International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 / https://doi.org/10.14445/22315381/IJETT-V70I5P208
Volume 70 Issue 5, 60-73, May 2022

Original Article

An Advanced Testing Effort Calculation Algorithm Architecture using GA and Neural Network

Vikas Chahar[1], Pradeep Kumar Bhatia[2]

[1, 2]*Department of Computer Science & Engineering, G. J. University of Science & Technology, Hisar, India.*

[1]vikas.chahar@gmail.com

Received: 31 January 2022 Revised: 13 April 2022 Accepted: 01 May 2022 Published: 19 May 2022

Abstract - *Early effort estimation holds a significant role when a new project is planned. Otherwise, whenever some changes are appended to the software system during the development phase, they need to be tested again to assure maintenance of software quality. Thus, in both situations, effort estimation plays a key role in completing a project. The paper presents improved software test effort estimations based on similarity analysis and metaheuristics. The designed model considers the effort classification into three test effort classes, namely, low, moderate, and high effort. in the process, the Genetic Algorithm (GA) is integrated for the attribute selection from NASA and Promise datasets used during the evaluation of the proposed model. The three similarity analysis techniques, Cosine, Jaccard, and Euclidean distance, are integrated to find the similarity in individual datasets fed to k-means cluster the data into three clusters. The test effort class prediction performed based on the designed rule set is used to categorize the effort class based on MSE and SE as the validation parameters in the presented work. The simulation analysis performed using two datasets shows the improved test effort predictions by integrating the concept of metaheuristics.*

Keywords - *Test Effort Estimation, Genetic Algorithm, Cosine Similarity, Jaccard Similarity, Euclidean Distance.*

1. Introduction

Effort estimation is one of the key activities performed at the initial stages of project planning and management. Numerous studies have associated the software development process with effort and size estimation. [1,2] However, the precise and accurate estimation remained an open challenge. in Software Development Life Cycle (SDLC), software testing is the mechanism used to analyze the difference between the actual and the expected results. in simpler terms, the goals of the software testing are straight and easy to understand but very difficult to be met. Mere completion of the project is not always sufficient, and it also requires testing to assure that the project functions properly. The primary objective of software testing is to remove bugs and improve the enormous aspects of software like performance, user experience, security, and so on. [3,4] The best deal of testing can enhance the overall quality of the software, which leads to great customer satisfaction.[5] The two broad categories of software testing, namely, static and dynamic testing, are described in Table 1 while highlighting the major difference between them.

Table. 1 Major type of software testing techniques

Static Testing	**Dynamic Testing**
in Static testing, testing is performed without the execution of the program.	in Dynamic testing, testing is performed by the execution of the program
It executed in the initial stage of software development	It is executed at the later stage of software development.
This testing does the verification process. This testing	does the validation process.
The static testing is related to the defect prevention	While dynamic testing is related to finding and fixing the defects.
It provides an assessment of code and documentation.	It produces bugs or bottlenecks in the software system.
for testing process, it involves a checklist.	It comprises test cases for execution purposes.
The execution of this testing is done before the compilation	Execution of this testing is done after compilation
It takes less time to find defects, and it is also cheap	It takes more time as compared to static testing, and it is very expensive

It completes 100% statement coverage incomparably less time	Whereas this testing only accomplishes less than 50% statement coverage.
This testing can discover a variety of errors.	It can only expose the exportable bugs by execution, thus discovering only a few kinds of bugs.

Although the tests are performed throughout the SCLC, the software is produced component by component.

1.1. Test Effort Estimation

In test planning, the key element is tested effort estimation, which requires will planned effort estimation to schedule various testing activities. The test effort estimation is usually performed using test estimation performed by employing a work breakdown structure. The test effort estimation engages the requirement analysis at the testing stage. Before performing any sort of estimations, the requirements of the product to be designed and tested need to be clearly understood. This required a detailed review of the application prototype. Depending on the complexity and the size of the application involved, the effort of nearly 1 to 2 weeks is required by the tester to perform test effort estimations. [6] Test effort is generally calculated in terms of total consumed time to evaluate the outcome of the software based on the supplied set of input. As the test effort term appears in the dataset, a link has to be created between the existing estimation methods for effort or quality. The proposed algorithm architecture presents a novel method of co-relating the test effort with the other effort measures and is illustrated in the methodology section.

The rest of the paper is organized in the following manner. The second section contains the literature survey, whereas the proposed methodology is illustrated in section 3. The results are evaluated and are presented in section 4, and the paper is concluded in section 5.

2. Literature Review

The section presents the literature review conducted for effort estimation workaround NASA, PROMISE, COCOMO, etc., as key terms. Yadav and Singh (2014) had integrated the optimization concept of a nature-inspired Genetic Algorithm (GA) to improve the effort estimation performed using the COCOMO-II model. The comparative analysis was performed using NASA and PROMISE project data resulting in RMSE of 8.868 using the COCOMO-II model after integration of GA for fine-tuning of parameters. The resultant project-specific parametric values were 2.564 and 0.862. Overall, it was observed that the metaheuristics significantly produced better results for both datasets. [7] Shivakumar et al. (2016) suggested a non-algorithmic method focused on adaptive neuro-fuzzy logic to resolve the problem of accuracy and reliability in software effort estimation models. The analysis showed that the model produced efficient effort estimation using the NASA dataset. The effort estimation accuracy was further evaluated using MMRE and MRE to analyze the divergence between the estimated and actual effort. The overall evaluation showed that the hybrid neuro-fuzzy method improved 11% in terms of MMRE. [8] Saljoughinejad and Khatibi (2018) also proposed integrating three metaheuristic algorithms to improve the accuracy of the estimations performed using the COCOMO model. The main stress was given to analyzing the cost drivers using metaheuristics. The selection of the coefficients was performed, followed by the reconstruction of the COCOMO. The resultant effort estimations show that the integration of optimization approaches, namely, PSO, GA, and Invasive Weed Optimization (IWO), resulted in enhanced estimation accuracy evaluated in terms of MMRE and PRED. [9] Sehra et al. (2019) had proposed a hybrid approach that combines a multi-criteria decision-making technique with the machine learning algorithms to improve the accuracy of the project effort estimations. The feature ranking was performed using the Fuzzy analytic hierarchy process, and the generated ranks were integrated into the least square support vector machine to perform further effort estimations. The evaluation was performed using MMRE and RMSE against NASA and COCOMO datasets. [10] Attri and Bal (2019) proposed an automated technique to estimate the size using AI. The training and classification had been done using the COCOMO model and evaluation in terms of MSE and size considering the project samples. The study used to estimate the size in three different parts, and the minimum error computed was 0.0115. The results were further compared for effectiveness in terms of TPR and MSE. [11] Kumar et al. (2019) introduced the effort estimation performed using Particle Swarm Optimization (PSO). The quantification of the effort improvement is performed using the Kilo Line of Code. The paper further investigates the effective parameters for the effort estimation. The proposed model resulted in better effort estimation with an MMRE of 56.57 compared to the COCOMO model with an MMRE of 245.39. [12] Chhabra and Singh (2020a) presented non-algorithmic modeling using soft computing techniques using COCOMO NASA dataset. The approaches implemented were genetic algorithms and fuzzy logic that can handle ambiguous definitions of the cost drivers. The fuzzy set parameters were then optimized for the selection performed using the GA fitness function. The evaluation showed a 25% improvement in terms of MMRE and PRED that was achieved due to the stability of GA in optimizing the fuzzy model. [13] Chhabra and Singh (2020b) had presented an effort estimation model based on the Fuzzy Inference System that could effectively compute the corresponding effort multiplier. It was presented that the uncertainty related to the intermediate COCOMO cost driver could be

addressed with FIS. The work involved the PSO as an evolutionary optimization technique used to optimize the fuzzy logic-based COCOMO model. The relative error matrices associated with the effort estimation using NASA and NASA2 datasets were used to validate the proposed work. [14] Suresh & Behera (2020) presented a comparative analysis of popular machine learning models, namely, SVM, RF, NN, KNN, and backpropagation, in improving the strength of software effort estimation. The orange data mining tool was also integrated into the methodology to evaluate the proposed work using the COCOMO'81 dataset comprising 63 projects and the Desharnais dataset comprising 81 projects. The comparative analysis showed that among various ML techniques, the backpropagation model achieved the most efficient effort estimation. [15]

