Performance Analysis of Software Test Effort Estimation using Genetic Algorithm and Neural Network

Vikas Chahar, Prof. Pradeep Kumar Bhatia
G. J. University of Science and Technology, Hisar, Haryana, India

***Abstract*—In present scenario, the software companies are frequently involving software test effort estimation to allocate the resources efficiently during the software development process. Different machine learning models are developed to estimate the total effort that would be required before the software product could be delivered. These computational models are used to use the past data to estimate the efforts. In the current studies, test effort estimation for software is predicted using the Genetic algorithm and Neural Network. The attributes are selected using the Genetic algorithm and similarity measure between the attribute values has been computed using the Cosine Similarity measure. The simulation experiments were done using the PROMISE and Kaggle repository and implementation was done using the MATLAB software. The performance metrics namely, precision, recall, and accuracy are computed to evaluate against the existing techniques. The accuracy of the proposed model is 91.3% and results are improved by 8.9% in comparison to existing technique and comparison has been made for superiority to predict the test effort for software development.**

***Keywords*—Test effort estimation; software testing; machine learning; computational intelligence; neural network**

I. Introduction

The success of a software project mainly depends on the determination of effort for software development [1]. The cost of the software can be computed by determining the efforts for the development of software. Software engineering is the study of techniques, quantifiable approach, software maintenance, and quantifiable approach during the development phase: application of engineering for software testing. The software testing plays a very significant role and accounts almost 50% for the total development in effort estimation.

Software testing allows the evaluation of attributes or system capability in determining the requirements to meet the desired results [2]. Software testing is mainly categorized as static and dynamic testing. In the former testing phase, testing has been done without executing the project and it is related to prevention of defects [3]. The documentation estimation is cheap and code assessment is provided in addition. Moreover, it also includes checklist, estimation of variety of errors, operated in the initial phase, and completion of 100% coverage of statement within less time. In the later testing phase, dynamic testing is very expensive, testing has been done during the execution of the project and it is related to fixing the defects [4]. The bugs estimation and assessment of bottleneck is provided whenever this phase is operated later or in the last phase of the project. Moreover, it also includes test cases, fixing the variety of errors, and completion of 50% coverage of statement. The primary motive of testing the software is to eliminate the bugs and improve the software security and other aspects such as performance, user satisfaction level, and experience. Furthermore, test effort estimation is necessary for the test process and plays a crucial role in the operation of software development life cycle. Software effort estimation allows the organization to provide or allocate the necessary resources accordingly. The best testing deal not only improves the overall quality but also enhances the customer satisfaction level. In a competitive market, there is a need to determine the highly reliable software effort estimates. In the software project development phase, the accurate estimation allows the success of the project [5]. Further, the cost of the software is estimated using the software effort required for the development of the software. In the literature, there were large number of techniques proposed to predict the software effort accurately. The estimation of software effort is helpful for the allocation of resources in a proper manner. The estimation of software effort in terms of month and day per person, duration of the project is very difficult to predict the project cost. It is crucial to negotiate with the customer by estimating the cost and effort in an accurate manner.

A. Computational Intelligence Techniques for Software Effort Estimation

Inaccurate prediction of software effort and cost usually results in huge financial loss and even in the failure of the project. However, there are number of techniques developed in the past such as expert judgement, machine learning techniques, fuzzy technique, and regression analysis [6]–[8] to minimize the instances of inaccurate prediction. Most of these techniques were based on the algorithmic models such as COCOMO and analogy-based estimation of effort techniques. The analogy techniques generally include the use of different characteristics such as size, interfaces, and effort of new project is estimated by determining the details of project of similar type. Furthermore, there are different techniques designed for different datasets as no single technique is applicable for all the datasets. In this process, programming language, development technique, programmer experience, tools etc. play a significant role in governing the software effort estimate. For instance, soft computing models are used to deal with computational problems and metaheuristic techniques are used to resolve the complex optimization issues [9], [10]. The evolution of neural network, fuzzy logic, support vector machine, optimization

technique, machine learning, chaotic theory, etc. fall under the category of computational intelligence models as shown in Fig. 1 [11].

Computational techniques proved fruitful results in predicting the attributes of software quality. In the past year, the computational models were developed and applied in software engineering to solve the different problems such as determining the prediction of change in software, prediction of stakeholder satisfaction, and estimation of reliability for component-based software projects. The estimation of test effort is generally performed by using the templates of Work Breakdown Structure (WBS) that fragments the project into sub-tasks. The analysis of each task during the testing phase allows the determination of defects and underlying errors. The project requirements are designed, and testing phase has been analyzed thoroughly to avoid any defect. This required a detailed overview of the project prototype and analyzing each task by coordinating with the stakeholders. This generally takes around 1.5-2 weeks to perform test effort estimation. However, employing the machine learning techniques for effort estimation eases the process in a fast manner and it is generally computed by determining the overall time to the total inputs given for the completion of the project. The proposed technique introduces the novel method by integrating the machine learning with genetic algorithm for the effort estimation of software testing. The contributions of the proposed work are summarized as follows: -

- Machine learning techniques are integrated for estimation of software effort.
- A novel combination of Genetic algorithm with Machine Learning is used to predict the software effort.
- Two datasets are employed to evaluate the computational strengths of the designed work.

