International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 / https://doi.org/10.14445/22315381/IJETT-V70I12P215

Volume 70 Issue 12, 138-146, December 2022
© 2022 Seventh Sense Research Group®

Original Article

Optimizing Design of Software Size Estimation model using Neural Network

Manisha[1], Rahul Rishi[2], Sonia Sharma[3], Renu[4]

[1,2,3]UIET, Maharshi Dayanand University, Rohtak, India
[4]Gaur Brahman Degree College, Rohtak, India

[1]Corresponding Author : mvmanishavatsa@gmail.com

Received: 19 August 2022 Revised: 13 November 2022 Accepted: 25 November 2022 Published: 24 December 2022

Abstract *- Size Estimation has always been an area of interest in the software industry. Estimating size directly could lead to the calculation of storage identities and costs. This paper proposes a neural network-based size estimation method which utilizes the architecture of Machine Learning. In this paper, the k-means algorithm is used to divide the data into multiple segments, which is further utilized by the Fuzzy logic-based inference engine to generate the class labels. In this model, the NASA-based PROMISE Dataset has been utilized, and there is no class label containing the project size. In order to validate the class label, the collected data is passed to a multi-class classifier which uses the Levenberg principle. The proposed model is evaluated using quantitative parameters, namely the class and overall class accuracy, and is compared with other classification architectures. The accuracy of the proposed model has been improved by 9.7% in comparison with other techniques and 0.7% in comparison to existing studies.*

Keywords *- Class accuracy, Fuzzy logic, Machine learning, Neural network, Size estimation.*

1. Introduction

Size estimation has always attracted the research world and the real-time software industry. NASA conducted an experiment in which 93 projects were evaluated based on different aspects of development, including the programmer's ability and processing knowledge, the total number of human resources applied to complete the project, etc. The aggregated data is presented in terms of a dataset known as the PROMISE dataset. It has been widely used in Software Engineering by a lot of researchers [1-3]. The dataset represents OOM metrics and computes the overall effort of the project calculated utilizing metrics. The categorization of the metrics has been done on the base of software evolution. The procedure of evolving a software product utilizing software engineering standards and systems is alluded to as software evolution. It incorporates the introductory improvement of software and its upkeep and upgrades until aved programming product is created, fulfilling development begins from the necessity collecting process. After which, engineers make a prototypical model of the expected software design and reveal it to the clients to get their input in the early phase of software product development. The clients recommend changes, on which a few sequential redesigns and upkeep also continue evolving. This procedure changes to the first software till the coveted software is succeeded [8]. The software program is expensive and time-consuming to develop and has a high cost in commercial, informational systems. The amount of investment in software programs is estimated to be close to $200 billion per annum. Boehm suggested thoroughly performing cost and benefit analysis before providing the required resources for the software project [9-10]. The correctness of software investment decisions is directly proportional to the quality of the software. If the cost and effort for the project are underestimated, such projects are liable to be abandoned mid-way. This withdrawal of the project is either owing to high costs which were not estimated at an early stage or due to the wrong estimation of time and related resources required. If the cost and effort for the project are overestimated, such overestimation may increase the project cost by putting less effort into programmers to be innovative.

Moreover, in these cases, the potential projects will likely be rejected due to high costs or time. Software size estimation is crucial for both software designers and ends users [11-15]. They can be used to create requests for proposals, contract negotiations, planning, and management. Underestimating the cost and effort may result in management quickly approving the project, and if the budget is increased, the project may fail, as it has in the past, with underdeveloped functions and poor quality. Overestimation may result in various resources being dedicated to the project, resulting in job losses [16-20]. Moreover, research on machine learning and soft computing techniques with software effort estimation was authored until early 2007. Earlier, authors provided a summary of effort estimation

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

using machine learning techniques that will be useful for researchers in offering future direction in the field of machine learning adoption in software effort estimation [21-23].

The limitation of the current study is precise designing and efficient resource allocation in line with the effort estimation of software for different projects. However, data mining and ML techniques provide a way for effort estimation but are limited in providing the desired results due to problems in fine-tuning and classification errors.

Furthermore, the main problem in conventional studies is the realistic estimation of cost since it is used to determine the resources required for the project and at what intervals each particular resource will be allocated to each activity, as well as to classify, rank, and arrange development projects in terms of the overall business strategy. It is simple to observe and evaluate the impact of changes, and it is simple to replan. Connecting the size to effort could be a dicey situation; hence, based on the evaluated effort, this paper categorizes the size into three categories and modifies the training architecture using Neural Network.

2. Related Work

Machine learning-based systems are known to be dynamically adaptable to any type of data, and researchers are eager to estimate the effort more precisely. Numerous strategies fail to effectively achieve the development goal for test applications that give the quick evolution of software products [27]. This research proposes a unique methodology to estimate the effort based on ensemble learning and feature reduction. The feature ranking and selecting mechanism allows the practitioners to estimate effort using criteria like size and cost using the suggested method [28]. Simulated results using the suggested technique and the COCOMO II dataset are encouraging. Much work has been discussed in Table 1, which shows the comparative analysis of the methodology used by different researchers from 2018 to 2022. The limitation of individual work has also been mentioned. Various authors worked on techniques like Deep Belief Networks (DBN), Swarm Optimization, Deep Neural Networks, use cases and Actors, Regression, etc., to analyze different data sets for software estimation. Different techniques and work methodologies helped the authors attain the research's valuable target and explain their findings in the paper. This table presented a perspective of the use of machine learning in software estimation utilising various methodologies implemented by different authors, which is highly helpful for academics to provide the future path in the prescribed field. The comparison of different techniques, research gaps, and limitations of these techniques is also shown in the table below.