Rhmann et al. (2021) presented a software effort estimation based on weighted hybrid search algorithms. The authors created the weighted ensembles using metaheuristic algorithms, namely, black hole optimization, genetic algorithm, and firefly algorithm. The evaluation of the effectiveness of the ensemble of metaheuristics algorithms to improve the ML-based prediction strength of effort estimation models was evaluated using different datasets present in the PROMISE repository. The R programming language using RKEEL and Metaheuristics forms the basis of the performed simulation analysis. The simulation analysis showed that the metaheuristics-based prediction demonstrated more realistic effort estimations. [16] Zakaria et al. (2021) had optimized the COCOMO-II model using PSO followed by the integration of machine learning algorithms, namely, SVM, Linear Regression (LR), and Random Forest (RF). The authors performed effort estimation using NASA dataset. The comparative analysis of the three algorithms showed that PSO with SVM outperformed the other variations, namely PSO with LR and PSO with RF. It was observed that PSO significantly improved the estimation accuracy of the COCOMO model by optimizing its parameters using the concept of metaheuristic swarm intelligence. The parameters used in evaluating the proposed work were correlation accuracy, MMRE, and p-value. [17] Ardiansyah et al. (2022) had proposed a modified chaotic PSO to overcome the shortcoming of the traditional PSO. This improved PSO was then implemented using three schemes, namely, chaotic inertia mapping, uniform initialization, and stochastic learning-based effort estimation models, to evaluate its effectiveness over PSO. The detailed analysis showed that the proposed chaotic PSO algorithm resulted in uniform particle initialization while avoiding getting trapped in local optimization solutions evaluated using three methods, namely, COCOMO, Agile, and UCP. [18] Kaushik et al. (2022) proposed a technique to address the effort estimation challenges using combining Whale optimizations with a deep belief neural network. Whale optimization is a technique inspired by the social behavior of humpback whales. The datasets used for the experimental evaluation of the proposed work were COCOMO81, MAXWELL, CHINA, and NASA93. It was concluded that the involvement of the optimization technique resulted in fine-tuning of the parameters and improved effort estimations compared to the deep belief network backpropagation model. [19] Ardiansyah et al. (2022) improved the performance of the PSO using the case points COCOMO model to avoid the trapping problem in local optima. The authors worked in an explorative manner and generated the solutions considering the inertia weight and software effort estimation had been done using the uniform approach. The outcomes were expressive and improvised with the enhancement of PSO. [20]

The literature survey showed that testing effort represents the total time consumed to test all the supplied blocks to build a software product. [21] Further, the overall computation time also depends upon the OOPM metrics, which have significantly improved time by time with the contribution of researchers. Numerous researchers have integrated COCOMO into their research work due to various features. for example, the COCOMO dataset developed by NASA gets an upgrade in terms of storage elements, and STO has been amended as a valid metric in OOPM architecture. [22, 23, 24] The present research work aims to compute the testing effort based on OOPM oriented model.

3. Methodology

In addition to estimating the test effort, the proposed design is also based on segregating and selecting the OOPM metric system and labeling the data based on the similarities calculated between project elements. The labeling of the selected dataset is performed in the preliminary phase. The overall work methodology is described in Figure 1.

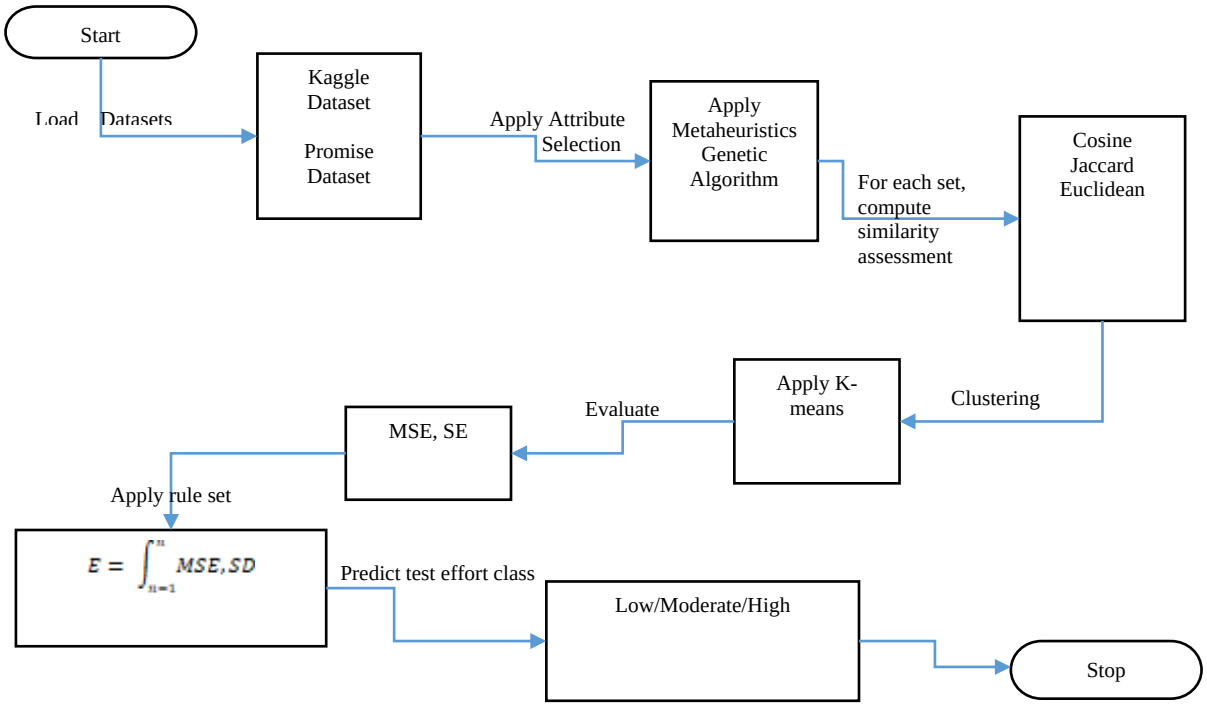

Fig. 1 Work Methodology

3.1. Data Source

Two dataset sources have been utilized in the proposed work and given similar treatment at each step.

3.1.1 Kaggle

The first dataset used to evaluate the proposed work is obtained from the global kaggle repository. The dataset covers more than 90 projects. It can be accessed at Kaggle.com [25], which provides OOPM-based quantitative and qualitative attributes covering various aspects of software testing.

3.1.2 Promise

It is the second dataset that has also been popularly used by various researchers in their effort estimation work. [26, 27] The dataset comprises matrices from 93 software projects labeled with actual effort and can be accessed at PROMISE online repository. [28]

3.2. Model Development

The design of the proposed model is described in two segments in which the first segment performs the data labeling. in contrast, the second one is dedicated to the training of the system to segregate supplied effort labels. in the initial stage, the GA metaheuristics are used to select the attribute set from the supplied dataset. The algorithmic structure of the implemented GA is given below.

