Received: February 28, 2019

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

An Advanced Mechanism for Software Size Estimation Using Combinational Artificial Intelligence

Varinder Kaur Attri[1]* **Jatinder Singh Bal[2]**

[1]*Department of Computer Science and Engineering,*
Punjab Technical University Kapurthala, India
[2]*Department of Computer Science and Engineering,*
Universal Institute of Engineering and Technology Mohali, India
* Corresponding author's Email: varinder2002@yahoo.com

Abstract: Software Size Estimation is the most essential and crucial calculations of the Software Development Life Cycle (SDLC) process. If done wisely, it may accumulate a large amount and time and if done badly, it may cost a lot of amounts. In this modern era of development, traditional methods of estimation do not stand a chance to provide output precisely. Constructive Cost Model (COCOMO)-2 is one of finest calculation methods for size and cost. This paper presents an automated early size estimation technique using Artificial Intelligence (AI). The categorization of the size estimation is into three parts, that is, UML diagrams/code folder mapping via COCOMO-2, COCOMO-2 metrics training, and the classification process to have an appropriate size. This paper focused on regression based training of AI which makes the estimation model more precise. For the evaluation, Mean Square Error (MSE) and Size estimation have been considered according to project samples. It has been seen that the proposed mechanism has attained a minimum MSE of 0.0115 and the difference in sizes is also not high. The comparison has also been done to depict the efficacy of the proposed work with A. B. Nassif et al. and S. Lohmor with B.B. Sagar for True positive rate and Mean square error.

Keywords: Early size estimation, Artificial intelligence, COCOMO-2, Mean square error, True positive rate.

1. Introduction

The Early Size Estimation (ESE) leads to the better success of a project1. A lot of previous studies have proved its significance in the success and failure of projects by ESE. Early estimation ensures less effort and low-risk factor. The accurate methodology of prediction generates sound estimations. Regression analysis leads to better efficiency in the size estimation. A detailed study is presented by A.B. Nassif [1]. The estimation of a project is a two-step process namely training and compilation or classification. The brief of two-step is given by Albrecht [2]. Function point analysis (FPA) is an Object Oriented Analysis (OOA) process which uses internal logical files and transactional functions for the estimation of the project size [3]. If the size is estimated in the early stage of the development, a lot of effort and time can be saved. An early estimation technique was presented proposed in 2013 using the log-linear model. The linear model is based on linear propagation model supported by [4]:

$$Z = ax + b \tag{1}$$

As depicted in Eq. (1), Z is the predicted output in terms of software size, a and b are subjective constants and × is the software component. The components of the software can be either Object Oriented Programming System (OOPS) based or regression based. The work done by Nassif was extended by Kocaguneli 2015 [5]. The author has also introduced a multilayer perception model using Artificial Neural Network (ANN). The concept of training and classification in size estimation was

also introduced in the model. Sarno has presented a comparison of different Neural Networks in cost prediction in 2015 [6]. COCOMO was utilized as a major component collector in this paper.COCOMO-2 was already introduced in 2000 by Dillibabu but it was just a case study [7]. Shalev-Shwartz has introduced Support Vector Machine (SVM) to reduce the training dependencies in 2008 [8]. This paper focuses on extending the possibilities of Neural Network by combining ANN (Artificial Neural Network) with SVM (Support Vector Machine). This research has utilized a hybridization of ANN and SVM as an artificial intelligence technique because in existing work several researchers proposed different classifiers for early size estimation but better performance has not been achieved. The existing classifiers have their own features and flow of an algorithm for the estimation of the software size is considered as time-consuming processes [9-10]. So, a hybrid classifier has been designed to minimize the time complexity of the system with the best true positive rate. The main advantages of the proposed work are given as:

- Software estimators understanding can be used to calculate a better-estimated cost.
- Based on the proposed research work, we get to know the very small dissimilarity between the preceding completed software projects and current software projects and this is also a very important way of discovering their real impacts.
- The proposed would need estimators for the discovery of attributes for the easy description of the software.

Some of the works which are used in this proposed architecture set are listed in the related work in section 2. The problem according to which the aims of the research are defined is in section 3. Later in section 4, the proposed architecture considering the dataset, the solution of the problem with the description is defined. The parameters that are taken for the implementation are also computed in this section and in section, the outcome is defined in the form of conclusion and scope of the work is also mentioned.