Table 1. Comparative Analysis of Related Work

Author	Technique	Work and Methodology	Limitation / Research Gap
Kaushik et al. in 2022	Deep Belief Network (DBN) along with the swarm-based Whale Optimization Algorithm (WOA)	A technique for software estimation considering the datasets COCOMO81, MAXWELL, CHINA, and NASA93 is discussed. Restricted Boltzmann Machine and Deep Belied Network (DBN) have been constructed, and DBN was estimated with WOA for fine-tuning.	The main limitation is the use of the traditional dataset. Parameters need to be optimized.
Khan et al. (2021)	Grey Wolf Optimizer (GWO) and Strawberry (SB), along with the DNN	The software error estimation was done to solve the multi-variable problem, and nine benchmark functions were used for comparison. The weight function had been levied in addition.	There is a need for improvement in training, and therefore research efforts are required using Deep Learning architectures.
R Silhavy et al. 2021	Actors and Use cases	Estimating software system's boundaries and derivation of software system actors and finding alternative use cases. The authors also focused on stepwise regression.	Testing window functions need to employ, and different project sizes need to be investigated
M Daud, and AA Malik in 2021	Analysis-to-Design Adjustment Factors (ADAFs)	The practical and empirical implications validate the applicability of determining the different metrics related to the class diagram along with adjustment factors.	There is a need to investigate the impact of frameworks designed using ADAFs
A.Banimustafa in 2018	Naïve Bayes, Logistic Regression and Random Forests.	NASA and COCOMO benchmarks that covered 93 projects and produced models were tested to evaluate the Precision, Recall, and classification accuracy.	The potential of data mining ML techniques is to be explored, and estimation accuracy needs to be enhanced.

Surendiran 2019	Development of secure software estimation during the software development life cycle.	The maximum software efforts estimated to track the design and analyzing the risk using the different techniques	The study is limited to identifying a secure system's key parameters and development.
N. Qomariah et al. 2020	Multiple Linear regression through associate research	Quantitative research for statistical data analysis and validity tests is conducted to determine the customer satisfaction rate.	The regression coefficient approaches 0.290 for service quality which limits the dependent variable.
S. Chhabra et al 2020	fuzzy logic-based COCOMO using PSO	PSO algorithm is used to compute the magnitude of relative error. The evaluation using COCOMO NASA had been considered to validate the proposed model.	The study is limited to validating the evaluation metrics for different datasets.
A. Ardiansyah et al 2022	PSO along with chaotic inertia weight map.	Agile and COCOMO case sets were used, and two PSO variants were employed for software estimation. The learning strategy applied was used to enhance the search mechanism.	The generation numbers were limited, and there is a need to enhance the performance of the proposed system.
N. A. Zakaria et al 2021	PSO to optimize the cost estimation model	PSO was employed in conjunction with a hybrid model of SVM, Linear Regression, and Random Forest to optimize the parameters.	The study is limited to optimize the COCOMO parameters as there is still a need for improvement to evaluate all attributes

Table 2. Fuzzy Rule Set

$\boldsymbol{Ifclassified_{label}}$> 30% Margin	very high software size is not beneficial as a project
Else if $\boldsymbol{Classified_{label}}$10% Margin and $\boldsymbol{Classified_{label}}$ <30% Margin	Moderate in software size and manageable if handled carefully
If Classified Label< 10% Margin	Low in computation and highly profitable as a project