GA Metaheuristics Algorithm

Input Parameters:

P_{Data} → Pre-processed Data

GA Options → Initialization of GA with basic operators and functions like:

- Population Size (50) // According to P_{Data}
- Selection Function (S_{fcn})
- Mutation Function (M_{fcn})
- Crossover Function (C_{fcn})

Output Parameters: F_{Index} → Index of selected features using an algorithm based on their fitness

$S_{Features}$ → Selected Features from P_{Data}

i) Start attribute selection

ii) for x in the range of P_{Data}

$Non - Zero - Record\ (x) = Count\ (find\ (P_{Data} > 0))$

iii)

iv) $F_T = \frac{\sum_{i=1}^{x} \text{Non-Zero-Record (x)}}{x}$ // Average count of non-zero features

v) End – for

vi) Define the fitness function of GA for selection,

vii) Compute threshold value as $Thr = \frac{75 \times F_T}{100}$ // having more than 75% non-zero features compared to F_T

viii) $$Fitness\ Function, Fit_{Fun}(x) = \begin{cases} True, & F_S \geq Thr \\ False, & Else \end{cases}$$

Where, F_S represents the feature set selected for evaluation from P-Data

ix) F_{Index} = [] // Initialize a empty matrix,

x) Count = 0

xi) $[Row, Col] = Size\ (P_{Data})$ // Calculate rows and columns of pre-processed data

xii) $\boldsymbol{Foreach\ \ i\ in\ Col}$

xiii) $Data\ = P_{Data}\ (All\ Row, i)$

xiv) $FS\ =\ Count\ (find\ (Data > 0))$

xv) $Fitness\ Function\ =\ @\ (e)FitFun\ (e, FS, FT)$ // e used to store the error value

xvi) Number of Variables = 1

xvii) Fit Val = GA (Fitness Function, Number of Variables, Options)

xviii) If (Fit Val ==True)

$F_{Index}\ (Count)\ =\ i$

$Count\ =\ Count + 1$

xix) End_{If}

xx) End_{For}

xxi) $S_{Features}\ =\ P_{Data}\ (All\ Row, F_{Index})$

xxii) $Return: F_{Index}\ and\ S_{Features}$

The optimization offers tremendous problem-solving options. The post-selection of the Genetic Solver further provides an option to integrate fitness function and all the rest of the required attribute constraints required for the processing of GA. The proposed work architecture GA is implemented by considering the attributes mentioned in Table 2 as standards within the development. The Genetic Algorithm is responsible for selecting the attributes provided from the dataset. The genetic algorithm utilizes the mutation behavior of the employed elements in the list. GA performs crossover operations to justify the mutation behavior.

Table 2. GA parameters

Selection Function	$Selectionstochunif$
Mutation	Uniform, Mutation value = 0.05
Crossover	Intermediate, Crossover value = 0.08

The algorithm describes how GA selects the attributes based on progressions made in the fitness function discussed in the pseudo-code. The fitness function takes three parameters as input. The first parameter is the propagation error generated through mutation and crossover interlink.

3.3. Similarity Measurement

The next step after attribute selection is similarity analysis. The similarity measurement helps in settling down the row elements more precisely. in addition to conventionally used Euclidean distance, Cosine similarity and Jaccard Similarity has been used in the present work.

3.3.1. Cosine Similarity

The similarity is a measure of alikeness. in cosine similarity measurement, vectors compute the similarity between two entities. A vector is a quantity that exhibits both magnitudes as well as direction. It is computed as the dot product of both attributes. By determining the cosine similarity, the author will effectively try to find the angle cosine between the two objects, as shown in the following equation.

$$\boldsymbol{Cos\theta = \frac{\vec{A}.\vec{B}}{\|\vec{A}\|\|\vec{B}\|}} \quad \textbf{(1)}$$

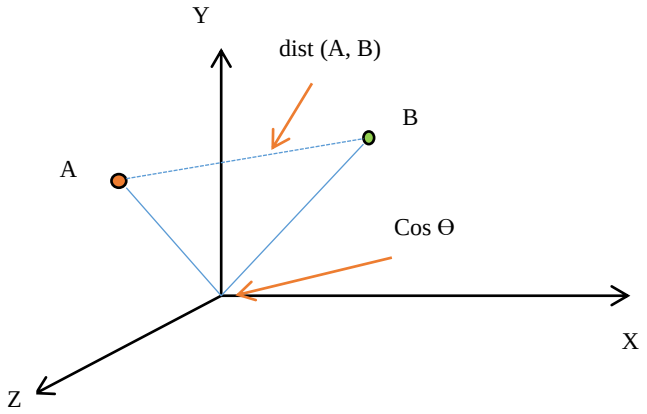

Fig. 2 Cosine Similarity

Figure 2 illustrates the similarity measurement using vectors. It is well-known that cosine0° is 1, and the vectors will have the highest similarity. Further, the cosine90° has a similarity of 0, which means that the two vectors are independent of their magnitude and have a similarity of-1.

3.3.2. Jaccard Similarity

Jaccard similarity is measured based on the common and non-common elements of a dataset value. The Jaccard similarity of two subsequent rows can be computed using the following relationship where rows are represented$r_1\ and\ r_2$.

$$\textbf{Jaccard Simialrity} = \frac{r_1 \cup r_2}{r_1 \cap r_2} \quad \textbf{(2)}$$

The combined similarity generation is performed using the following algorithm.

Similarity Measurement Algorithm
Input: dataset //attributes selected dataset
Output: Cos_{sim} //cosine similarity
$Jaccard_{Sim}$ //Jaccard similarity

1. Initialize empty matrices
2. $Cos_{sim} = []$
3. $Jaccard_{sim} = []$
4. Initialize similarity count variable as $Sim_{count} = 0;$
5. Foreach i in lenght(dataset) // Foreach row in the dataset //
6. $current_{data} = dataset\ (i);$ // Fetching current row//
7. Foreach $j = i + 1$: dataset. length // Initiating another loop that starts from 1 but goes till the value of i.//
8. k2= $Cosine(Current_{Data}, Dataset(j))|;$
9. // Evaluate cosine similarity
10. k2= $Jaccard(Current_{Data}, Dataset(j))|;$
11. $Cos_{sim}[Sim_{count}, 0] = cuurrent\_doc;0$
12. $Cos_{sim}[Sim_{count}, 1] = dataset(j);$
13. $Cos_{sim}[Sim_{count}, 2] = k1;$
14. $Jaccard_{Sim}[Sim_{count}]=k2;$
15. $Sim_{count} = Sim_{count} + 1;$
16. End_{for}
17. End_{for}
18. Return$Cos_{sim}, Jaccard_{Sim}$

The similarity measurement is performed using the above algorithm on both Kaggle and Promise datasets. The output of this step is graphically illustrated in Figure 2.