2. Related work

The researchers have put their effort in the software engineering area for the early estimation of software size. [A.B.Nassif and L. F. Capret in 2013] [1] have presented a multilayer perceptron model which classifies the utilization of the hidden layer. The proposed algorithm has used case point analysis or model for effort prediction. Although this research article was not directly for the execution of early size prediction, though, the concept of multilayer perception model helped us to understand the concept of multilayer perceptron model and that is why the proposed article has utilized multilayer neural network. [Emin Borandag et al. in 2016] [3] have computed the software project size via function point technique with Mark II FPA technique. Numbers of approaches are there for the estimation of software size considering as function point method like IFPUG FPA, MK IIFPA, and COSMUC FFP. The work has already being done for software projects. The obtained data have been compared with traditional approaches. As the researchers have not utilized the Artificial Intelligence techniques that could be used to obtain the desired results. [Ekrem Kocaguneli et al. in 2015] [5] has checked is it probable to discover the transfer learners for the effort estimation of software. The researchers have utilized data on 154 projects from two sources for the investigation of transfer learning among varied time intervals and 195 projects from 51 sources for the provision of evidence on value for existing cross-company learning issues of transfer learning. It has been discovered the similar transfer learning technique could be useful for transfer effort estimation outcome for the problem of cross-company learning and cross time learning issues. The research has lacked in transferring the instances between time intervals and domain intervals for the transfer of data in the data. The instance-based transfer is considered as the most challenging task to carry out the data from varied projects using diverse ontologies. [A.L.I. Oliveir. in 2006] [19] has presented a regression model utilizing SVM for component analysis. The research of this article has inspired users to use SVM. The presented regression model uses various kernels for the bifurcation of the component class and attributes. The research has utilized SVM as a classifier which is not that efficient as compared to neural network classification. [S.Lohmor and B.B.Sagar in 2017] [22] have proposed an effective method for the Software reliability growth model with hybridization of dolphin echolocation optimization artificial neural network via parallel computation. Dolphin echolocation optimization has been utilized for weight optimization with ANN structure for the reduction of computational complexity. The proposed mechanism has shown its effectiveness by contrasting the work with traditional approaches and the considered parameters are flexible and efficient but lack in MSE decrement rate. [Forrest W.

Crawford et al. in 2015] [25] has shown the method of utilizing network data being analyzed by RDS (Respondent-driven sampling) for the estimation of hidden population size. The researchers have used an effective Bayesian technique for the integration of missing edges for employed subgraph individuals. Validation of techniques is done by the simulated data and the techniques are applied for the estimation of a number of users who have taken drugs in Russia. Better techniques of learning could be utilized to capture the target population size.

3. Problem formulation

Software size estimation refers to the estimation of project size before it is completely deployed [11]. Software design and development with size estimation involves a range of practices with varying levels of formalities [12]. Some of the instances like test-driven development, formal methods, design patterns, and coding styles. The common goal of the proposed work is to produce high-quality software to sort out the problems of existing work. This process is done in order to attain optimal cost for the project and the total amount of expected cost so that the resources can be managed in an efficient manner because it is the most challenging factor of software estimation [13]. As technology is leading towards a new era of development, hence, traditional methods like computing the object-oriented metrics' and then identifying the project total size that will not be very effective here. There are several modern-day algorithms which can be opted in this contrast but the question is what will the selected architecture sustains for long and can handle complex architectures also? The problem of this research work looks into the matter of estimating size with advanced mechanisms and surfing through the complex architectures of codes and UML diagrams [14]. There are a lot of techniques for the software size estimation but due to some drawback, they cannot provide better efficiency in this filed. When there is a utilization of SVM, then there is an enhancement in the complexity because of its binary nature. If the only the neural network is used then the estimation time becomes more. So, this work has put an effort to minimize this problem by using a neural network with SVM [15]. The factors motivating to execute the research in the domain of artificial intelligence, metrics, and quality are:

- The complexity of the software is increasing day by day so the estimation of size is quite difficult.
- Human life is dependent upon the software and its quality. Therefore, in the case of large size software is the main concern and early estimation of software size is a big requirement of a human.
- To ensure software size in an early stage, software metrics are required.
- To tackle with the time complexity, artificial intelligence techniques are required with better prediction rate.

Hence, the researchers were motivated to pursue the research in the matter of metrics based hybridization of artificial intelligence techniques in the SDLC process.

4. Proposed architecture

4.1 Collection of Data

Dataset is a collection of UML diagrams which has been drawn from different code architecture of the JFree Chart. The Internet provides a lot of utility tools for the conversion of code frame into diagram frame. Gliffy is the best conversion utility tools which are available online. Code architectures have been changed here https://www.gliffy.com into UML diagrams [16].

4.2 Proposed solution

The proposed solution for the size estimation is divided into the following section:

i. Mapping of the UML diagrams / Code Folder through COCOMO-2
ii. Training of COCOMO-2 metrics for the further size estimation
iii. Classification for the appropriate size.