The rest of the paper is organized as Section II that illustrates the related work, Section III that describes the research methodology including the different datasets and discusses the technique used for software test effort estimation. Section IV illustrates the results and finally concluded in Section V.

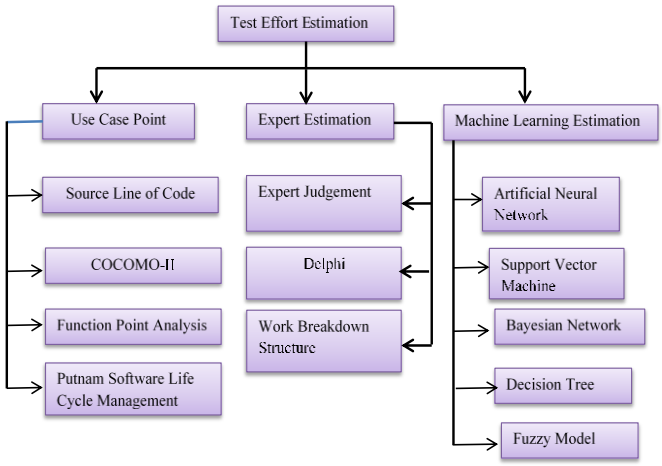

Fig. 1. Test Effort Estimation Models.

II. Related Work

The estimation of effort involved in the software development is a crucial activity for monitoring the project cost, time, and quality as well as for the software development life cycle. As a result, proper estimating was crucial to the success of projects and to lowering risks. Software effort estimation has drawn a lot of research interest recently and has become a problem for the software industry. Many academics and industry professionals have suggested statistical and machine learning-based approaches for estimating software effort over the past 20 years.

Saljoughinejad and Khatibi, 2018 had taken advantage of three metaheuristic techniques to enhance the effort accuracy estimations associated with the COCOMO model. The concept of metaheuristics mainly focused on the detailed analysis of the involved cost derivers involved in the effort estimation. The study had reflected that the integration of techniques such as PSO, Invasive Weed Optimization and GA had significantly improved the accuracy measures associated with estimations. However, despite of better performance, the work was unable to meet the desired level due to instability issues [12].

Nassif et al. 2019 compared the three different fuzzy models to estimate the software effort. The authors designed the models and conduct the regression analysis to evaluate the performance of the proposed system. The evaluation of the proposed regression fuzzy logic was measured by measuring the criteria such as standardized accuracy, effect size, and relative error. The authors used the ISBSG dataset, and it was estimated that different projects have similar size with better productivity ratio. The mean for effort dataset 1 was 883.5 and effort dataset 4 was 706 with a standard deviation of about 1194 with a skewness of about 5.8. Further, Scott Knott test was performed to determine the validity and best performance achieved using the Suzzeno fuzzy model [7].

Ghatasheh et al. 2019, evolutionary algorithms called the Firefly Algorithm was presented for optimising the parameters of three COCOMO-based models. The authors used the NASA dataset in which 30% data was tested and 60% data was trained to acquire the adequate objectives. The proposed model and two additional models that was suggested in the literature as expansions of the fundamental COCOMO model. The evaluation results using the Firefly algorithm show better accuracy. The limitation of the study was instability issues, prediction model; dataset type was affected by size [13].

Chhabra and Singh 2020 had proposed integration of non-algorithmic modelling for software effort estimation based on soft computing approaches. In the process, they had integrated genetic algorithm followed by fuzzy logic and utilized the COCOMO dataset for the evaluation of the designed work architecture. It has been observed that due to improved selection owing to the GA fitness function a 25% reduction has been observed in Mean Magnitude of Relative Error. This high improvement was mainly due to increased stability of GA in optimizing the fuzzy model that improved the overall prediction accuracy for the effort estimation [14].

Öztürk et al. 2021, a feed forward DNN algorithm (FFDNN) was put forth in this article. Finding hyperparameter was done using a binary-search-based technique in the algorithm. In the experiment using two performance parameters, FFDNN performs better than five comparative algorithms. The study's findings indicate that: 1) Using conventional techniques like grid and random search significantly lengthens tuning time. Instead, sophisticated parameter search techniques that was compatible with the structure of regression methods should be developed; 2) SEE performance was improved when the associated hyperparameter search technique was designed in accordance with the key principles of selected deep learning approach; and 3) Deep learning models outperform tree-based regression techniques like CART DE8 in terms of CPU time. The drawback of the study was that tuning time need to plan along with pruning of network [11].