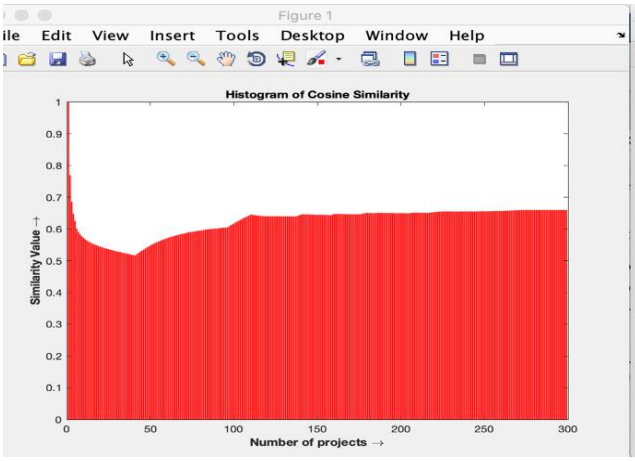

Fig. 3(a) Cosine Similarity for Kaggle dataset

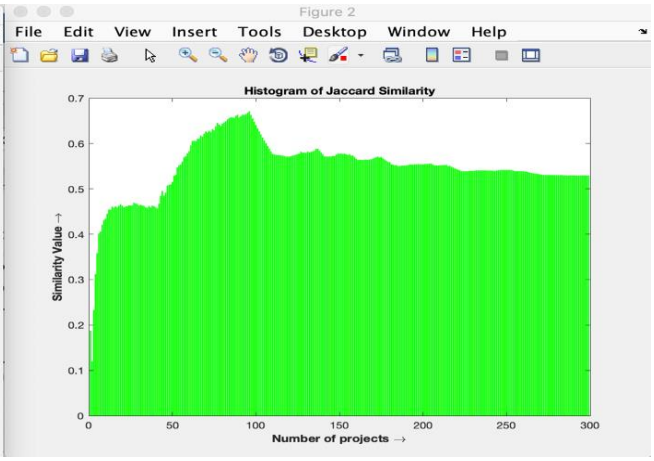

Fig. 3(b) Jaccard Similarity for Kaggle dataset

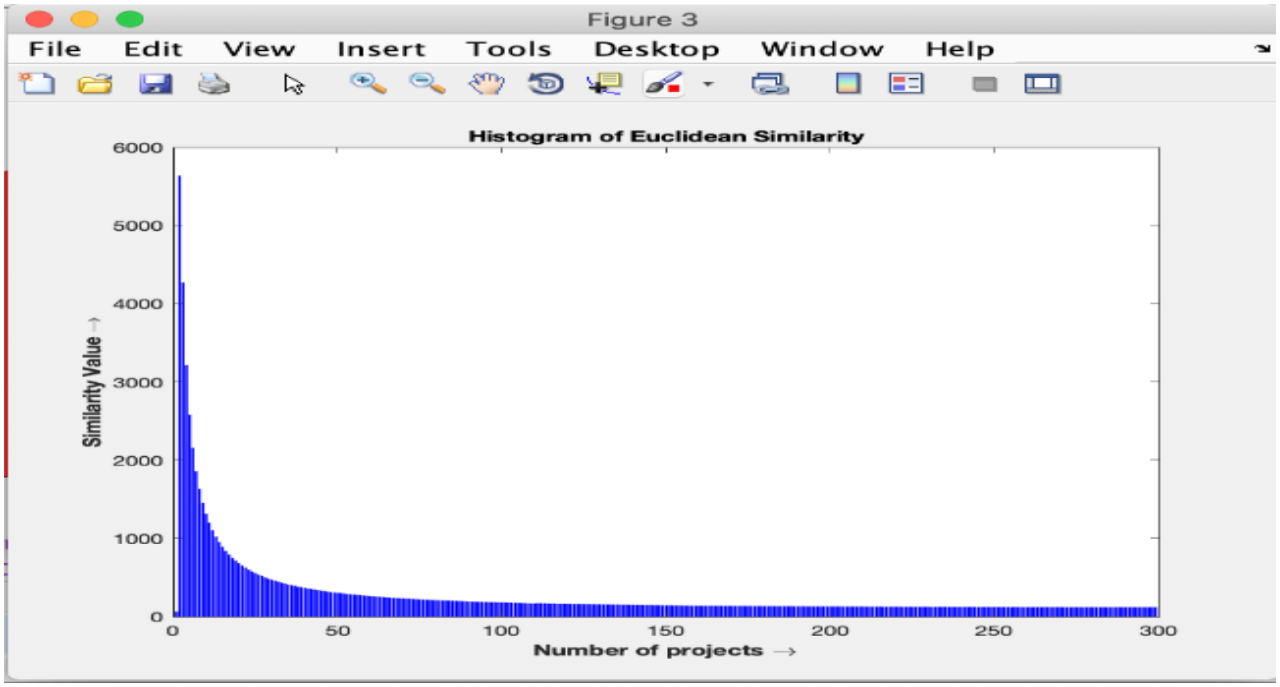

Fig. 3 (c) Euclidian distance Similarity for Kaggle dataset.

In addition to the Kaggle dataset, the similarity is also computed for the PROMISE dataset. It is to be noted that there are 97 project records in the PROMISE dataset, as illustrated by the similarity analysis graphs shown in Figure 4.

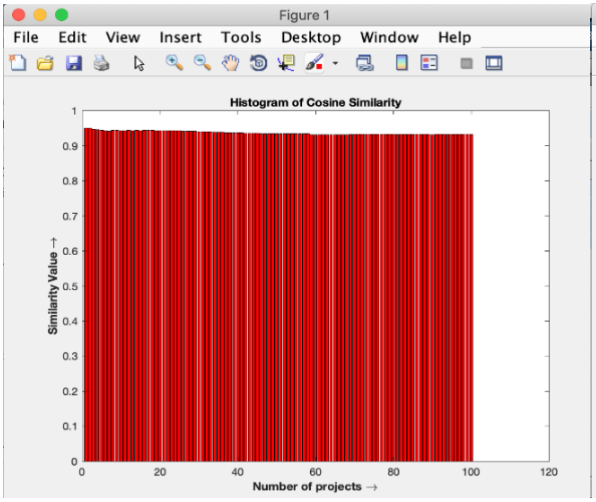

Fig. 4 (a) Cosine Similarity for Promise dataset

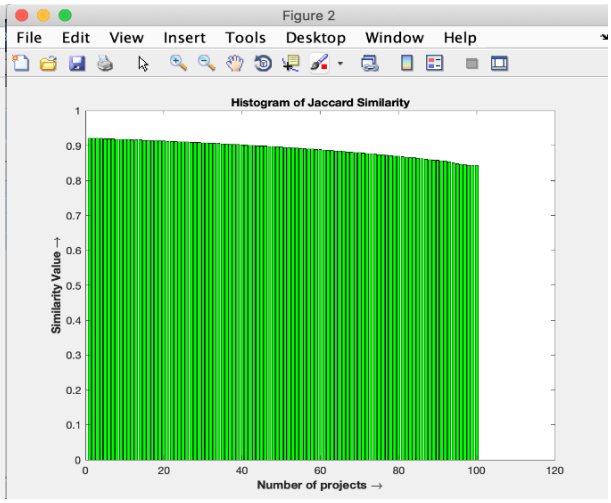

Fig. 4(b) Jaccard Similarity for Promise dataset

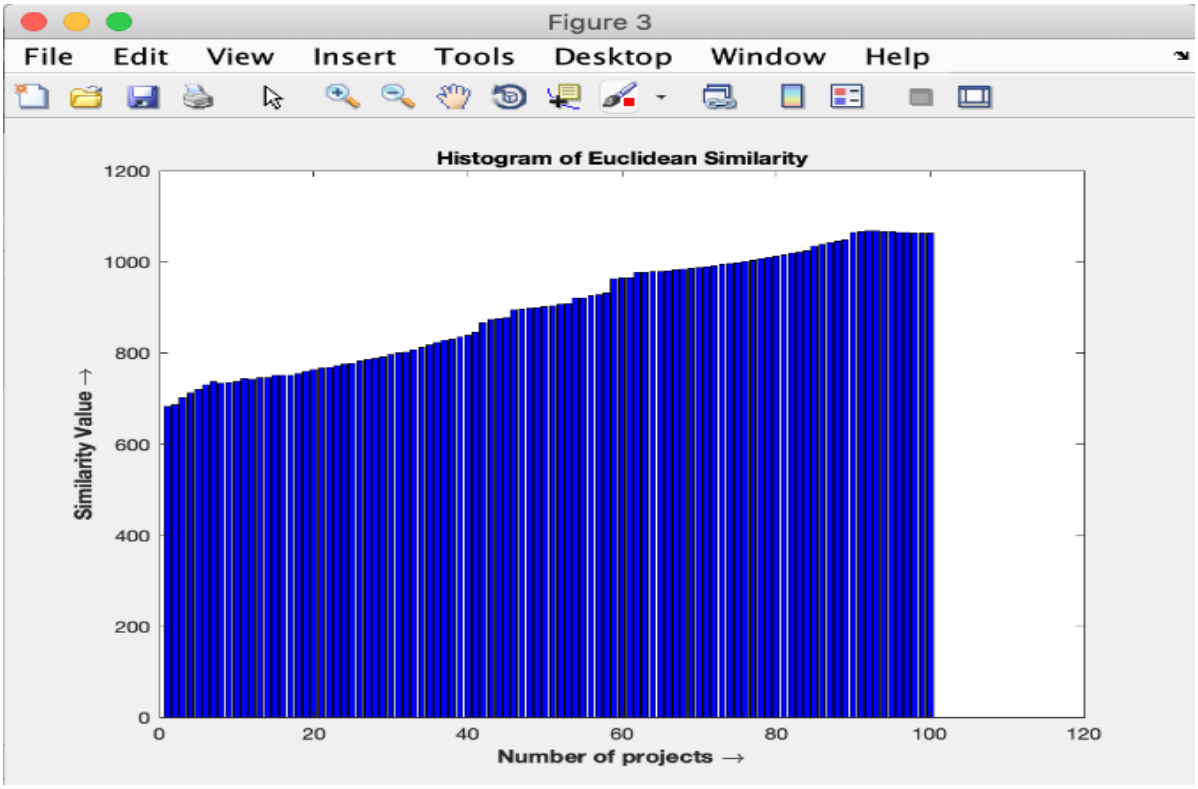

Fig. 4 (c) Euclidian distance Similarity for Promise dataset

The data is passed to the k-means algorithm containing all these features, namely Cosine similarity, Jaccard Similarity, and Euclidian distance. The k-means does not opt for the labeling of the data, and therefore, statistical architecture is integrated into the proposed work for the purpose. The approaches used to design the if-then rule set using Mean Squared Error (MSE) and Standard Error (SE).