4.3 Mapping of UML Diagrams / Code Folder through COCOMO2

COCOMO-2 is the advanced software architectures for the cost, effort and size estimation and it provides early mapping through complex metric architecture. The following utility metrics fall under COCOMO-2 [17]. Table 1 shows varied COCOMO2 metrics and in graphical form is shown in Fig. 1.

Table 1.COCOMO2 metrics

RELY	CPLX	RUSE	STOR	TIME IN MS	LTEX
0.23	0.41	0.26	0.78	300	OO
0.26	0.56	0.28	0.87	323	OO
0.45	0.37	0.39	0.54	256	OO
0.74	0.12	0.59	0.41	263	OO

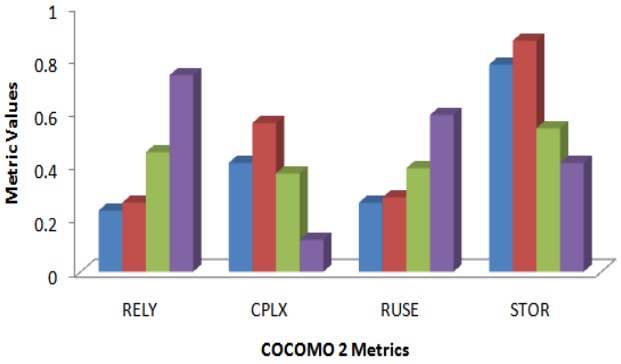

Figure.1 Metric comparison for different diagrams

RELY, CPLX, RUSE, and STOR is divided into three segments as detailed below:

$0 < x < 0.33$	$Average$	$Case\ 1$
$0.33 < x < 0.66$	$High$	$Case\ 2$
$x > 0.66$	$Very\ High$	$Case\ 3$

RELY (Product Reliability): It estimates the product reliability and is calculated using the output to the desired output ratio.

RUSE (The Reuse required): It computes the estimated project reusability according to the total number of classes used in the architecture. The mathematical expression is shown in Eq. (2).

$$RUSE = \frac{\sum_{i=1}^{n} Total\ class\ components}{\sum_{i=1}^{n} Total\ components\ in\ code} \quad (2)$$

As shown in Eq. (2), n is the total lines of codes.

STOR (Storage): It is the required space for the storage of every data used in the diagram or code. Mathematically, it can be demonstrated as Eq. (3):

$$STOR = \sum_{i=1}^{n} MemSpace(Alltypevariable) \quad (3)$$

As shown in Eq. (3), n is the Total line of codes and n is the Total lines of codes and $MemSpace$ is the total memory space of software.

4.4 TIME (Total time frame)

It may refer to the total time of completion of a code frame or total time frame for the creation of a project. It relies on the uniqueness of the code, the types of components getting used in the program. It also depends upon how many code frames has to be developed and how much code frame is already in stock. It also considers the frames which have to be borrowed or purchased through any third-party vendor. In terms of execution, it depends upon how many lines of codes have been written and what is the complexity of the code [18].

4.5 CPLX (Complexity)

It relies on the design of the frame and the level of experience of the developer. If the developer is experienced then the complexity of the code or architecture will be low as the developer will not attempt to write the entire code on his own but he would prefer to use the reusable components and class diagrams which make the complexity of architecture to a lower side [19].

4.6 LTEX (Language and Tools Experience)

It represents the type of programming style opted in most of the cases, these metrics are divided into the following categories [20].

- Nominal
- Average
- High
- Very High
- Results

A total of 100 diagrams including code diagrams have been used and the metrics have been evaluated, out of which, 4 is presented here [21]:

Before moving to the second part of the proposed methodology, the following data is also evaluated.

Estimated- Effort: It is the total effort which will be applied in order to attain the goal of development. The mathematical expression is depicted in Eq. (4).

$$E = \left\{ \frac{B + 0.01 \times Total\ (Diagram\ Components\ Estimated\ Effort)}{A \times ((Total\ diagram\ components)^{E}) \times Effort\ Multiplier} \right\} \quad (4)$$

Actual Lines of Code through COCOMO-2

$$KLOC = \{(\log(E - A) - \log(B))^{C}\} \quad (5)$$

As depicted in Eq. (5), A, B, and C are arbitrary constants used in the proposed work.

4.7 Training of COCOMO-2 Metrics for the Further Size Estimation

The second phase is to make the classification algorithm to understand the design of the desired output. The idea is to classify in order to get the presented diagram size and to compare with KLOC to justify how accurate the proposed architecture is [22].

Three types of training algorithm have been presented namely Support Vector Machine, FFBPNN (Feed Forward Back Propagation Neural Network) and a combination of SVM and NN. SVM

is a binary class classifier and if the classifier has to be utilized for multiple classes, it works with turn by turn concept.