Karimi and Gandomani 2021 This research introduces a new fuzzy inference technique and the differential evolution (DE) algorithm. To estimate software development labor, more precisely, this approach is capable of providing a more accurate estimate for software projects than earlier efforts using the COCOMO model. The suggested approach outperformed existing optimization algorithms derived from genetic, stochastic, conceptual, and Neuro-fuzzy technique, and could increase accuracy using the proposed technique up to 7%. The limitation of the study was assessment criteria and convergence rate, still a challenge for the accurate software effort estimation [15].

Zakaria et al. 2021 had integrated PSO as a swarm intelligence technique to optimize the existing COCOMO II. The optimized set was then fed to different machine learning techniques to evaluate their strength for the prediction of effort using NASA dataset. The machine learning techniques integrated were, Linear Regression (LR), SVM and Random Forest (RF). The simulation analysis had shown that SVM had outperformed the other machine learning techniques in terms of MMRE, accuracy and p-value computed for each of the implemented combination [16].

López-Martín 2022 proposed software testing effort estimation using the machine learning models. The authors investigate the effort of software testing using the datasets stored in the repository. The project selection was entirely based on the rating of data quality, development type, platform, programming language, sizing method, and level of resources for projects. Further, the authors investigate the performance of five machine learning models for software effort estimation using the COCOMO model. The prediction accuracy was computed for different ML techniques such as Neural Network, Decision Tree, Genetic algorithm, SVM and Case based reasoning. The limitation of the study was that software effort estimation depends upon the certain factors such as quality expectations, developer experience, tools and many other that are not sufficient to consider for accurate prediction of software effort estimation [6].

III. Research Methodology

The proposed methodology is divided into two parts in which first part Implementation using the Genetic algorithm has been done and Machine Learning model such as Neural Network toolbox in MATLAB was used for the classification of test effort estimation. The dataset used in this study for implementation are illustrated in the later sections.

A. Dataset

There are several free datasets that were used in the literature such as Kaggle, COCOMO NASA-I, COCOMO NASA-II, Kaggle, and PROMISE [17], [18]. Out of these, Kaggle and Promise datasets have been used in this study. These are further divided into five datasets namely KC1, PC3, PC4, MW1, and CM1 that support these datasets [19]. The PROMISE repository was used to extract the attributes. The dataset is an open source data set and is freely available online [20]. The dataset contains the attributes that are extracted after the operations performed through Object Oriented Programming Architecture (OOPA). As for example, RELY is an attribute that illustrates the reliability of a software and in the similar fashion, RES represents the reusability of the software component. Based on these attribute values, the overall computation effort is also provided. The dataset does not have independent attribute as they have been computed via OOPA. The data retrieval was estimated using the KC1 classes and further defect was analyzed. This study incorporates the Kaggle and PROMISE database datasets that includes different attributes such as KC1, CM1, MW1, PC3, and PC4. The datasets was stored and further assisted for implementation in the MATLAB software. Although, Kaggle and Promise data repository includes the different data, but extraction of revenant data is particularly important before the implementation. Therefore, Cosine similarity technique was applied to extract the data as per requirement for test effort estimation during the software development life cycle.

B. Genetic Algorithm for the Selection of Relevant Attributes

In this study, the relevant attributes have been selected using the Genetic Algorithm which is a heuristic technique employed to avoid the challenges of modelling and optimization techniques. The main features of the GA are to utilize the features of crossover operator and execute the operations to obtain the candidate solutions. The operation steps are graphically illustrated using Fig. 2. The Genetic Algorithm has been applied by considering the following steps: -

1) Start: The random population is generated considering the n-chromosomes for best solution of the problem. The random population is generated which is of n chromosomes.

2) Fitness: The fitness function f (a) is evaluated that corresponds to the chromosome (a) in the generated population.

3) New Population: The new population is initialized by repeating the following steps until the optimal solution is attained.

- Selection: The selection has been done by considering the fitness function of two chromosomes and then higher fitness leads to the selection of chromosomes with more possibility.
- Crossover: When the probability of crossover crosses then new offspring attained. If there is no crossover, then offspring is a duplicate of parent.

- Mutation: The mutation probability is attained when the new offspring mutates at each position in the population.
- Accepting: The new offspring has been placed with respect to new population.

4) Replace: The generated population is utilized considering the aim to execute the new solution.

5) Test: The best solution has been returned after the current population attained.

6) Loop: Return to second step.

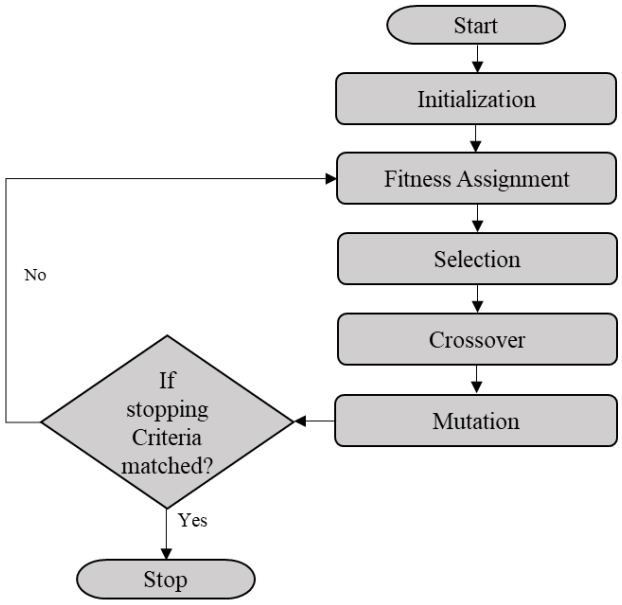

Fig. 2. Operation of GA.