3.4. A rule set design based on Validation Parameters

3.4.1 Mean Squared Error (MSE)

It is defined as the mean of the squares of the errors incorporated in an estimation work. Mathematically, it can be represented by the following equation in which n a number of predictions are performed on a sample comprising of n data points.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Observed - Predicted) \quad (3)$$

3.4.2. Standard Error (SE)

It is a statistical term that defines accuracy associated with the sample distribution representing a population using standard deviation. It is computed using the following relationship for a dataset comprising n a number of samples and σ as standard deviation.

$$SE = \frac{\sigma}{\sqrt{n}} \quad (4)$$

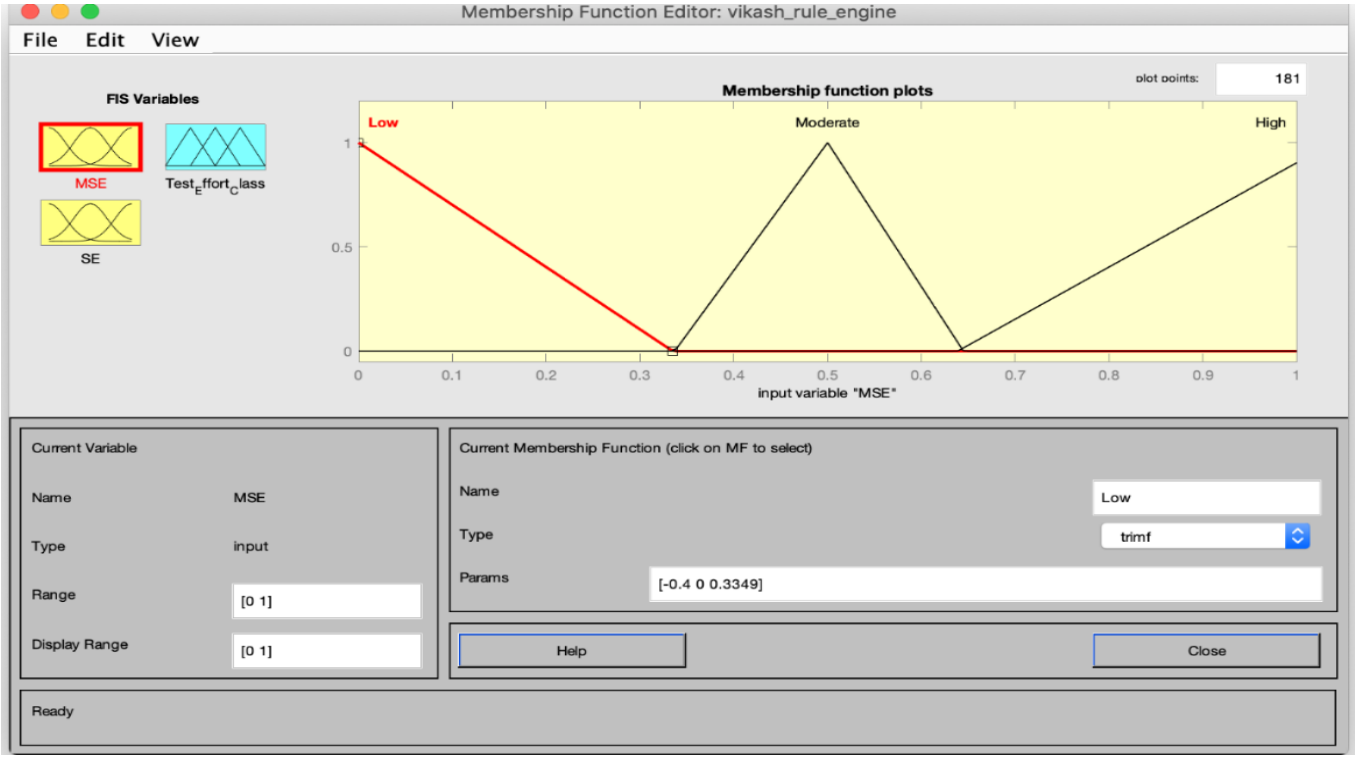

Fig. 5 Fuzzy inference engine with inputs, outputs, and membership functions

The process of statistical analysis has been performed on both datasets, namely, Kaggle and PROMISE datasets. Following this, MSE and SE are passed to the fuzzy inference rule engine to create a ruleset for labeling the clusters. The proposed work has opted Mamdani inference engine due to its lower computation complexity. Figure 5 illustrates the sample, MSE, and SE as input and test effort classes predicted as low, moderate, and high classes.

3.5. Ruleset

The membership function described in Figure 5 is used to design 6 rule sets used for the creation and propagation of the model.

Table. 3 Rule Sets

MSE	SE	Test Effort Class
Low	Low	Low
Low	Moderate	Low
Moderate	Moderate	Moderate
High	Moderate	High
High	High	High
Moderate	High	High

The validated data from each class category is passed to the attribute set selection to the Cuckoo-Search algorithm. Cuckoo search is a Swarm Intelligence (SI) algorithm architecture that uses a Levy flight mechanism for the processing. The selected data from the Cuckoo Search Algorithm is passed to propagation-based learning behavior, viz. Feed Forward Back Propagation Neural Network (FFBPNN). The ordinal measures of FFBPNN are listed as follows in Table 4.

Table. 4 The propagation architecture

Propagation model	Levenberg
Propagation Type	Linear/Polyspace
Number of propagation layers	5-15 depending upon the size of the data
Selection method	Gradient-based
Validation type	Mean Squared Error (MSE)

FFBPNN incorporates regression-oriented learning for training and classification purposes. The categorization rule is formed so that the testing effort would be below if the classified ground truth label meets the test label. Else the testing effort would be high. As explained even earlier, there is also one more class called moderate testing effort. It occurs when the difference between the classified target class is either not too far from the original ground-truth value or not too close to the categorization of other classes. The FFBPNN performs three levels of regression, namely the R for training, R-for testing, R-for validation, and R-overall.

4. Results and Discussion

The results are evaluated based on the data division in the separation mechanism containing two separation windows, namely 70-30 ratio and 80-20 ratio. in the first segment, 70% of the data will be considered training data, and the rest of the 30% data will be considered test data. in a similar proceeding, 80% of the data would be considered the training data for the second window, and the rest of the 20% data will be considered the test data. The results have been evaluated based on the following parameters.

a) True positive rate: The total number of truly identified objects against supplied ground-truth value

b) False positive rate: It is the total number of false identified objects against supplied ground truth value.

c) Accuracy: The arithmetic means of the true positive rate for all listed data labels.

MATLAB 2016, containing a Neural Network toolbox, has been used to train the system. The training model is illustrated in Figure 6(a). for this instance, 11 features have been selected using GA, and each selected feature forms a feature set when combined with other selected features. 25 layers are passed, and the Levenberg architecture propagates the data through the validated gradient policy. The gradient is also bounded with epochs which are in total supplied to 1000 iterations. If the gradient is not attained within 1000 simulation iterations, the training will stop and produce the trained architecture. A total of 6 validation for each propagation layer is imposed on the training mechanism. It is necessary to overcome at least 3 validations. As it is clear from Figure 6(a), 5 out of 6 validations are executed successfully to produce the training architecture. Figure 6(b) represents the regression architecture illustrated earlier in the same section.