Different in architecture from SVM, NN is a multiple class classifier but it has been seen often that it gets confused between two classes if they fall almost in the same range. To remove the vagueness of NN, a combination of SVM and NN is used as the main proposal of this research work. The architecture of training is represented in Fig. 2.

The presented architecture in Fig. 2 describes a hybrid structure of NN and SVM. NN has three layers in which the first layer contains the input vector l1 and l2 which goes into the intermediate layer of the NN. The classified vector is lx. If the classified vector has more than or equal to two classes then SVM would be called for the further binary classification.

4.8 Classification for appropriate size

The classification process requires a training mechanism which is mentioned in phase II of the proposed solution. This architecture takes the COCOMO 2 Cost drivers as the input and passes it to the training layer of the NN. As explained in Fig. 2 that the training for SVM is required only when the NN falls into any conflict between two classes [23].

Algorithm: Neuro-SVM for Size Estimation

Trained Neural=Function TrainNeuraln (Cost Drivers) // function for training using neural
1. Training_data=[ ]; // Initializing the training vector as empty
2. VecCount=0; // to increment in array
3. for each vec in Cost_Drivers/Metrics
4. Training_data(VecCount,1)=Vec; // Input data for training
5. VecCount=VecCount+1; // Increment in the array position
6. End For
7. Net= newff(Training_data, Group,20)// Initializing the Neural Network Training Architecture, Group will be its related Kloc & 20 is the no of the count of hidden neurons in the training architecture
8. Net.trainparams.epochs=50 // Neural Network is iterative and hence a total of 100 iterations is supplied. It is not compulsory that total running iteration equalizes to the total provided iterations. It completely depends upon the input data and the trajectory which the Neural Network performs in order to get trained.

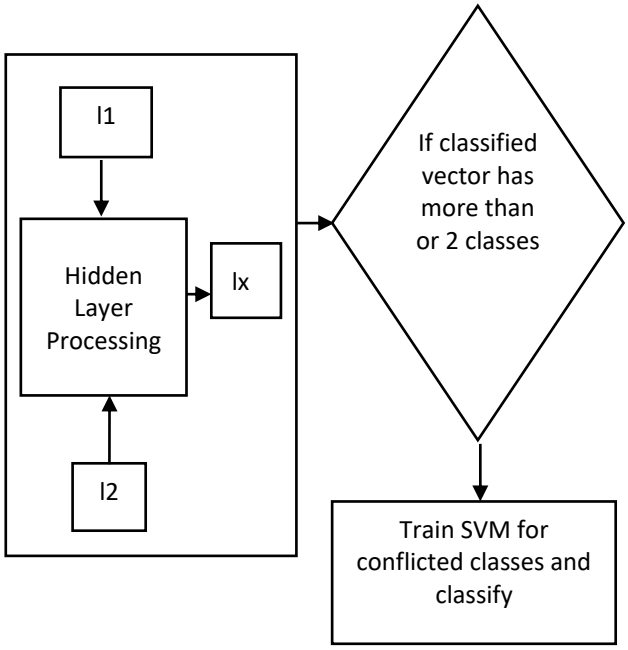

Figure.2 Proposed training and classification architecture

9. Net=train (net, Training_data); // Training of system using train command
10. End Function

The algorithms of Neuro-SVM are utilized for the training of the proposed system and return a trained structure of the system. The algorithm of SVM training is given below section. The above architecture produces the outcome represented in fig. 3 and 4.

4.9 Training of COCOMO2 metrics for the further size estimation

The training of NN is done in two phases that are defined below and are shown in Figs. 3, 4, 5 and 6 [24].

1. Mean Square Error (MSE) justification

$$MSE = \frac{\sum Outcome - \sum Input}{Length} of\ Outcome \qquad (6)$$

As shown in Eq. (6), $Outcome$ is the output of the neural network during the training and Input is input data of the neural network.

2. Regression analysis

It has been done in order to make sure each and every segment of data is understandable by the Neural Architecture.

The outcome of the regression analysis is represented in Figs. 5 and 6.