After the operation of GA, further cosine similarity technique has been applied for the selection of similar attributes.

C. Cosine Similarity

Cosine similarity is a technique in which similarity of two documents computed to determine the correlation between the vectors. The information is represented in the vector form that makes the process easier to eliminate the irrelevant data. Furthermore, the angle between the vectors is determined as the cosine angles between the attributes. Cosine similarity is one of the most important similarity measures used for different applications such as clustering, effort estimation, etc. and also used to retrieve essential information. The cosine similarity between two attribute vectors $\overrightarrow{V_m}$ and $\overrightarrow{V_n}$ is given by;

$$Cos_{Sim}\left(\overrightarrow{V_m} \text{ and } \overrightarrow{V_n}\right) = \frac{\overrightarrow{V_m}.\overrightarrow{V_n}}{|\overrightarrow{V_m}|\times|\overrightarrow{V_n}|} \quad (1)$$

Here, $\overrightarrow{V_m}$ and $\overrightarrow{V_n}$ are the n number of dimensional vectors in each term set of $V = \{V_1, V_2, V_3 \dots \dots \dots \dots \dots \dots \dots \dots \dots . V_n\}$. Every dimension in the term set includes weight, which is positive, and therefore, the cosine similarity is positive and can be bounded between $\{0,1\}$.

A significant feature of cosine similarity is that it is the independent of its attribute for different set efforts in days or months. For example, integrating two effort values of different days require to obtain a novel attribute value, the cosine similarity between Att' *and* Att is 1, which indicates that the test effort attributes can be considered identical and can be stored for further processing.

ALGORITHM 2: Cosine Similarity for effort estimation

Input: $data_{files}$

Output: $Cosine\ Similarity_{dataset}$

1. $Function$: $Cosine\ Similarity_{dataset} = FunctionCosine\ Similarity_{dataset}(data_{records})$
2. $//Cosine\ Similarity_{dataset_{relation}} = [\ ];/$ $/Similarity\ Index\ calculation\ using\ empty\ array$
3. $// Sim_{count} = 0;//identifiedrelationsCount$
4. $Fors = 0\ todatarecords.count\ /$ $/ Totalno.ofdatarecords$
5. $Present_{dataset} = data_{record}(s)$;
6. $Forz = I + 1\ totaldatarecords.count// Nextseries$
7. $P = \left|Cosine\ Similarity_{dataset}(Present_{catalogue}) - \cos\left(datarecords(z)\right)\right|$;
8. $Cosine\ Similarity_{dataset}[Relation_{count}, 0] = present\_catalogue$;
9. $Cosine\ Similarity_{dataset}[Relation_{count}, 1] = datarecords(z)$; 10. $Cos_{catalogue_{relation}}[Dataset_{count}, 2] = P$; The similarity value
11. $Simulation_{count} = Sim_{count} + 1$; Count is incremented
12. $Endfor$;
13. $Endfor$;
14. $Endfunction$;

After the extraction of data for software test effort estimation, the data is processed to select the attribute values using the Genetic Algorithm and classification was done later using the Neural Network.

D. Effort Estimation using the Computational Intelligence Models

After the extraction of similar attributes, Neural Network has been applied that contain processing elements that are connected using some weights. It attempts to depict the biological nervous system as per both architectures including information processing logics. This network needs to train first by applying an appropriate learning algorithm for the prediction of weights which are interconnected. After training of weight test signals are classified. The neural network's class used basically for task of classification is called the multilayer perceptron network. The ordinal measures of Neural Networks are as follows (Table I).

TABLE I. ORDINAL MEASURES OF NEURAL NETWORKS

Propagation Architecture	Software test effort estimation
Neuron Count	5-25
Nature of propagation	Progressive
Propagation behaviour model	Levenberg
Root node validation	Mean Squared Error (MSE)
Validation parameters	a) Total number of epochs b) Gradient c) Count of fails in the validation
Cross validation	Linear Regression
Regression equation	$z = ax + b$ (eq hh) Where x is a multi-objective fitness function defined by sigmoid function of neural networks

The neural network propagation is designed using Neural Network toolbox of MATLAB and it is a propagation-based model and hence the number of hidden layers has been varied to check the performance of the network.