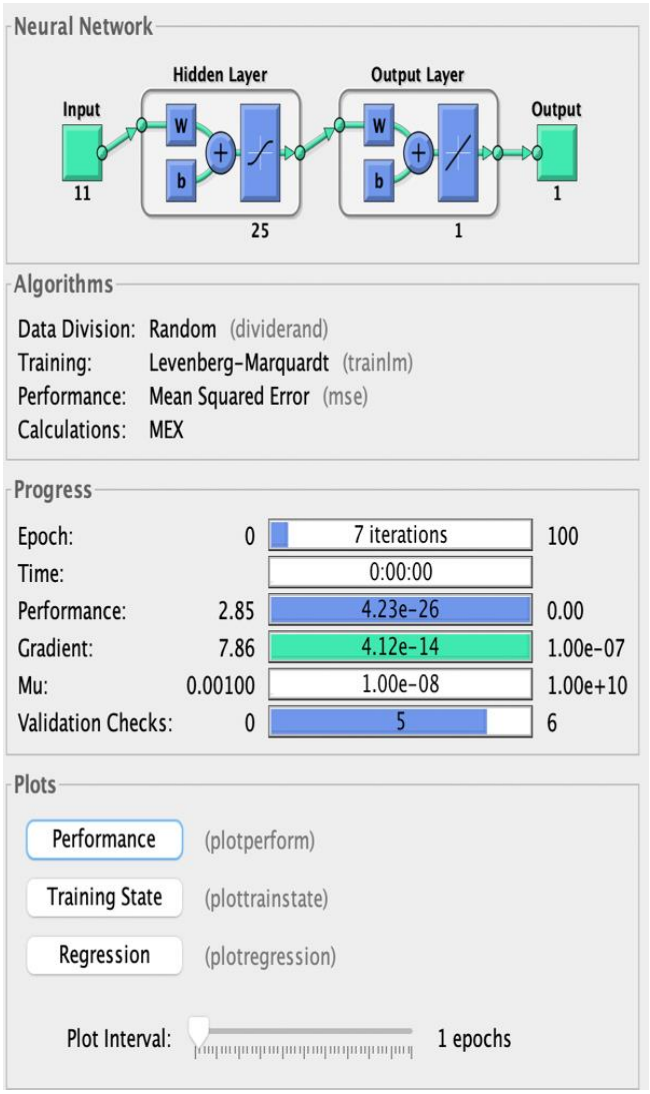

Fig. 6(a) The Neural Training

The value of R changes with the changes in the instance values. The formation of GA and Cuckoo Search also affected the overall evaluation of the regression values. The rest of the evaluation parameters are listed in Tables 5 and 6.

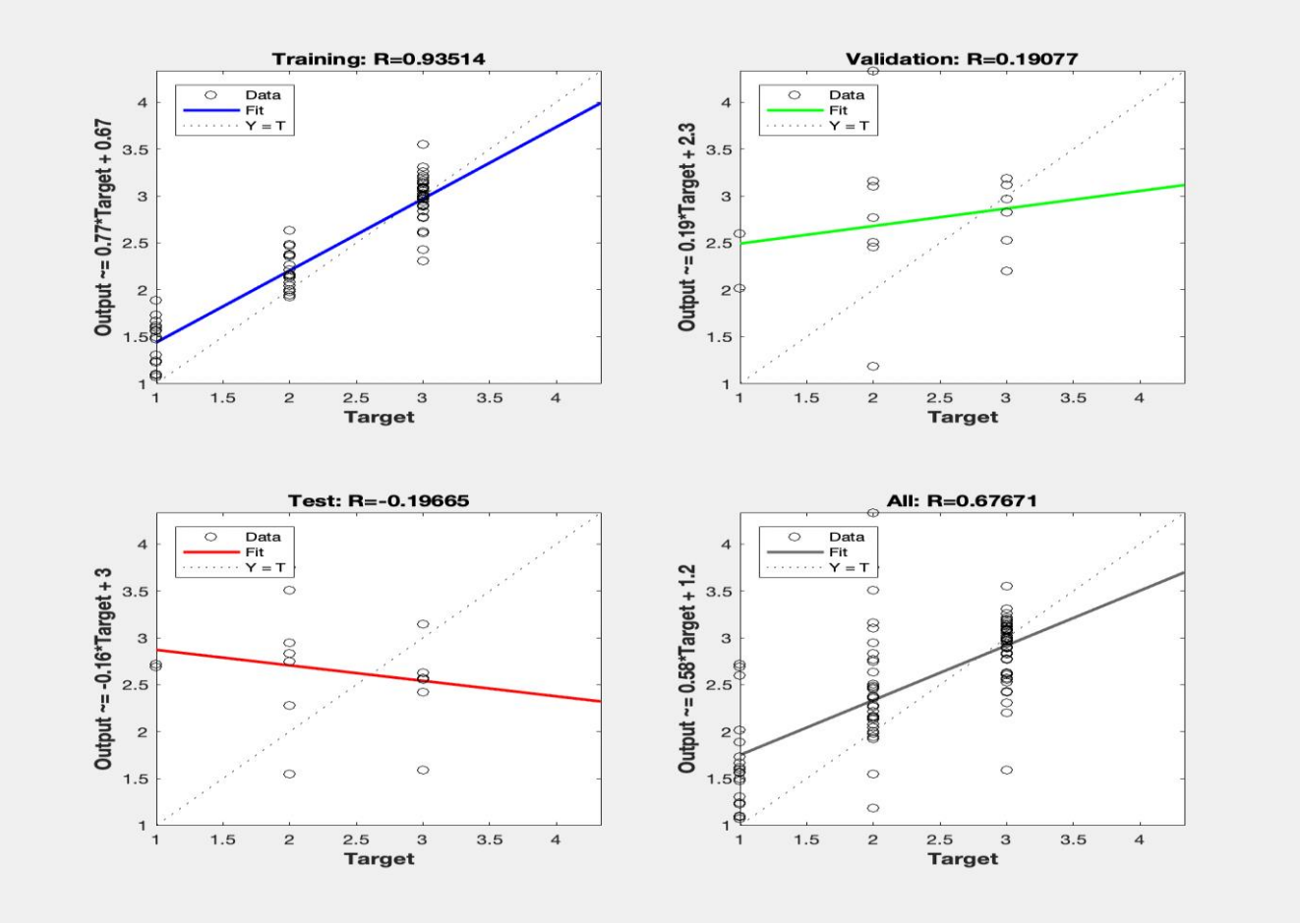

Fig. 6(b) The regression analysis

As evident from Tables 5 and 6, the learning rate for more projects is comparatively high compared to a low number of projects. to append the data, 90 projects have been listed from the PROMICE data set, whereas the rest of the projects have been listed from Kaggle. The maximum TPR for the proposed work is .95221695 for 1000 projects for 80-20 distribution. The same data is then passed to 70-30 distribution, and the results are slightly less but still high compared to the other state of art techniques.