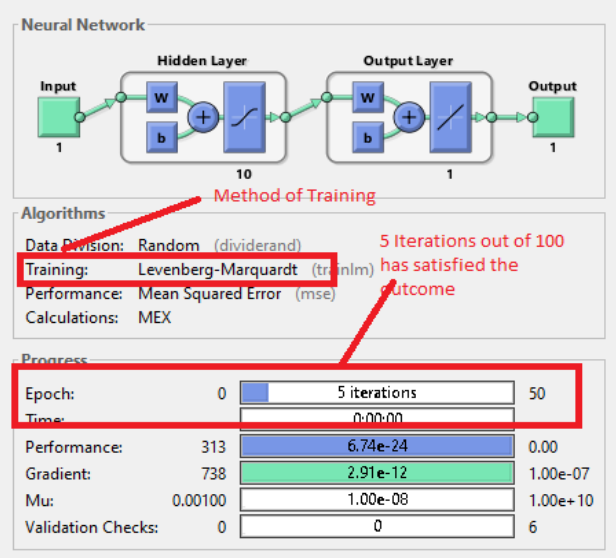

Figure.3 Training architecture of neural network

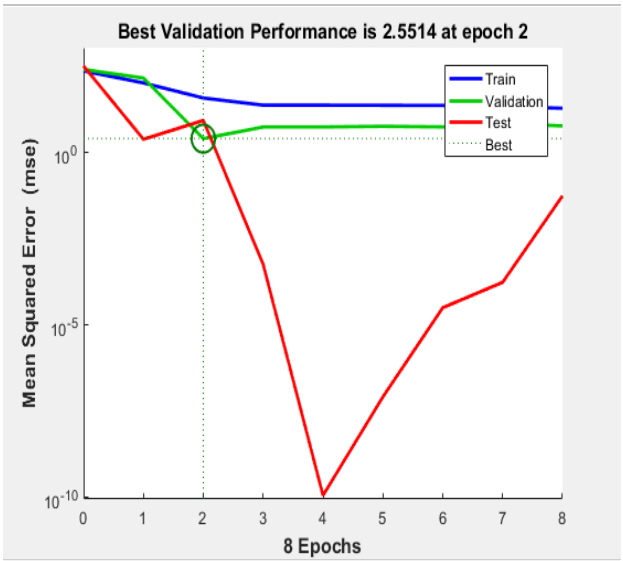

Figure.4 MSE validation for training

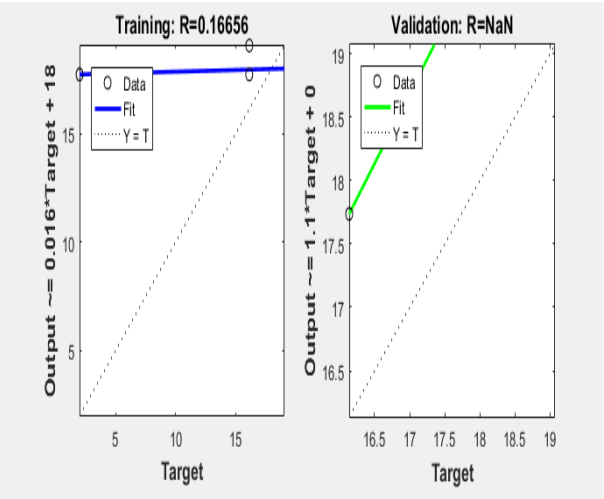

Figure.5 Regression stage1

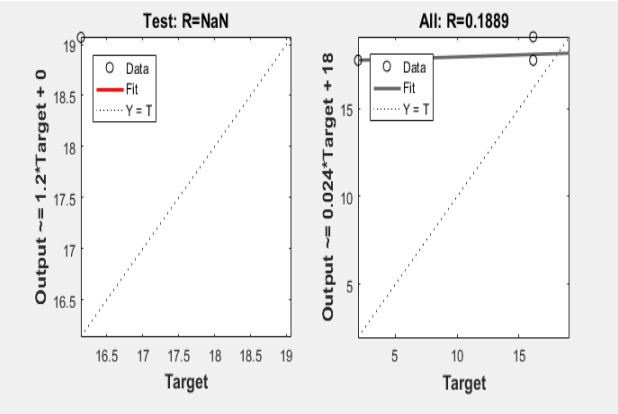

Figure.6 Regression Stage 2

This trained architecture aims to classify the data for an expected size and then it would be compared to the size calculated by COCOMO2 but when the SVM return multiple sizes then the proposed architecture utilizes SVM for the conflicted classes [25].

Algorithm Train Classify SVM (Input – Conflicted Sizes from Neural)

1. Train DataSVM=[ ]; // Empty variable to store training structure
2. Train count=0
3. for each conflict in Input For each conflict in Input
4.TrainDataSVM(Traincount,1)=Conflict Output Neural(conflict) // Input set will be conflicted output
5. GroupSVM(Train Count)=KLOCoutput Groups(Train Count)=KLOCoutput // Total target of training because it is supervised technique
6. Train Count=TrainCount+1;
7. End For
8.SVMStruct=TrainSvm (TrainDataSVM, GroupSVM);
9. TestDataSVM=TrainDataSVM; // As SVM is a supervised learning method, hence the test data will be equal to the traindata
10.SvmClass=ClassifySVM(SVMStruct,TestDataSVM)
11. Publish SvmClass
12. End Function

The following architecture diagram represented in Fig. 7 is attained after the training of SVM.