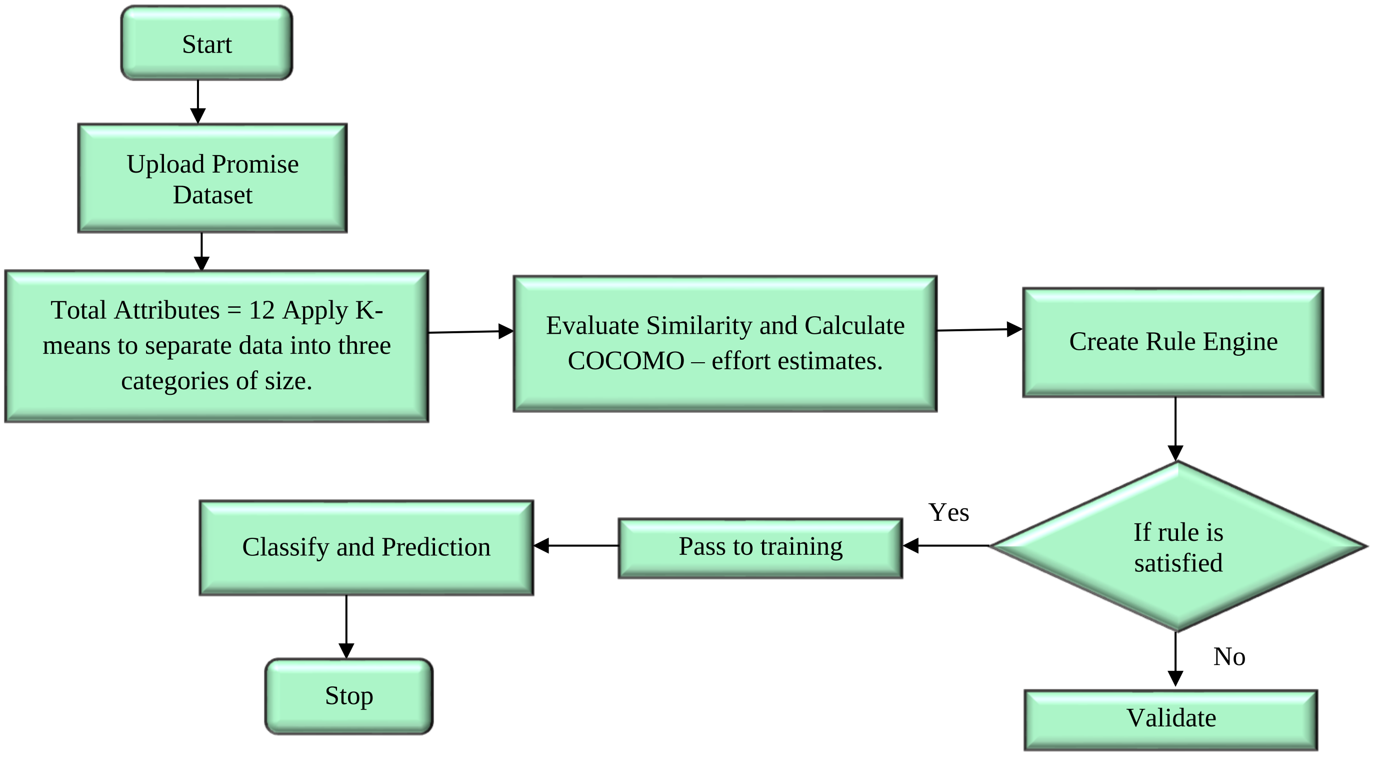

Start
Upload Promise Dataset
Total Attributes = 12 Apply K-means to separate data into three categories of size.
Evaluate Similarity and Calculate COCOMO – effort estimates.
Create Rule Engine
If rule is satisfied
Yes
Pass to training
Classify and Prediction
Stop
No
Validate

Fig. 1 A Flow Chart for Proposed Architecture

3. Proposed Technique and Methodology

The flow chart for the proposed architecture of size estimation is depicted in Fig 1. The K-Means algorithm is applied to a total of 12 attributes of the PROMISE dataset, and similarity is calculated using COCOMO-2 effort. A Fuzzy rule engine is created, as shown in Table 2, which is further evaluated to see if the rule is satisfied or not. If the rule is satisfied, it is passed to the training module, which subsequently classifies and predicts. In order to validate the class label, the collected data is passed to a multi-class classifier, which uses the Levenberg principle to validate. The proposed model is evaluated using quantitative parameters, namely the class and overall class accuracy, and is compared with other state-of-the-art classification architectures.

The proposed methodology is divided into binary segments Seg 0 and Seg 1. Seg 0 applied Machine Learning (ML) for the evaluation of size labels

$$\text{Labels} \in \{\text{High}_{\text{Size}}(0), \text{Moderate}_{\text{Size}}(1), \text{Low}_{\text{size}}(2)\} \qquad (1)$$

The identified class is set to be (0), i.e. High Size if the co-relation between the group elements is high and the predicted effort is also high. In order to predict the effort of the software project, the proposed methodology uses COCOMO-2 Effort estimation architecture.

1. Development Effort
2. A basic version of the Basic COCOMO effort equation is as follows:

$$\text{MM} = \text{a} * \text{KDSI}^{\text{b}} \qquad (2)$$

Which is based on MM - man-month / person-month / staff-month is one month of effort by one person.

3. Efforts and Development Time (TDEV)

$$\text{Effort} = \text{a} * \left(\text{KLOC}^{\text{b}}\right) * \left(\textstyle\prod_{\text{j}} \text{EM}_{\text{j}}\right) \qquad (3)$$

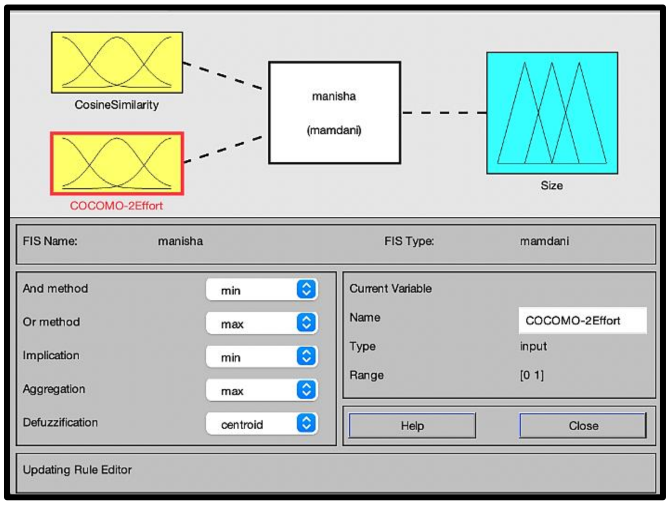
$$\text{TDEV} = 2.5 * \text{MM}^{\text{c}} \qquad (4)$$

The coefficients a, b and c depend on the development mode. A fuzzy inference engine is designed to separate the class labels, as demonstrated in Fig. 2, 3, and 4. In order to pass the data with ground truth values, the inference engine is supplied with cosine similarity and calculated effort of the groups. The data is separated utilizing the k-means algorithm by passing the 12-attribute set to the separation mechanism. The K-means is known for its placement architecture of the objects to its related cluster based on the distance evaluated using eq. (5) [6-8].

$$d = \int_{i=1}^{n_r} \int_{k=1}^{n_r} \int_{j=1}^{atc} (a_{ij} - a_{ik})\text{^}2 \qquad (5)$$

where n_r is the total number of records in the list and act is the total number of attributes in the list viz.12. The proposed algorithm calculates the average cosine similarity of each cluster by using eq. (6) and passes it to the fuzzy inference engine.

$$Cos_{sim} = \frac{CA_i.CB_i}{||CA_i|| \times ||CB_i||} \qquad (6)$$



FIS Name: manisha
FIS Type: mamdani
And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid
Current Variable
Name: COCOMO-2Effort
Type: input
Range: [0 1]
Help
Close
Updating Rule Editor