Table. 5 The 70-30 distribution result

Total Number of Projects	TPR Proposed	TPR Without GA	TPR Varinder et al. [20]	FPR Proposed	FPR without GA	FPR Varinder et al. [20]
10	0.902212	0.76614117	0.81549276	0.097788	0.23385883	0.18450724
20	0.90734326	0.76927859	0.81849662	0.09265674	0.23072141	0.18150338
30	0.90968826	0.77037303	0.82236947	0.09031174	0.22962697	0.17763053
40	0.91056093	0.77336635	0.82285533	0.08943907	0.22663365	0.17714467
50	0.91250108	0.77603793	0.82606708	0.08749892	0.22396207	0.17393292
60	0.91624772	0.7786901	0.82897132	0.08375228	0.2213099	0.17102868
70	0.91867988	0.78046197	0.83141249	0.08132012	0.21953803	0.16858751

80	0.92160854	0.78351338	0.83312001	0.07839146	0.21648662	0.16687999
90	0.92345481	0.7865408	0.83560802	0.07654519	0.2134592	0.16439198
100	0.93221	0.81205268	0.86019063	0.06779	0.18794732	0.13980937
200	0.9346799	0.83702165	0.88581887	0.0653201	0.16297835	0.11418113
300	0.9399875	0.86041754	0.90217016	0.0600125	0.13958246	0.09782984
400	0.94112	0.88629011	0.91001	0.05888	0.11370989	0.08999
500	0.94567	0.8911	0.921114	0.05433	0.1089	0.078886
700	0.943321	0.8987999	0.922449	0.056679	0.1012001	0.077551
1000	0.94221	0.90111	0.92788	0.05779	0.09889	0.07212

Table 5 shows the TPR and FPR computed using the 70:30 ratio. The analysis results show that the TPR of the proposed model increases with an increase in the number of projects, and FPR decreases. It is seen that TPR and FPR for Varinder et al. [20] show 0.81 and 0.18 respectively for 20 projects and 0.76 and 0.23, using the GA. However, the proposed model shows a TPR of about 0.90 and an FPR of 0.09. Similarly, TPR for 90 projects increases to 0.92, and that of existing techniques and GA is 0.83 and 0.78, respectively. The overall TPR and FPR for 1000 projects are 0.94 and 0.05 using the proposed approach and 0.92 and 0.07, respectively, using the Varinder et al. [20]. Thus, the proposed technique shows better results in comparison to existing techniques.

Table. 6 The 80-20 distribution result

Total Number of Projects	TPR Proposed	TPR Without GA	TPR Varinder et al. [20]	FPR Proposed	FPR without GA	FPR Varinder [20]
10	0.90325093	0.76815558	0.81661566	0.09674907	0.23184442	0.18338434
20	0.90944227	0.77021591	0.81902469	0.09055773	0.22978409	0.18097531
30	0.91276608	0.77506919	0.82130693	0.08723392	0.22493081	0.17869307
40	0.91614222	0.77896306	0.82374494	0.08385778	0.22103694	0.17625506
50	0.91961234	0.78184222	0.82596965	0.08038766	0.21815778	0.17403035
60	0.92116638	0.78425299	0.82841232	0.07883362	0.21574701	0.17158768
70	0.92650087	0.78939136	0.83274246	0.07349913	0.21060864	0.16725754
80	0.9299947	0.79149856	0.83457478	0.0700053	0.20850144	0.16542522
90	0.93278673	0.79503425	0.83755162	0.06721327	0.20496575	0.16244838
100	0.93321592	0.82151814	0.86264824	0.06678408	0.17848186	0.13735176
200	0.93668071	0.8560542	0.88751949	0.06331929	0.1439458	0.11248051
300	0.94299042	0.86537892	0.90388464	0.05700958	0.13462108	0.09611536
400	0.94512834	0.89014087	0.91401356	0.05487166	0.10985913	0.08598644
500	0.95067387	0.89610207	0.92611913	0.04932613	0.10389793	0.07388087
700	0.95032782	0.9058004	0.92945552	0.04967218	0.0941996	0.07054448
1000	0.95221695	0.91111535	0.93788335	0.04778305	0.08888465	0.06211665

Table 6 shows the TPR and FPR computed using the 80:20 ratio. The analysis results show that the TPR of the proposed model increases with an increase in the number of projects, and FPR decreases. It is seen that TPR and FPR for Varinder et al. [20] show 0.86 and 0.17, respectively, for 100 projects and 0.82 and 0.066, using the GA. However, the proposed model shows a TPR of about 0.93 and an FPR of 0.06.

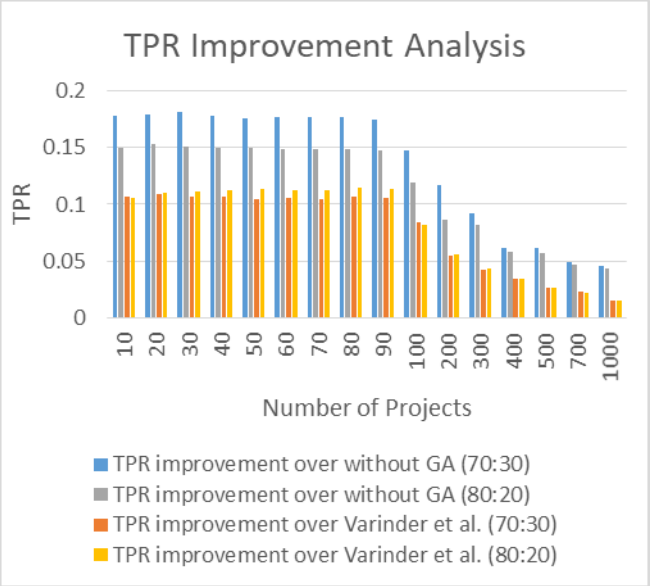

Fig. 7 TPR for different test effort class

Similarly, TPR for 400 projects increases to 0.94, and that of existing techniques and GA is 0.91 and 0.89, respectively. The overall TPR and FPR for 1000 projects are 0.95 and 0.04, respectively, using the proposed approach and 0.93 and 0.06 using the Varinder et al. [20]. Thus, the proposed technique shows better results in comparison to existing techniques.

Figure 7 shows the Improvement analysis for TPR of the proposed solution, which is compared when the Genetic Algorithm is not directly applied. The proposed algorithm architecture outcast the existing approach by 6-8%. This is due to the sophisticated architecture presented by the selection and propagation mechanisms. The propagation mechanism filters the data, and the cuckoo helps organize the categorized data. Looking at the overall accuracy, Fig. 9 represents the bar representation for the accuracies attained against each proposed class.

Figure 8 shows the improvement analysis for FPR using 70:30 distribution and 80:20 distributions. A comparison with GA has been presented for the effectiveness of the proposed approach. The improvement analysis graph shows that the proposed approach's average value is 0.07 and 0.17 without GA. The analysis results show that the proposed technique using 80:20 and 70:30 has been improved by 6-7% compared to without GA only. However, analysis results compared to Varinder et al. [20] are around 5 -6% using for 80:20 and 70:30 distribution.

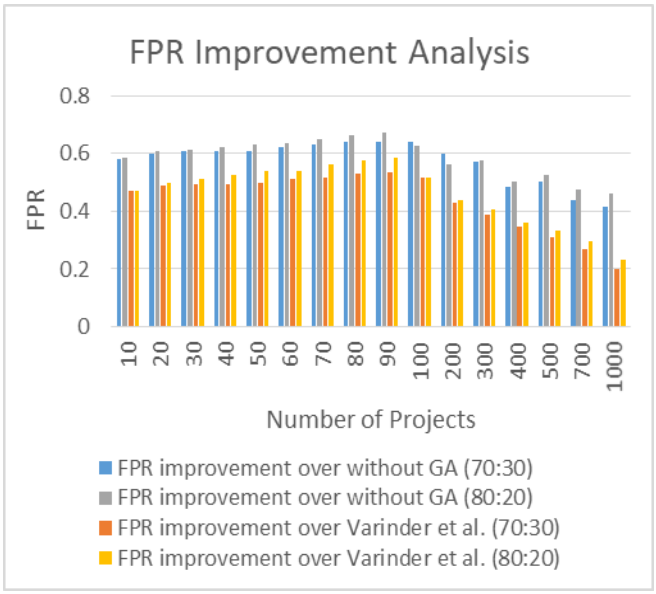

Fig. 8 FPR for different test effort class

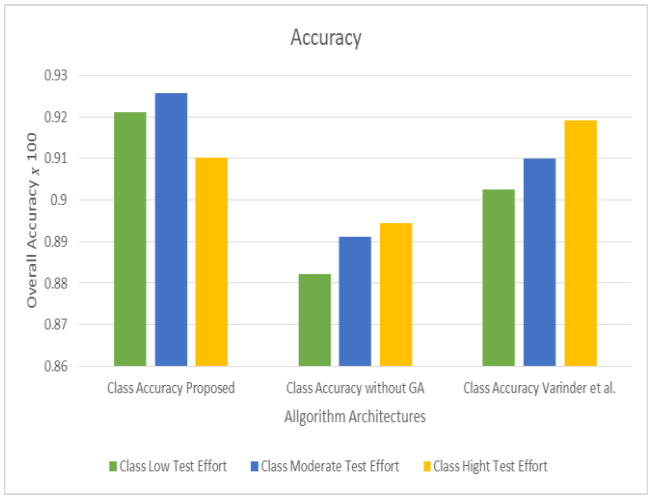

Fig. 9 Accuracy for different test effort class

As it is clear from Figure 9 and evident from the result in Table 5 and Table 6, the proposed algorithm architecture has a higher overall accuracy that lies between 91-93% based on the training and test data samples, whereas if it is passed without utilizing the selection algorithm, the accuracy in test effort calculation or estimation will lie under 88-89 % and utilizing Varinder et al. architecture, the accuracy increases as compared to without GA architecture but stays behind the proposed algorithm architecture. The accuracy has been evaluated for all the classes listed in the proposed methodology section.

