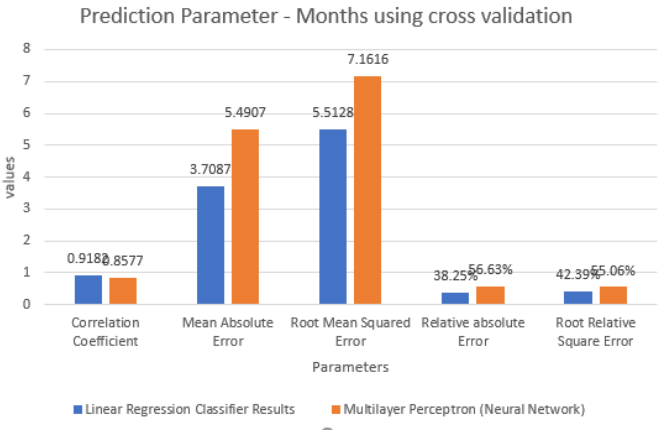

Figure 2: Graphical analysis of cross validation model (parameter – months).

The figure 1, shows the graphical comparison of various classification method such as Linear regression, MLP in term of mean absolute error, root mean squared error, correlation coefficient, relative absolute error and root relative square error.

Table 2: Classification Results Using Percentage Split Model Taking 70% Data as Training And 30% as Testing (Prediction Parameter - Months)

Name	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error	Relative absolute Error	Root Relative Square Error
Linear Regression	0.7601	10.3411	17.7903	125.00%	187.59%
Multilayer Perceptron	0.7922	5.6043	6.8429	67.74%	72.15%

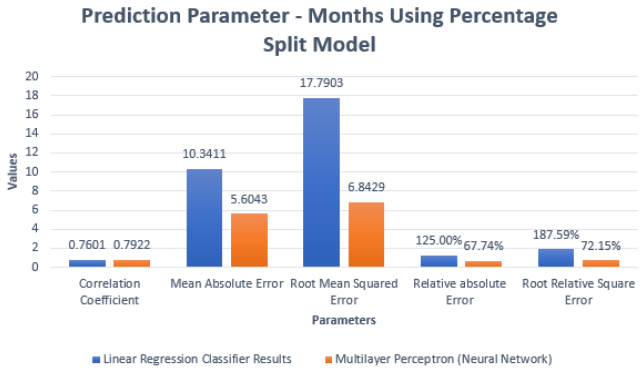

Figure 3: Graphical analysis of percentage split model (parameter – months).

The figure 3, shows the graphical analysis of percentage split with various classification method such as Linear regression, MLP Classifier in term of mean absolute error, root

mean squared error, correlation coefficient, relative absolute error and root relative square error.

Table 3: Classification Results Using Cross Validation Model Taking 10 Folds (Prediction Parameter Efforts x Months)

Name	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error	Relative absolute Error	Root Relative Square Error
Linear Regression	0.819	16851.4397	39263.1607	51.49%	57.05%
Multilayer Perceptron	0.9136	11800.2644	51076.9424	36.06%	74.21%

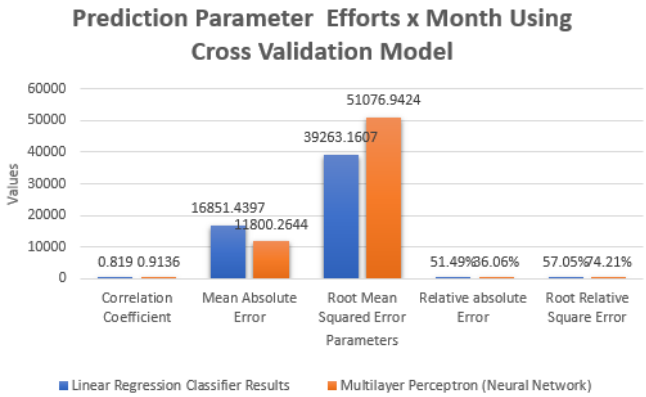

Figure 4: Graphical analysis of cross validation model (Prediction Parameter Efforts x Month).

The figure 4 shows the graphical analysis of cross validation model with various classification method such as Linear regression, MLP Classifier in term of mean absolute error, correlation coefficient, relative absolute error, root mean squared error, and root relative square error. The orange color illustrates multilayer perceptron and blue color illustrates the Linear Regression classifier results.