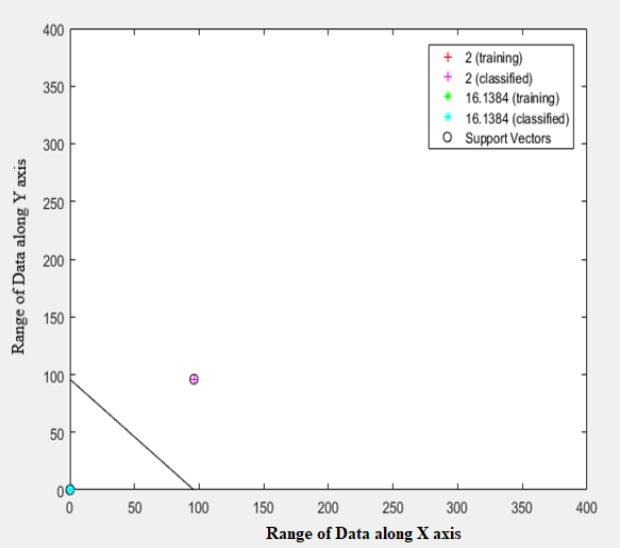

Figure.7 Training architecture of SVM

Table 2.Size difference and MSE of the proposed architecture

Project samples	Real Size	COCOMO 2 size	Pro-posed Size	MSE in size estimation	Validation with Diverse Neurons
1	890	900	879	0.023	10
2	900	934	869	0.069	12
3	899	945	872	0.084	14
4	920	963	899	0.066	16
5	910	978	865	0.012	18

Results represented in table 2 have been evaluated as per the proposed framework.

Table 2 represents the numeral value analysis of the proposed architecture with a varying number of neuron validations. Varying neurons will result in a change in the regression model and its regression analysis. An increasing number of neurons rapidly do not assure the true desired result. Hence, an incremental two neuron set has been applied and, on an average, for that data set used, 18 neurons have been evaluated to provide optimal results.

Graphical representation represented in Figs. 8 and 9 is developed according to Table 2.

Results shown in Figs. 8 and 9 have demonstrated that the proposed framework that sustains minimum MSE of 0.0115 and the difference in sizes is also not high.

The average difference between proposed architecture and COCOMO2 size is 30% which are appreciable. The proposed architecture has utilized the features of both Neural Network and Support Vector Machines very wisely and as a result, the estimation error is very low.

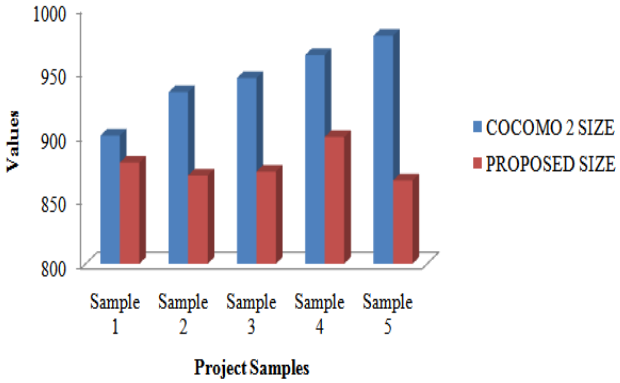

Figure.8 Size comparison

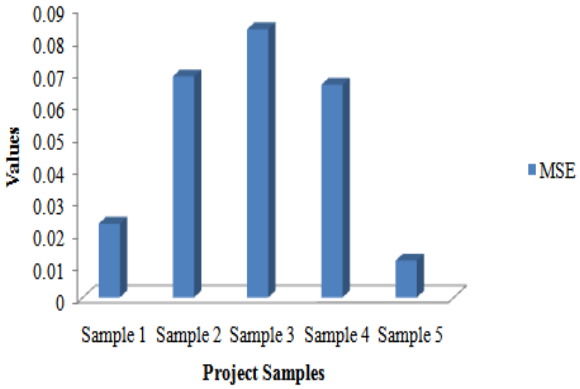

Figure.9 MSE representation

Table 3. Comparison of True Positive Rate (TPR) for COCOMO-2 and Proposed

Iteration	True positive rate by A. B. Nassif et al. (2013) [1]	True positive rate-Proposed
1	0.561	0.695
2	0.589	0.6935
3	0.5796	0.6993
4	0.556	0.7012
5	0.5896	0.7011