5. Conclusion

Test effort estimation is a construct that can be evaluated using a lot of software-oriented factors. This article illustrates the evaluation of the test effort using other effort measurement architecture. The proposed algorithm architecture has utilized a Genetic Algorithm to process and select the dataset, followed by the attribute set selection method utilizing Cuckoo Search. The proposed algorithm has divided the entire data into three categories, and simulation behaviors have been tested against two data separation segments, namely 70-30 and 80-20. The separated classes were tested, and based on the rule set and propagation mechanism, the classes were classified. The proposed algorithm uses Levenberg's Feed Forward Propagation mechanism for the propagation mechanism, which is a gradient-based architecture. The proposed algorithm is evaluated for three parameters, namely the True Positive Rate, False Positive Rate, and Overall Accuracy. Due to the sophisticated selection architecture of the proposed algorithm, the proposed algorithm architecture attains maximum accuracy of 93%, whereas for a similar set of input parameters, the other state of art mechanisms lags by 3-5%, which could be stated as a significant improvement.

Conflicts of Interest

The authors declare that they have no conflict of interest.

Funding Statement

The author has received no funding.

References

[1] M. M. Asheeri, and M. Hammad, Machine Learning Models for Software Cost Estimation. in 2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ict): Ieee. (2019) 1-6.

[2] M. Dawson, and W. Christian, an Artificial Neural Network Approach to Software Testing Effort Estimation, Wit Transactions on Information and Communication Technologies. 20 (1970).

[3] J. Singh, and B. Sahoo, Software Effort Estimation With Different Artificial Neural Network (2011).

[4] P. R. Srivastava, A. Varshney, P. Nama, and X. S. Yang, Software Test Estimation: A Model Based on Cuckoo Search, International Journal of Bio-Inspired Computation, 4(5) (2012) 278-285.

[5] P. B. Nirpal, and K. V. Kale, Using Genetic Algorithm for Automated, Efficient Software Test Case Generation for Path Testing, International Journal of Advanced Networking and Applications, 2(6) (2011) 911-915.

[6] L. P. Kafle, an Empirical Study on Software Test Effort Estimation, International Journal of Soft Computing and Artificial Intelligence. 2(2) (2014) 109-125.

[7] C. S. Yadav and R. Singh, Tuning of Cocomo Ii Model Parameters for Estimating Software Development Effort Using Ga for Promise Project Data Set, International Journal of Computer Applications. 90(1) (2014) 99-123.

[8] N. Shivakumar, N. Balaji, and K. Ananthakumar, A Neuro-Fuzzy Algorithm to Compute Software Effort Estimation, Global Journal of Computer Science and Technology. 2(1) (2016) 232-243.

[9] R. Saljoughinejad, and V. Khatibi, A New Optimized Hybrid Model Based on Cocomo to Increase The Accuracy of Software Cost Estimation, Journal of Advances in Computer Engineering and Technology, 4(1) (2018) 27-40.

[10] S. K. Sehra, Y. S. Brar, N. Kaur, and S.S. Sehra, Software Effort Estimation Using Fahp and Weighted Kernel Lssvm Machine, Soft Computing. 23(21) (2019) 10881-10900.

[11] V. K. Attri, and J. S. Bal, an Advanced Mechanism for Software Size Estimation Using Combinational Artificial Intelligence, International Journal of Intelligent Engineering and Systems. 12(4) (2019) 32-41.

[12] A. Kumar, B. D. Patro, and B. K. Singh, Parameter Tuning for Software Effort Estimation Using Particle Swarm Optimization Algorithm, International Journal of Applied Engineering Research. 14(2) (2019) 139-144.

[13] S. Chhabra, and H. Singh, Optimizing Design Parameters of Fuzzy-Model-Based Cocomo Using Genetic Algorithms. International Journal of Information Technology, 12(4) (2020a) 1259-1269.

[14] S. Chhabra, and H. Singh. Optimizing The Design of A Fuzzy Model for Software Cost Estimation Using Particle Swarm Optimization Algorithm, International Journal of Computational Intelligence and Applications. 19(01) (2020b) 205-215.

[15] P. Suresh Kumar, and H. S. Behera, Estimating Software Effort Using Neural Network: an Experimental Investigation. in Computational Intelligence in Pattern Recognition. Springer, Singapore. (2020) 165-180

[16] W. Rhmann, B. Pandey, and G. Ansari, Software Effort Estimation Using an Ensemble of Hybrid Search-Based Algorithms Based on Metaheuristic Algorithms, Innovations in Systems and Software Engineering. (2021) 1-11.

[17] N. A. Zakaria, A. R. Ismail, N. Z. Abidin, N.H. M. Khalid, and A. Y. Ali, Optimization of Cocomo Model Using Particle Swarm Optimization, International Journal of Advances in Intelligent Informatics, 7(2) (2021) 177-187.

[18] A. Ardiansyah, R. Ferdinand, and A. E. Permanasari, Mucpso: A Modified Chaotic Particle Swarm Optimization With Uniform Initialization for Optimizing Software Effort Estimation. Applied Sciences. 12(3) (2022) 71-81.

[19] A. Kaushik, N. Singal, and M. Prasad Incorporating Whale Optimization Algorithm With Deep Belief Network for Software Development Effort Estimation, International Journal of System Assurance Engineering and Management, 2(1)(2022) 1-15.

[20] A. Ardiansyah, R. Ferdinand, and A. E. Permanasari, Mucpso: A Modified Chaotic Particle Swarm Optimization With Uniform Initialization for Optimizing Software Effort Estimation, Applied Sciences. 12(3) (2022) 72-81.

[21] D. B. Singh, D. A. Kumar, D. K. Sahni, D. Shree, A. Khushboo, K. Sirohi, D. Khurana. A Model to Measure Software Testing Effort Estimation in The Integrated Environment of Ernn, Bmo & Pso, International Journal of Engineering Trends and Technology. 69(8) 81-88.

[22] G. Somya, and A. Parashar, Machine Learning Application to Improve Cocomo Model Using Neural Networks, International Journal of Information Technology and Computer Science (Ijitcs) 3 (2018) 35-51.

[23] P. Pandey, and L. Ratnesh, Fuzzy Ahp Based Identification Model for Efficient Application Development, Journal of Intelligent & Fuzzy Systems. 38(3) (2020) 3359-3370.

[24] P. Singal, A. C. Kumari, and P. Sharma, Estimation of Software Development Effort: A Differential Evolution Approach. Procedia Computer Science 167(2020) 2643-2652.

[25] Kaggle's Online Data Source Available At Https://Www.Kaggle.Com/Sayedmohsin/Sqa-Dataset

[26] A. Kaushik, and N. Singal A Hybrid Model of Wavelet Neural Network and Metaheuristic Algorithm for Software Development Effort Estimation, International Journal of Information Technology. 3(1) (2019) 1-10.

[27] P. Singal, A. C. Kumari, and P. Sharma, Estimation of Software Development Effort: A Differential Evolution Approach, Procedia Computer Science, 167(8) (2020) 2643-2652.

[28] Promise Repository Is Available At Http://Promise.Site.Uottawa.Ca/Serepository/Datasets-Page.Html.