ALGORITHM 6: Test effort estimation using the Neural Network (NN)

Input: Optimized feature (T)
Output: Test Effort Estimation Results

1) Initialization of NN parameters

$E \rightarrow$ Simulation or Epochs for NN
$N \rightarrow$ Neurons Count
Performance Measure→ Accuracy, Precision, Recall, and F-measure
Techniques →Levenberg Marquardt
Data Division→ Random

2) For I = 1 → T

3) If (T matcheswith 1st feature category)

4) Group (1) = Features of training data according to the 1st category

5) Else if (T matcheswith 2nd feature category)

6) Group (2) = Features of training data according to the 2nd category

7) Else

8) Group (3) = Extra properties of training data

9) End-if

10) End-for

11) Initialize the NN using Training data and Group

12) Net = patternet $(T, Group, N)$

13) Set the training parameters and train the system

14) Net = Train (Net, Training data, Group)

Testing Phase:

15) Current Data = Feature of current efforts in dataset

16) Output = simulate (Net, Current Data)

17) If Output is valid)

18) $Test\ Effort\ Estimation \rightarrow Accurate$

19) Else

20) $Test\ Effort\ Estimation \rightarrow Inaccurate$

21) End-if

22) Return: Prediction as an output

Since the NN architecture used multiple hidden layers, therefore the input data is filtered many times and hence chances of providing better results are increased. In this research, the performance of NN is examined and cross-validation outcomes are evaluated.

IV. RESULTS AND DISCUSSION

The performance is evaluated by dividing the total dataset using the separation mechanism of training dataset to testing dataset ratio.

A. Statistical Analysis

The results have been computed using the 70:30, 80:20, and 90:10 ratio analysis. The results obtained using the implemented methodology in which four different performance metrics has been computed as illustrated below: -

$$Precision = \frac{True_{positive\ rate}}{True_{positive\ rate} + False_{positive\ rate}}$$

$$Recall = \frac{True_{positive\ rate}}{True_{positive\ rate} + False_{negative\ rate}}$$

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

The evaluation has been done to determine the superiority of the proposed approach and labelled dataset has been used which is pre-defined and specific to determine the required outcomes.

Table II shows the Recall, Precision, F-measure, and Accuracy computed using the 70:30 dataset distribution ratio. It is generalized that with increase in the number of projects the recall of the proposed model also gets improved. The proposed model shows a recall of about 0.90% for 80 projects. It is seen that Precision and Recall show 0.9% and 0.91% respectively for 20 projects and 0.90% and 90% for F-measure and Accuracy respectively using the GA and NN. The proposed results for 70:30 analyses are robust and improvised using the Genetic Algorithm in conjunction with NN.

Table III shows the analysis of performance metric computed using the 80:20 ratios. It is generalized that with increase in the number of projects the performance of the proposed model also gets enhanced. The proposed model shows a recall of about 0.93 for 300 projects and F-measure of about 0.92. It is seen that average Precision and Recall show 0.9% and 0.91% respectively and 0.91% and 91% for F-measure and Accuracy respectively using the GA and NN. The proposed

results for 80:20 analyses are robust and improvised using the Genetic Algorithm in conjunction with NN.

TABLE II. PERFORMANCE METRIC OF THE PROPOSED TECHNIQUE USING THE 70:30 RATIO ANALYSIS

Total Number of Projects	Recall	Precision	F-measure	Accuracy
10	0.852212	0.896543345	0.873815768	88.73637
20	0.8604666	0.897088945	0.878396221	88.9276
30	0.8687212	0.897634545	0.882941232	89.11883
40	0.8769758	0.898180145	0.887451329	89.31007
50	0.8852304	0.898725745	0.891927027	89.5013
60	0.893485	0.899271345	0.896368834	89.69253
70	0.9017396	0.899816945	0.900777247	89.88376
80	0.9099942	0.900362545	0.905152751	90.075
90	0.9182488	0.900908145	0.909495825	90.26623
100	0.9195034	0.901453745	0.910389117	90.45746
200	0.920758	0.907999345	0.914334166	90.64869
300	0.9220126	0.914544945	0.91826359	90.83992
400	0.9232672	0.921090545	0.922177588	91.03116
500	0.9245218	0.927636145	0.926076354	91.22239
700	0.9257764	0.934181745	0.92996008	91.41362
1000	0.927031	0.940727345	0.933828955	91.60485

TABLE III. PERFORMANCE METRIC OF THE PROPOSED TECHNIQUE USING THE 80:20 RATIO ANALYSIS

Total Number of Projects	Recall Proposed	Precision Proposed	F-measure	Accuracy
10	0.862212	0.899543	0.880482	88.93637
20	0.870467	0.900089	0.88503	89.1276
30	0.878721	0.900635	0.889543	89.31883
40	0.886976	0.90118	0.894022	89.51007
50	0.89523	0.901726	0.898466	89.7013
60	0.903485	0.902271	0.902878	89.89253
70	0.91174	0.902817	0.907256	90.08376
80	0.919994	0.903363	0.911603	90.275
90	0.928249	0.903908	0.915917	90.46623
100	0.929503	0.904454	0.916807	90.65746
200	0.930758	0.910999	0.920773	90.84869
300	0.932013	0.917545	0.924722	91.03992
400	0.933267	0.924091	0.928656	91.23116
500	0.934522	0.930636	0.932575	91.42239
700	0.935776	0.937182	0.936479	91.61362
1000	0.937031	0.943727	0.940367	91.80485