Fig. 2 Fuzzy Inference Engine

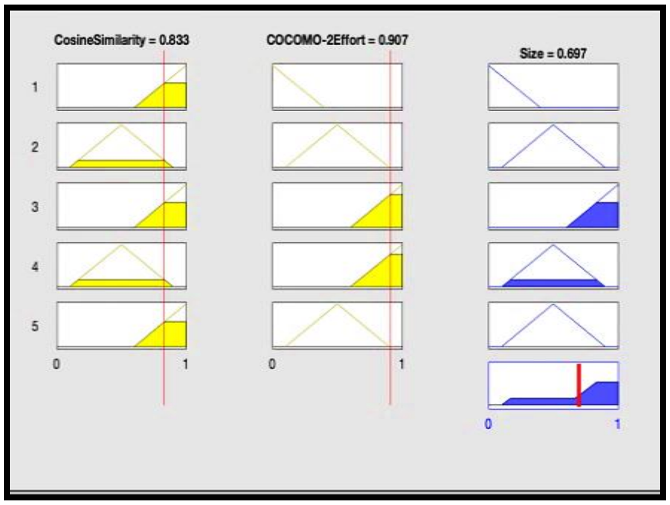


Fig. 3 The Rule Engine

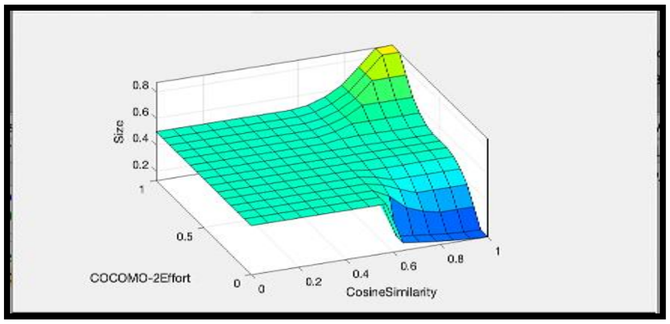
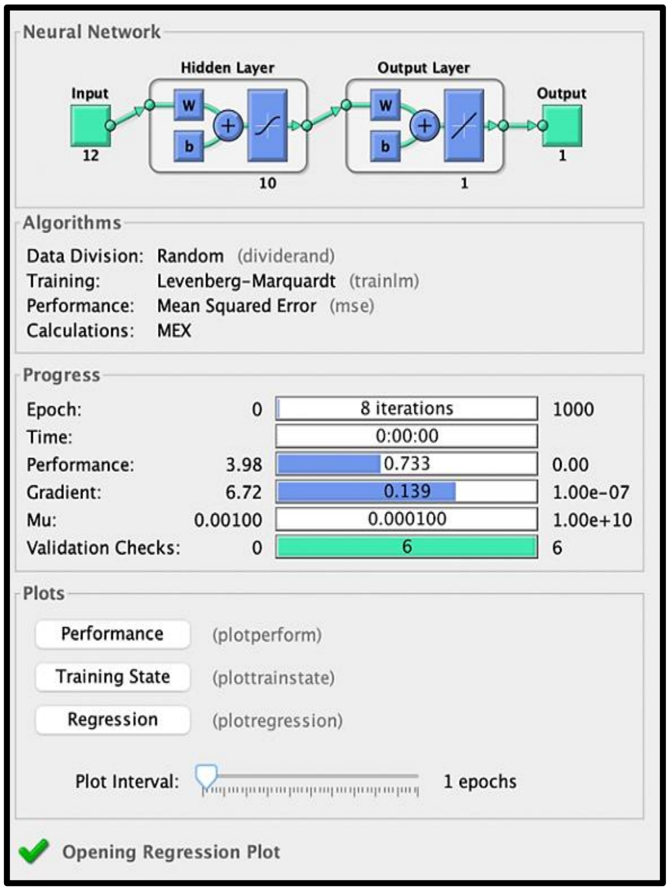


Fig. 4 The Surface View



Neural Network

Hidden Layer
Output Layer
Input
Output
W
b
W
b
12
10
1
1

Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

Epoch:	0	8 iterations	1000
Time:		0:00:00	
Performance:	3.98	0.733	0.00
Gradient:	6.72	0.139	1.00e-07
Mu:	0.00100	0.000100	1.00e+10
Validation Checks:	0	6	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