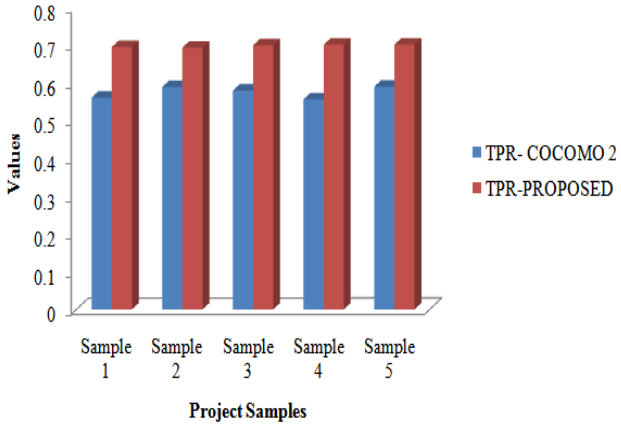

Figure.10 Comparison of true positive rate of existing and proposed work

Table 4.Comparison of MSE for COCOMO-2 and Proposed

MSE by S. Lohmor and B.B. Sagar (2017) [22]	MSE- Proposed
0.87	0.0115

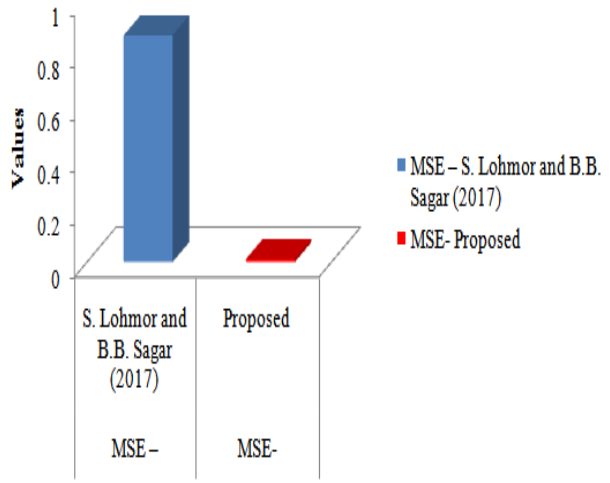

Figure.11 Comparison of MSE of existing and proposed work

Fig. 10 and Table 3 represents the True Positive Rate (TPR) of the proposed architecture and existing work by A. B. Nassif et al. (2013) [1]. TPR is the proportion of the total number of accurate prediction to the total number of predictions as shown in Eq. (7).

$$TPR = \frac{Correct\ Predictions}{Total\ Predictions} \quad (7)$$

The proposed model remains consistent for all the project samples undertaken. The maximum TPR attained by the proposed model is 0.71 whereas the COCOMO2 model exhibits a total TPR of 0.58. In this paper, we have compared the proposed work with work by A. B. Nassif in which the researcher has used COCOMO-2. The proposed work has used the combination of COCOMO-2 along with Neuro-SVM technique to overcome the problem of COCOMO-2 and the improvement is shown in the above figures.

Fig. 11 and Table 4 depicts the comparison of MSE for proposed work and existing work by S. Lohmor and B.B. Sagar (2017) [22]. MSE is the mean of the squares of the difference among the observed value and the estimated value of the software errors being detected. Mathematically, it could be represented as:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(n_i - n(q_i))^2 \quad (8)$$

In Eq. (8), n_i is the estimated real failures and $n(q_i)$ is the observed failures. It can be seen from the contrast that the proposed work has less MSE value as compared with the existing work by Lohmor and B.B. Sagar.

5. Conclusion and future scope

The proposed architecture has utilized the effectiveness of COCOMO-2 and has combined it with NN and SVM architecture to make the estimation more precise. The result section demonstrates that the proposed mechanism is more precise as compared to COCOMO-2. The evaluation is done as per size estimation and MSE computation. The MSE improvement of the proposed model stands 30% more efficient than COCOMO-2. A comparison is conducted between conventional works with the proposed work. It has been seen that the proposed work has a TPR of 0.71 whereas existing work has achieved a TPR of 0.58 and the MSE with proposed work is less that is 0.0115 and the MSE with existing is 0.87.

The current research work has a wide scope of advancement. Different neurons could be used to check the difference in the MSE. Other than that, instead of SVM, Linear Discriminant Analysis can also be utilized.

Acknowledgments

We would like to thank the Punjab Technical University, Jalandhar for extending amazing support to us. Further, we wish to thank all who helped us for finishing this work.

References

[1] A.B. Nassif and L. F. Capret, “Towards an early software estimation using log-linear regression BMI and a multilayer perceptron model”, Journal *of Systems and Software*, Vol. 86, No. 1, pp.144-160, 2013.

[2] A.J. Albrecht and J.E. Gaffney, “Software function, source lines of code, and development effort prediction: a software science validation”, *IEEE Transactions on Software Engineering*, Vol. 9, No.6, pp. 639-648, 1983.