TABLE IV. PERFORMANCE METRIC OF THE PROPOSED TECHNIQUE USING THE 90:10 RATIO ANALYSIS

Number of Projects	Recall Proposed	Precision Proposed	F-measure	Accuracy
10	0.875412	0.909543	0.892151	89.93637
20	0.883667	0.910089	0.896683	90.1276
30	0.891921	0.910635	0.901181	90.31883
40	0.900176	0.91118	0.905645	90.51007
50	0.90843	0.911726	0.910075	90.7013
60	0.916685	0.912271	0.914473	90.89253
70	0.92494	0.912817	0.918838	91.08376
80	0.933194	0.913363	0.923172	91.275
90	0.941449	0.913908	0.927474	91.46623
100	0.942703	0.914454	0.928364	91.65746
200	0.943958	0.920999	0.932337	91.84869
300	0.945213	0.927545	0.936295	92.03992
400	0.946467	0.934091	0.940238	92.23116
500	0.947722	0.940636	0.944166	92.42239
700	0.948976	0.947182	0.948078	92.61362
1000	0.950231	0.953727	0.951976	92.80485

Table IV shows the analysis of performance metric computed using the 90:10 ratio. It is generalized that with increase in the number of projects the performance of the proposed model also gets improved. The proposed model shows a recall of about 0.94 for 300 projects precision is 0.92 with F-measure of about 0.94. It is seen that average Precision and Recall show 0.92% and 0.91% respectively and 0.91% and 91.3% for F-measure and Accuracy respectively using the GA and NN. The proposed results for 90:10 analysis are robust and improvised using the Genetic Algorithm in conjunction with NN.

Table V shows the comparison of recall analysis with the existing techniques. It is seen that recall for Attri et al. 2019 and without GA show 0.81 and 0.77 respectively for 20 projects. The proposed model exhibited a recall of 0.94 when analysed for 200 projects. Similarly, recall for 1000 projects increases to 0.95 and using Attri et al. work and GA is 0.93 and 0.82 respectively. The overall recall using the proposed approach is 0.92 and 0.86 using the Attri et al. 2019. Thus, the proposed outperformed the existing techniques due to the use of Genetic algorithm and Neural Network.

TABLE V. COMPARATIVE ANALYSIS OF RECALL AGAINST ATTRI ET AL. WORK

Number of Projects	Recall Proposed	Recall Without GA	Recall Attri et al. 2019
10	0.875412	0.77543345	0.81661566
20	0.8836666	0.77597905	0.81902469
30	0.8919212	0.77652465	0.82130693
40	0.9001758	0.78707025	0.82374494
50	0.9084304	0.78761585	0.82596965
60	0.916685	0.78816145	0.82841232
70	0.9249396	0.78870705	0.83274246
80	0.9331942	0.78925265	0.83457478
90	0.9414488	0.78979825	0.83755162
100	0.9427034	0.79034385	0.86264824
200	0.943958	0.79688945	0.88751949
300	0.9452126	0.80343505	0.90388464
400	0.9464672	0.80998065	0.91401356
500	0.9477218	0.81652625	0.92611913
700	0.9489764	0.82307185	0.92945552
1000	0.950231	0.82961745	0.93788335

TABLE VI. COMPARATIVE ANALYSIS OF PRECISION AGAINST ATTRI ET AL. WORK

Number of Projects	Precision Proposed	Precision without GA	Precision Attri et al. 2019
10	0.909543	0.824474	0.8453636
20	0.910089	0.8320093	0.8518202
30	0.910635	0.8395446	0.8582768
40	0.91118	0.8470799	0.8647334
50	0.911726	0.8546152	0.87119
60	0.912271	0.8621505	0.8776466
70	0.912817	0.8696858	0.8841032
80	0.913363	0.8772211	0.8905598
90	0.913908	0.8847564	0.8970164
100	0.914454	0.8922917	0.897473
200	0.920999	0.892827	0.9039296
300	0.927545	0.9003623	0.9103862
400	0.934091	0.9078976	0.9168428
500	0.940636	0.9154329	0.9232994
700	0.947182	0.9229682	0.929756
1000	0.953727	0.9305035	0.9362126

Table VI shows the comparison of precision analysis with the existing techniques. The precision for Attri et al. 2019 and without GA 0.86 and 0.84 for 40 projects. Similarly, precision for 1000 projects the recall value increases to 0.95 for proposed work and Atri et al. work and GA is 0.93. The overall precision using the proposed estimation model is 0.92 and using Attri et al. is 0.89. Thus, an improved performance is exhibited by the proposed work using Genetic algorithm and Neural Network.