✓ Opening Regression Plot

Fig. 5 Neural Network

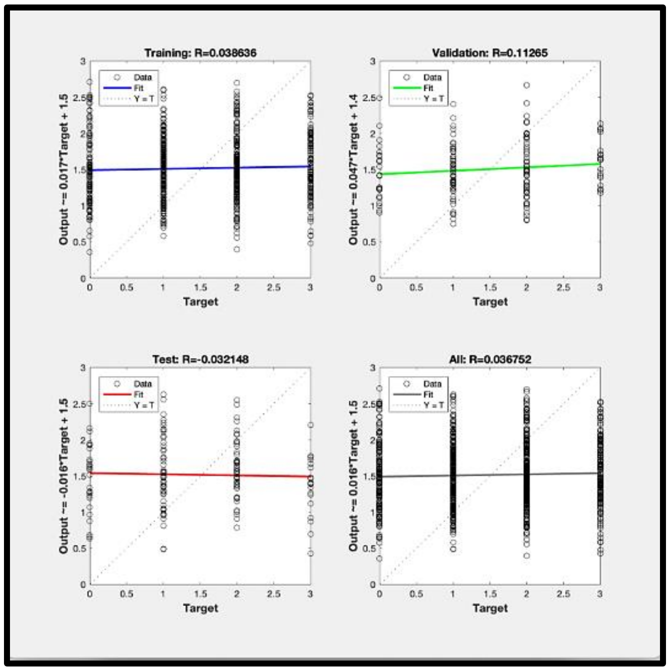


Fig. 6 Regression Propagation

The categorized class is passed to multi-class classifier-based Neural Networks, which incorporate the categorization based on the Levenberg propagation principle. The stopping criteria for such algorithmic architecture depend on multiple factors to attain the maximum training accuracy incorporated against the classes.

```
1PC.  Pseudo Code: Neural Training Engine
2PC.  Inputs: Separated Data (Sd), Attribute Set (As) : Outputs: Trained architecture, Classification Metric
3PC.  $\text{Training}_{\text{Data}} = [\,]$// Initialize a training data metric to empty //
4PC.  $\text{Training}_{\text{Group}} = [\,]$ // Initialize training group //
5PC.  For each class is Sd // Extract the total classes separated by fuzzy inference engine //
6PC.  $\text{Training}_{\text{Data}}.\text{Append}(\text{As.class.data})$ // Append data values of the selected class //
7PC.  $\text{Training}_{\text{Group}}.\text{Append}(\text{class})$ // Append Class value
8PC.  End For
9PC.  Initiate Neural Engine
10PC. Total Epochs = 100;
11PC. Training.validation = Mean Squared Error(MSE) // The training engine will be validated by MSE
12PC. $\text{Propagation}_{\text{type}} = \text{Levenberg}$;
13PC. $\text{Validation}_{\text{ratio}} = .70$ // Pick only 70 % data for the training to check result//
14PC. $\text{Propagation}_{\text{Layer}} = 10$ // Pick 10 layers of propagation network //
15PC. Start Training( ) // Initiate Training //
16PC. $\text{Validate}(\text{Test}_{\text{Data}}, \text{Trained}_{\text{Data}})$;
17PC. Stop;
18PC. Return validation
```

Table 3. Regression values for proposed work based on multiple layers

'TOTAL NUMBER OF LAYERS'	'R TRAINING'	'R VALIDATION'	'R TEST'	'OVERALL R'
3	0.07456009	0.1745015	0.10126737	0.11677632
5	0.19420026	0.1820251	0.41423499	0.26348678
8	0.09513968	0.35115863	0.25978745	0.23536192
10	0.28017973	0.47102497	0.44884813	0.40001761
13	0.2976054	0.75503402	0.6566659	0.56976844
15	0.72530482	0.14197195	0.70718684	0.5248212
18	0.1582384	0.01500296	0.78464565	0.31929567
20	0.03594338	0.80671423	0.65133264	0.49799675
23	0.38844263	0.61567457	0.69300804	0.56570841
25	0.56002239	0.76794695	0.18561648	0.50452861

Table 4. Class Accuracy

Total number of objects/software projects	Class accuracy proposed c1	Class Accuracy proposed c2	Class accuracy proposed c3	Class accuracy c1 Naïve Bayes	Class accuracy c2 Naïve Bayes	Class accuracy c3 Naïve Bayes
92	91.1033852	91.6599947	90.5565415	89.233	89.553	89.665
150	92.6702589	93.6214411	91.6810141	90.7424692	90.0874233	90.4895898
300	92.8118828	94.3605123	92.5298893	91.8233403	90.7516069	90.9717328
500	93.7541031	94.591133	94.0890525	92.644946	92.7131661	91.1343712
1000	94.3853899	95.2952678	96.0301499	94.2547288	94.1665498	91.4408218
2000	96.3630656	97.1531224	97.3409622	95.0891116	96.1162956	92.739186
3000	96.6751047	98.1890338	98.7075347	95.3610846	97.2650536	94.1458567

The proposed methodology illustrates the working and training mechanism of Neural Networks based on the architecture that is proposed. The input layer contains 12 feature vector values that have been propagated under various layers of propagation. As shown in Figure 5, the neural network propagates for 10 layers producing one output architecture. In order to validate the propagation, regression is analyzed to finally sum up and check what layer count satisfies the overall training architecture. The regression is divided into three segments: the R-value for training, validation and testing. The analysis regression value is computed by performing the mean of all the regression values, as shown in Fig 6. The regression values for the propagation are demonstrated in Table 3. The propagation network has been supplied with a total of 1000 epochs. The network does not need to run for all provided epochs; if the gradient of the data satisfaction is attained before it reaches the maximum supplied epochs, it will stop the training. The network propagation stopped at 8 iterations as validation checks were completed. After the 8th iteration, it will propagate backwards to check the best possible regression value. As illustrated in Table 4, the proposed training for supplied data input values gains a maximum R value with 13 layers; hence, the training architecture generated at the 13th layer will be used for classification.

4. Result and Discussion

The evaluation of the proposed technique is done by comparing the performance with the state of art technique classifiers, namely Neural Network, Naïve Bayes and multi-class SVM.

The trained architecture is classified and evaluated for two parameters, namely class accuracy and overall Classification Accuracy, by using (7) and (8).

$$\text{Classification Accuracy} = \frac{\text{No of identified objects in class}_i}{\text{Total number of objects in class}_i} \quad (7)$$

$$\text{Overall Accuracy} \frac{\sum_{i=1}^{cs} \text{Classification Accuracy}}{cs} \quad (8)$$

It is evident from Figure 1 that the proposed algorithm outcasts the existing architecture by a significant margin. Ninety-two records have been utilized from the NASA dataset, and the rest have been generated through the Monte Carlo simulation. As the training data increases, the classification rate increases due to more availability of the records to the training mechanism. Though the comparative algorithms also perform well over newly simulated and PROMISE data, they are significantly behind by a maximum margin of 6% in the overall accuracy comparison. The maximum attained overall accuracy through the proposed algorithm is 98%. The validation for a number of layers has been evaluated using the regression value R, which has been evaluated for all aspects, including training, validation and testing. For the supplied data set, the best regression value was attained on the 13th layer; hence, the architecture generated at the 13th layer has been utilized for training and classification. A bifurcation of 70-30 in training and test data is insulated to further check the overall classification accuracy. The obtained overall accuracy is shown in Fig. 7. The overall evaluated accuracy of the proposed model is 98% for a set of 3000 records in the list.