[3] E. Borandag, F. Yucalar, and S. Z. Erdogan, “A case study for the software size estimation through MK II FPA and FP methods”, *International Journal of Computer Applications in Technology*, Vol.53, No.4, pp. 309-314, 2016.

[4] P. Achimugu, A. Selamat, R. Ibrahim, and M. Naz’riMahrin, “A systematic literature review

of software requirements prioritization research", Information *and software technology*, Vol. 56, No.6, pp.568-585, 2014.

[5] E. Kocaguneli, T. Menzies, and E. Mendes, "Transfer learning in effort estimation", *Empirical Software Engineering*, Vol.20, No.3, pp .813-843, 2015.

[6] R. Sarno, J. Sidabutar, and Sarwosri, "Comparison of different Neural Network architectures for software cost estimation", In: *Proc. of the International Conference on Computer, Control, Informatics and its Applications*, pp. 68-73, 2015.

[7] R. Dillibabu and K. Krishnaiah, "Cost estimation of a software product using COCOMO II. 2000 model–a case study", *International Journal of Project Management*, Vol.23, No.4, pp.297-307, 2005.

[8] S.S. Shwartz and N. Srebro, "SVM optimization: inverse dependence on training set size", In: *Proc. of the 25th international conference on Machine learning*, pp.928-935, 2008.

[9] J. Joanna F. DeFranco and P. A. Laplante, "Review and Analysis of Software Development Team Communication Research", *IEEE Transactions on Professional Communication*, Vol. 60, No.2, pp.165-182, 2017.

[10] M. Shahin, M.A. Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment", *A Systematic Review on Approaches, Tools, Challenges and Practices*, *IEEE Access*, Vol. 5, pp. 3909-3943, 2017.

[11] I. Bate, A. Burns, and R. I. Davis, "An enhanced bailout protocol for mixed criticality embedded software", *IEEE Transactions on Software Engineering*, Vol.43, No.4, pp.298-320, 2017.

[12] O. Pedreira, F.Garcia, N. Brisaboa, and M. Piattin, "Gamification in software engineering–A systematic mapping", *Information and Software Technology*, Vol.57, pp 157-168, 2015.

[13] L. G. Wallace and S. D.Sheetz, "The adoption of software measures: A technology acceptance model (TAM) perspective", *Information & Managemen*t, Vol.51, No.2, pp.249-259, 2014.

[14] A. Sharma and D.S. Kushwaha, "Estimation of Software Development Effort from Requirements Based Complexity", *Procedia Technology*, No. 4, pp.716-722, 2012.

[15] K. Pandey and N.K. Goyal, "Introduction. Early Software Reliability Prediction", *Springer, India*, pp. 1-16, 2013.

[16] S. M. Satapathy, M. Kumar, and S. K. Rath, "Fuzzy-class point approach for software effort estimation using various adaptive regression methods", *CSI Transactions on ICT*, Vol.1, No.4, pp.367-380, 2013.

[17] M.J Basavaraj and K.C Shet, "Software Estimation using Function Point Analysis: Difficulties and Research Challenges", *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, pp.111-116, 2007.

[18] J. J. Dolado, "A validation of the component-based method for software size estimation", *IEEE Transactions on Software Engineering*, Vol.26, No.10, pp.1006-1021, 2000.

[19] A.L.I. Oliveir, "Estimation of software project effort with support vector regression", *Neuro computing*, Vol.69, No. 13-15, pp.1749-1753, 2006.

[20] Heiat, "Comparison of artificial neural network and regression models for estimating software development effort", *Information and software Technology*, Vol.44, No.15, pp.911-922, 2002

[21] G.R. Finnie, G.E. Wittig, and J-M. Desharnais, "A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models", *Journal of Systems and Software*, Vol.39, No.3, pp. 281-289, 1997.

[22] S. Lohmor and B.B. Sagar, "Estimating the parameters of software reliability growth models using hybrid DEO-ANN algorithm", *International Journal of Enterprise Network Management*, Vol.8, No.3, pp.247-269, 2017.

[23] J. Razmi, R. Ghodsi, and M. Jokar, "Cost estimation of software development: improving the COCOMO model using a genetic algorithm approach", *International Journal of Management Practice*, Vol. 3, No. 4, pp.346-368, 2009.

[24] A. Kaushik, A. K. Soni and R. Soni, "A hybrid approach for software cost estimation using polynomial neural networks and intuitionistic fuzzy sets", *International Journal of Computer Applications in Technology*, Vol. 52, No. 4, pp. 292-304, 2015.

[25] F. W. Crawford, J. Wu, and R. Heimer, "Hidden population size estimation from respondent-driven sampling: a network approach", *Journal of the American Statistical Association*, pp. 1-12, 2018.