TABLE VII. COMPARATIVE ANALYSIS OF F-MEASURE AGAINST ATTRI ET AL. WORK

Number of Projects	F-measure Proposed	F-measure without GA	F-measure Attri et al. 2019
10	0.892151182	0.799202127	0.830740998
20	0.896683196	0.80301799	0.835100588
30	0.901180958	0.806805868	0.839384987
40	0.905644474	0.815973231	0.843741665
50	0.910075216	0.819748813	0.847977383
60	0.914472674	0.823497386	0.852319047
70	0.918838317	0.827219343	0.857654582
80	0.923172111	0.830915072	0.861658862
90	0.927473993	0.834584952	0.866264719
100	0.928363848	0.838229357	0.879716109
200	0.932337179	0.842134687	0.895649385
300	0.936295462	0.84914163	0.907123771
400	0.940238375	0.856148552	0.915425994
500	0.944165606	0.863155454	0.924707115
700	0.948078351	0.870162335	0.929605736
1000	0.95197579	0.877169198	0.93704723

Table VII shows the comparison of F-measure analysis with the existing techniques. The analysis results show that there is an increase in F-measure with increase in project count. It is seen that F-measure for Attri et al. 2019 and without GA is 0.87 and 0.83 for 100 projects. Further, it is observed that the F-measure for 1000 projects shows a rise and increases to 0.95 and Attri et al. and GA is 0.87. The overall average F-measure using the proposed approach is 0.92 and 0.88 using the Attri et al. 2019 which shows that the proposed work outperformed the Attri et al work.

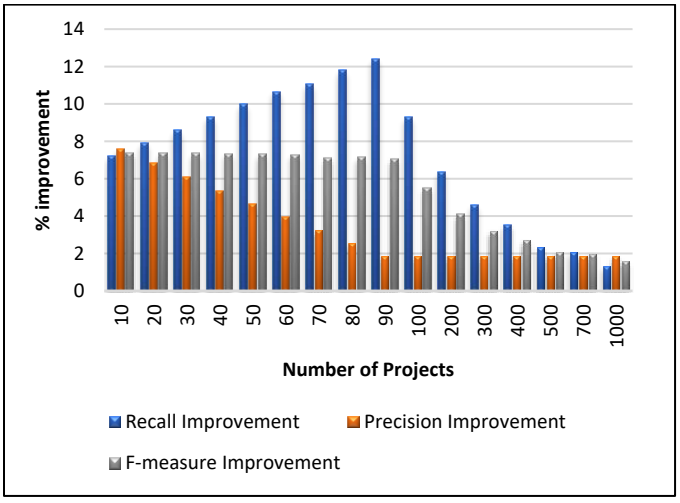

Fig. 3. Improvement Analysis of the Proposed Work over Attri et al. Work.

The observed performance in terms of precision, recall and f-measure values of both proposed and the existing work of Attri et al. are further analyzed to identify the extent of improvement exhibited by the proposed work. The individual % improvement for each of the parameters is individually computed and plotted in Fig. 3 for graphical illustration. It is concluded that despite of the variable % improvement observed

in each of the case, the overall analysis depicts the outperformance of the proposed work.

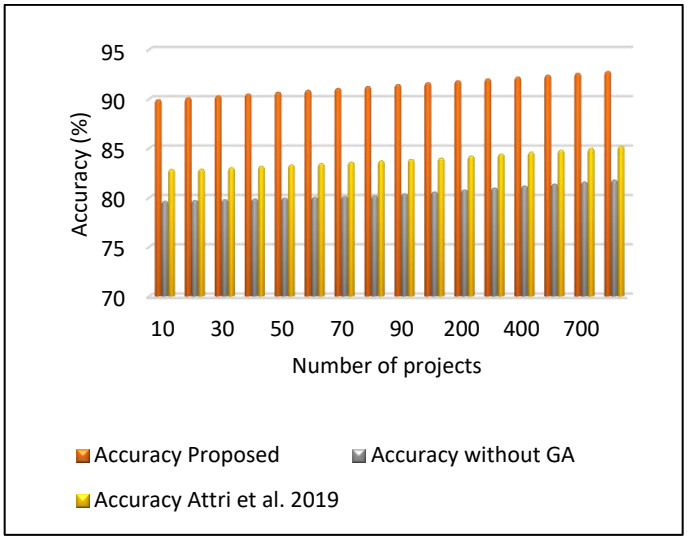

Fig. 4. Comparative Analysis for Accuracy.

Fig. 4 shows the comparison analysis for Accuracy analysis using the proposed and existing techniques. The analysis results show that there is an increase in Accuracy with increase in project count. It is seen that average Accuracy for Attri et al. 2019 and without GA is 84% and 81%. However, the proposed model shows Accuracy of about 91.3. The proposed technique has been improved by 8.9% in comparison to without GA and Attri et al. 2019. Thus, the proposed outperforms the existing work due to the integration of Genetic algorithm and Neural Network.