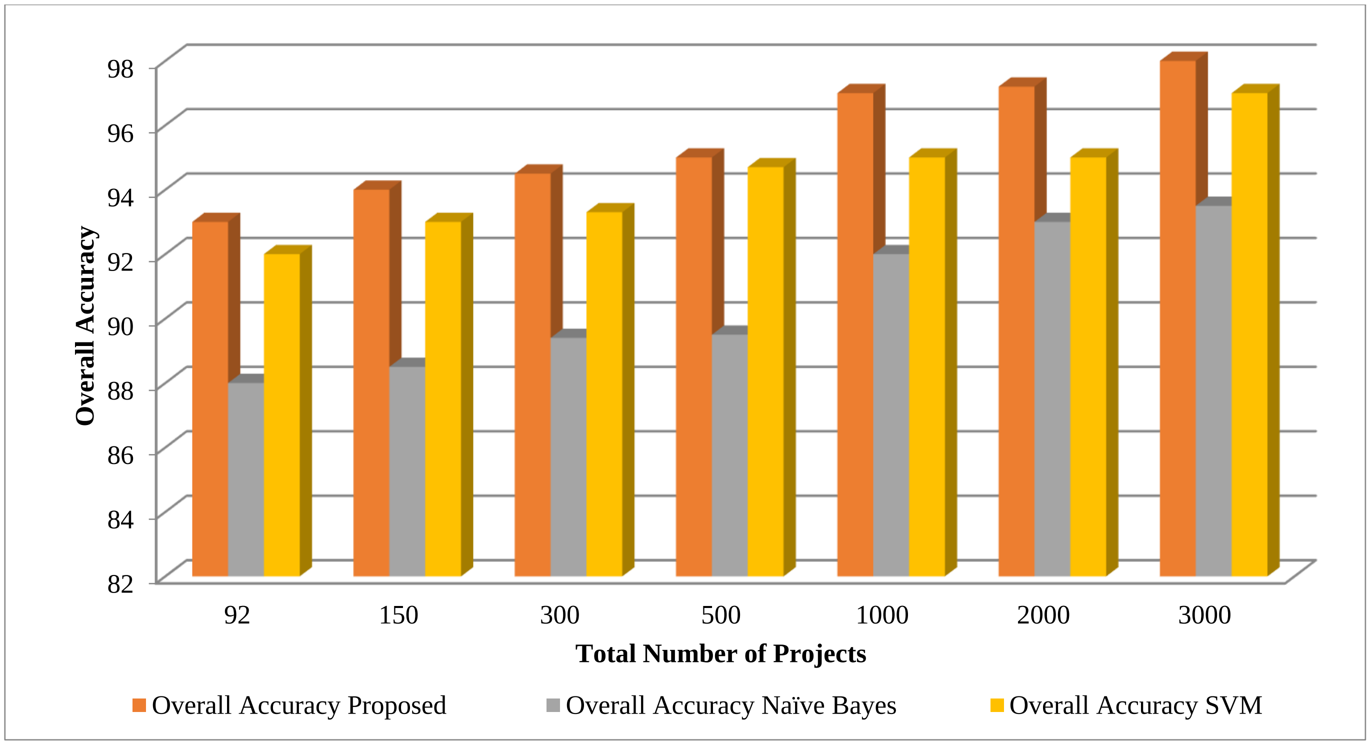


Fig. 7 Overall Accuracy vs total number of projects

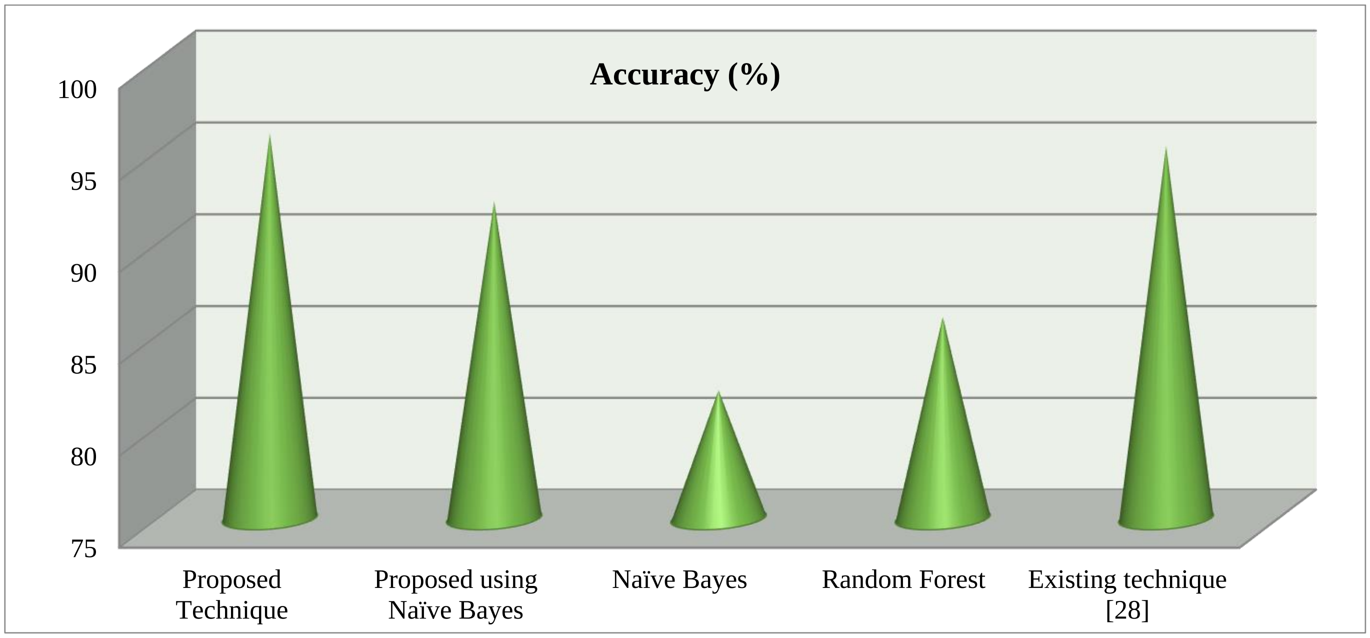


Fig. 8 Comparative Analysis of Accuracy with the existing Techniques

The minimum attained accuracy is 91% for the proposed algorithm system, whereas the other state of art techniques, including Neural Network, Naïve Bayes and multi-class SVM, remain under 95% even for the highest amount of training data. The classification accuracy has been compared with the existing techniques for validation, as shown in Fig 8. The proposed model's outcomes show an improvement of about 13% and 9.7% compared to Naïve Bayes [18] and Random Forest [18]. However, the proposed approach compared to Naïve Bayes improves by 3%. However, the existing technique [28] proposed ensemble learning for software test effort estimation but was limited to providing the desired results with an accuracy of about 95.31%.

Thus, consistent and prominent results have been obtained. The proposed technique is improved by 0.7% from the existing studies. Future work on this subject may focus on the possibility of using ML techniques and refining the model and drivers of the COCOMO model, both of which could improve the model's predictive performance and lower the error rate for its prediction.

5. Conclusion

The paper presents a machine learning-oriented solution to the size estimation utilizing a k-means algorithm supported by a fuzzy inference engine. The separated data is passed to a multi-class classifier model, illustrated based on the Levenberg propagation model, which propagates the data into multiple layers against the size labels. The sizes have been evaluated using two propagation parameters, namely, class accuracy and overall accuracy. The overall accuracy is the arithmetic mean of the class accuracies. The propagation architecture utilized the PROMISE dataset and extended the dataset using Monte Carlo simulation. The proposed work used Levenberg-oriented training and classification mechanism as the primary classifier; hence the illustration of the selection has been provided in the proposed work section. The validation for a number of layers has been evaluated using the regression value R, which has been evaluated for all aspects, including training, validation and testing. For the supplied data set, the best regression value has been attained on the 13th layer; hence, the architecture generated at the 13th layer has been utilized for training and classification. A bifurcation of 70-30 in training and test data has been insulated to check the overall classification accuracy further. The overall evaluated accuracy of the proposed model is 98% for a set of 3000 records in the list. The minimum attained accuracy is 91% for the proposed algorithm system, whereas the other state of art techniques, including Neural Network, Naïve Bayes and multi-class SVM, remain under 95% even for the highest amount of training data. The accuracy of the proposed model has been improved by 9.7% compared to the state-of-the-art techniques.