Table 4: Classification Results Using Percentage Split Model Taking 70% Data as Training And 30% as Testing (Prediction Parameter Efforts x Month)

Name	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error	Relative absolute Error	Root Relative Square Error
Linear Regression	0.4772	39716.8498	65752.4897	166.86%	262.00%
Multilayer Perceptron	0.9264	12667.6719	14761.218	53.22%	58.82%

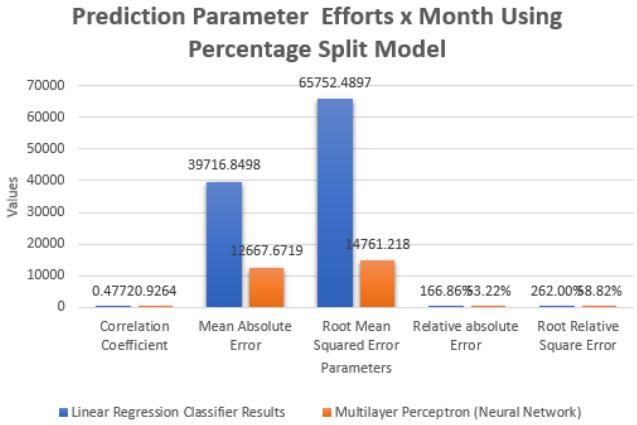

Figure 5: Graphical analysis of percentage split model (Prediction Parameter Efforts x Month).

The figure 5, shows the graphical analysis of percentage split model with various classification method such as Linear regression, MLP in term of correlation coefficient, root mean squared error, mean absolute error, and root relative square error relative absolute error.

V. CONCLUSION

Software Cost Estimation assumes a critical part in programming advancement. It includes estimating the effort and cost as far as cash to finish the product advancement. It is vital when lines of code for the specific project exceeds certain utmost, and when the product sent with an

excessive number of bugs and revealed necessities as the project goes incomplete. Error and accuracy computation in Software development projects is a thoroughly researched area in software engineering. Researchers have used a number of techniques to estimate software efforts errors and accuracy. Some of the existing techniques from the well-known data mining tool weka have studied to analyse the error. In this paper, Linear Regression Classifier (LRC) and Multilayer Perceptron (MLP) is analysed and compared on the basis of Months and Efforts * Months for the PROMISE Dataset.

REFERENCES

[1]. Boehm B., "Software Engineering Economics", Prentice-Hall, 1981

[2]. Boem B., Abts C., "Software Development Cost Estimation Approaches: A Survey", Ph.D thesis extension unpublished

[3]. Shepperd M., Schofield C., and Kitchenham B., "Effort Estimation using Analogy", In Proceedings of the 18th International Conference on Software Engineering, 1996, pp. 170-178, Berlin.

[4]. Bishop, C. M. (2006). Pattern recognition. Machine Learning, 128, 1-58.

[5]. Heiat, A. (2002). Comparison of artificial neural network and regression models for estimating software development effort. Information and software Technology, 44(15), 911-922.

[6]. Braga, P. L., Oliveira, A. L., Ribeiro, G. H., & Meira, S. R. (2007, August). Bagging predictors for estimation of software project effort. In Neural Networks, 2007. IJCNN 2007. International Joint Conference on (pp. 1595-1600). IEEE.

[7]. Srinivasan, K., & Fisher, D. (1995). Machine learning approaches to estimating software development effort. IEEE Transactions on Software Engineering, 21(2), 126-137.

[8]. Du, W. L., Capretz, L. F., Nassif, A. B., & Ho, D. (2015). A hybrid intelligent model for software cost estimation. arXiv preprint arXiv:1512.00306.

[9]. Prasad P., Sudha K., Rama S., and Ramesh S., "Software Effort Estimation using Radial Basis and Generalized Regression Neural Networks," Journal of Computing, vol. 2, no. 5, 2010.

[10]. Bhatnagar, R., and Ghose, M. K., (2012), "Comparing Soft Computing Techniques For Early Stage Software Development Effort Estimations," International Journal Of Software Engineering and Application, 3(2) pp. 119-127.