V. Conclusion

In the present work, machine learning based algorithms such as Neural Network, Genetic Algorithm and their attributes selection have been analyzed for the prediction of software effort. Software testing allows the evaluation of attributes or system capability in determining the requirements to meet the desired results. The primary motive of testing the software is to eliminate the bugs and improve the software security and other aspects such as performance, user satisfaction level, and experience. The study is based on the development of computational intelligence models to deal with the different complex problems. The implementation using the PROMISE and Kaggle dataset has been done and machine learning models such as Genetic algorithm and Neural Network used for implementation. The results of the proposed technique are promising. The accuracy of the proposed model is 91.3% and results are improved by 8.9% in comparison to existing technique. In future, an attempt has been made to improve the accuracy using the other computational techniques such as Fuzzy logic.

References

[1] A. Saeed, W. H. Butt, F. Kazmi, and M. Arif, "Survey of software development effort estimation techniques," in Proceedings of the 2018 7th International Conference on software and computer applications, 2018, pp. 82–86.

[2] V. K. Attri and J. Singh Bal, "An Advanced Mechanism for Software Size Estimation Using Combinational Artificial Intelligence," Int. J. Intell. Eng. Syst., vol. 12, no. 4, 2019, doi: 10.22266/ijies2019.0831.24.

[3] W. Rhmann et al., "Survey on software defect prediction techniques," Int. J. Appl. Sci. Eng., vol. 17, no. 4, pp. 331–344, 2019.

[4] C. L. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," in 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), 2020, pp. 728–733.

[5] Y. Mahmood, N. Kama, A. Azmi, A. S. Khan, and M. Ali, "Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation," Softw. Pract. Exp., vol. 52, no. 1, pp. 39–65, 2022.

[6] C. López-Mart\'\in, "Machine learning techniques for software testing effort prediction," Softw. Qual. J., vol. 30, no. 1, pp. 65–100, 2022.

[7] A. B. Nassif, M. Azzeh, A. Idri, and A. Abran, "Software development effort estimation using regression fuzzy models," Comput. Intell. Neurosci., vol. 2019, 2019, doi: 10.1155/2019/8367214.

[8] A. Banimustafa, "Predicting Software Effort Estimation Using Machine Learning Techniques," 2018 8th Int. Conf. Comput. Sci. Inf. Technol. CSIT 2018, pp. 249–256, Oct. 2018, doi: 10.1109/CSIT.2018.8486222.

[9] K. Dutta, V. Gupta, and V. S. Dave, "Analysis and comparison of neural network models for software development effort estimation," in Research Anthology on Agile Software, Software Development, and Testing, IGI Global, 2022, pp. 165–193.

[10] W. Rhmann, B. Pandey, and G. A. Ansari, "Software effort estimation using ensemble of hybrid search-based algorithms based on metaheuristic algorithms," Innov. Syst. Softw. Eng., vol. 18, no. 2, pp. 309–319, 2022.

[11] M. M. Öztürk, "A tuned feed-forward deep neural network algorithm for effort estimation," J. Exp. \& Theor. Artif. Intell., vol. 34, no. 2, pp. 235–259, 2022.

[12] R. Saljoughinejad and V. Khatibi, "A new optimized hybrid model based on COCOMO to increase the accuracy of software cost estimation," J. Adv. Comput. Eng. Technol., vol. 4, no. 1, pp. 27–40, 2018.

[13] N. Ghatasheh, H. Faris, I. Aljarah, and R. M. H. Al-Sayyed, "Optimizing software effort estimation models using firefly algorithm," arXiv Prepr. arXiv1903.02079, 2019.

[14] S. Chhabra and H. Singh, "Optimizing design parameters of fuzzy model based COCOMO using genetic algorithms," Int. J. Inf. Technol., vol. 12, no. 4, pp. 1259–1269, 2020.

[15] A. Karimi and T. J. Gandomani, "Software development effort estimation modeling using a combination of fuzzy-neural network and differential evolution algorithm," Int. J. Electr. Comput. Eng., vol. 11, no. 1, p. 707, 2021.

[16] 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," Int. J. Adv. Intell. Informatics, vol. 7, no. 2, pp. 177–187, 2021.

[17] "Software Quality Attributes Dataset | Kaggle." https://www.kaggle.com/datasets/sayedmohsin/sqa-dataset (accessed Aug. 09, 2022).

[18] A. Kaushik and N. Singal, "A hybrid model of wavelet neural network and metaheuristic algorithm for software development effort estimation," Int. J. Inf. Technol., pp. 1–10, 2019.

[19] P. Singal, A. C. Kumari, and P. Sharma, "Estimation of software development effort: A Differential Evolution Approach," Procedia Comput. Sci., vol. 167, pp. 2643–2652, 2020.

[20] "PROMISE DATASETS ENGINEERING REPOSITORY." http://promise.site.uottawa.ca/SERepository/datasets-page.html (accessed Oct. 21, 2022).