References

[1] Anupama Kaushik, Niyati Singal, and Malvika Prasad, “Incorporating Whale Optimization Algorithm with Deep Belief Network for Software Development Effort Estimation,” *International Journal of System Assurance Engineering and Management*, pp. 1637–1651, 2022. *Crossref,* https://doi.org/10.1007/s13198-021-01519-8

[2] Noor Azura Zakaria et al., “Optimization of COCOMO Model using Particle Swarm Optimization,” *International Journal of Advances in Intelligent Informatics*, vol. 7, no. 2, pp. 177-187, 2021. *Crossref,* https://doi.org/10.26555/ijain.v7i2.583

[3] BaniMustafa A, “Predicting Software Effort Estimation using Machine Learning Techniques,” *2018 8th International Conference on Computer Science and Information Technology (CSIT), IEEE,* pp. 249-256, 2018. *Crossref,* https://doi.org/10.1109/CSIT.2018.8486222

[4] Giuliano Antoniol, R. Fiutem, and Chris Lokan, “Object-Oriented Function Points: An Empirical Validation,” *Empirical Software Engineering*, vol. 8, no. 3, pp. 225-254, 2003. *Crossref,* https://doi.org/10.1023/A:1024472727275

[5] Ashman R, “Project Estimation: A Simple Use-Case-Based Model,” *IT professional*, vol. 6, no. 4, pp. 40-44, 2004. *Crossref,* https://doi.org/10.1109/MITP.2004.41

[6] Shashank Mouli Satapathy, Barada Prasanna Acharya, and Santanu Kumar Rath, “Early Stage Software Effort Estimation Using Random Forest Technique Based on Use Case Points,” *IET Software*, vol. 10, no. 1, pp. 10-17, 2016. *Crossref,* https://doi.org/10.1049/iet-sen.2014.0122

[7] Radek Silhavy, Petr Silhavy, and Zdenka Prokopova, “Using Actors and Use Cases for Software Size Estimation,” *Electronics*, vol. 10, no. 5, p. 592, 2021. *Crossref,* https://doi.org/10.3390/electronics10050592

[8] Marriam Daud, and Ali Afzal Malik, “Improving the Accuracy of Early Software Size Estimation Using Analysis-to-Design Adjustment Factors (ADAFs),” *IEEE Access*, vol. 9, pp. 81986-81999, 2021. *Crossref,* https://doi.org/10.1109/ACCESS.2021.3085752

[9] Sonia Chhabra, and Harvir Singh, “Optimizing Design of Fuzzy Model for Software Cost Estimation using Particle Swarm Optimization Algorithm,” *International Journal of Computational Intelligence and Applications*, vol. 19, no. 1, p. 2050005, 2020. *Crossref,* https://doi.org/10.1142/S1469026820500054

[10] Barry Boehm, Chris Abts, and Sunita Chulani, “Software Development Cost Estimation Approaches—A Survey,” *Annals of Software Engineering*, vol. 10, no. 1, pp. 177-205, 2000.

[11] Pinkashia Sharma, and Jaiteg Singh, “Systematic Literature Review on Software Effort Estimation using Machine Learning Approaches,” *2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS), IEEE,* pp. 43-47, 2017. *Crossref,* https://doi.org/10.1109/ICNGCIS.2017.33

[12] Wasiur Rhmann, Babita Pandey, and Gufran Ahmad Ansari, “Software Effort Estimation using Ensemble of Hybrid Search-Based Algorithms Based on Metaheuristic Algorithms,” *Innovations in Systems and Software Engineering*, vol. 18, no. 2, pp. 309-319, 2022. *Crossref,* https://doi.org/10.1007/s11334-020-00377-0

[13] Manisha, Rahul Rishi, and Sonia Sharma, "Improved Data Segmentation Architecture for Early Size Estimation using Machine Learning," *International Journal of Advanced Computer Science and Applications,* vol. 13, no. 6, pp. 738-747, 2022. *Crossref,* https://doi.org/10.14569/IJACSA.2022.0130687

[14] Omar Hidmi, and Betul Erdogdu Sakar, "Software Development Effort Estimation Using Ensemble Machine Learning," *International Journal of Computing Communications and Instrumentation Engineering*, vol. 4, no. 1, pp. 143-147, 2017. *Crossref,* https://doi.org/10.15242/IJCCIE.E0317026

[15] Siti Hajar Arbain, Nor Azizah Ali, and Noorfa Haszlinna Mustaffa, "Adoption of Machine Learning Techniques in Software Effort Estimation: An Overview," *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 551, no. 1, p. 012074, 2019. *Crossref,* https://doi.org/10.1088/1757-899X/551/1/012074

[16] Manisha, and Rahul Rishi, "An Enhanced Metaheuristic Based Cuckoo Search Algorithm for Software Size Estimation," *2021 4th International Conference on Recent Developments in Control, Automation & Power Engineering (RDCAPE), IEEE,* pp. 526-520, 2021. *Crossref,* https://doi.org/10.1109/RDCAPE52977.2021.9633575

[17] Luigi Lavazza, and Sandro Morasca, "Empirical Evaluation and Proposals for Bands-Based COSMIC Early Estimation Methods," *Information and Software Technology*, vol. 109, pp. 108-125, 2019. *Crossref,* https://doi.org/10.1016/j.infsof.2019.02.002

[18] Muhammad Sufyan Khan et al., "Metaheuristic Algorithms in Optimizing Deep Neural Network Model for Software Effort Estimation," *IEEE Access*, vol. 9, pp. 60309-60327, 2021. *Crossref,* 10.1109/ACCESS.2021.3072380

[19] Ardiansyah Ardiansyah, Ridi Ferdiana, and Adhistya Erna Permanasari, "MUCPSO: A Modified Chaotic Particle Swarm Optimization with Uniform Initialization for Optimizing Software Effort Estimation," *Applied Sciences*, vol. 12, no. 3, p. 1081, 2022. *Crossref,* https://doi.org/10.3390/app12031081

[20] Pandey Prateek, and Litoriya Ratnesh, "Fuzzy AHP-Based Identification Model for Efficient Application Development," *Journal of Intelligent & Fuzzy Systems*, vol. 38, no. 3, pp. 3359-3370, 2020. *Crossref,* https://doi.org/10.3233/JIFS-190508

[21] Akanksha Baghel, Meemansa Rathod, and Pradeep Singh, "Software Effort Estimation using Parameter-Tuned Models," *arXiv preprint arXiv:2009.01660,* 2020. *Crossref,* https://doi.org/10.48550/arXiv.2009.01660

[22] Vahid Khatibi Bardsiri, and Mahboubeh Dorosti, "An Improved COCOMO-based Model to Estimate the Effort of Software Projects," *Journal of Advances in Computer Engineering and Technology*, vol. 2, no. 2, 2016.

[23] Manisha, and Rahul Rishi, "Early Size Estimation using Machine Learning," *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom), IEEE,* pp. 757-762, 2021.

[24] Daniel Spikol, et al., "Supervised Machine Learning in Multimodal Learning Analytics for Estimating Success in Project-Based Learning," *Journal of Computer Assisted Learning*, vol. 34, no. 4, pp. 366-377, 2018. *Crossref,* https://doi.org/10.1111/jcal.12263

[25] Nurul Qomariah, Achmad Fahrurrozi, and Yusron Rozzaid, "Efforts to Increase Retail Customer Satisfaction," *SSRG International Journal of Economics and Management Studies,* vol. 7, no. 7, pp. 23-29, 2020. *Crossref,* https://doi.org/10.14445/23939125/IJEMS-V7I7P105

[26] R. Surendiran, "Secure Software Framework for Process Improvement," *SSRG International Journal of Computer Science and Engineering,* vol. 3, no. 1, pp. 19-25, 2016. *Crossref,* https://doi.org/10.14445/23488387/IJCSE-V3I12P105

[27] Hanan Qassim Jaleel, "Testing Web Applications," *SSRG International Journal of Computer Science and Engineering,* vol. 6, no. 12, pp. 1-9, 2019. *Crossref,* https://doi.org/10.14445/23488387/IJCSE-V6I12P101

[28] K. Eswara Rao, and G. Appa Rao, "Ensemble Learning with Recursive Feature Elimination Integrated Software Effort Estimation: A Novel Approach," *Evolutionary Intelligence*, vol. 14, no. 1, pp. 151-162, 2021. *Crossref,* https://doi.org/10.1007/s12065-020-00360-5