

Ensembling of Deep Learning Models to Evaluate Software Effort Estimation

Anup Narendra Kadu¹ and Nisha Wandile Kimmatkar²

¹⁻²Department of Computer Engineering, JSPM's Rajashri Shahu College of Engineering, Tathawade, Pune-33, India
Email: anupkadu1306@gmail.com, nishakimmatkar@gmail.com

Abstract—In the software industry, software effort estimation is an essential activity since it has the potential to dramatically improve the quality of the parameters that the team is working with. The failure of a company to pay attention to crucial aspects during the process of developing and delivering software frequently results in major financial losses for the company. Some of the most conventional methods for design and analysis, such as the COCOMO model and others, provide an estimate of the costs and the amount of effort involved in an abstract manner. All of the requirements for the project management process will not be satisfied by this method. In response to this, a number of authors have proposed a methodology for estimating the amount of effort required to develop software that makes use of artificial intelligence and machine learning. A large number of models are based on a single neural network model approach, which is not always sufficient to produce satisfactory outcomes. Consequently, the assembly of many deep learning models in order to improve the estimation of the amount of work required by software is a suitable choice. In addition, a few of the authors go into further information about this topic by presenting a multitude of innovative concepts that, in the end, open up the possibilities of the software effort estimating process. As a result, the system is proposed to estimate the software efforts by employing the Support vector machine to classify the data. On the basis of this classification protocol, LSTM and ANN Neural network are implemented. The values of the predictions that were obtained are made available to the hybrid model so that it can make a decision on the estimation of the software efforts.

Index Terms— Deep learning, Software effort estimation, Support vector machine, Artificial neural network, Long –short term memory, Hybrid model.

I. INTRODUCTION

Estimating the amount of work that needs to be done is an essential component of project management. It plays a vital part in establishing schedules that are attainable and in effectively allocating resources. The process entails making an estimate of the amount of time and effort that will be necessary to finish a specific project or endeavour. An effort estimation is a procedure that involves making a prediction about the amount of effort that will be necessary to develop or maintain a software application. Traditionally, this effort is quantified in terms of the number of hours that a person puts in, or the amount of money that is required to pay for this job. During the early phases of the software development life cycle, effort estimation is utilized to assist in the process of drafting project plans and budgets. When a project manager or product owner follows this technique, they are able to accurately anticipate costs and allocate resources in accordance with those predictions. In light of the exponential expansion that has occurred in the software industry, it has been recognized that the

process of developing software projects is distinct from the process that is associated with traditional software development. This is primarily due to the increased number of software that has been introduced as a result of the invention of cloud computing technologies. Continuous integration and continuous development (CD & CI) are terms that describe the capabilities offered by cloud computing technologies.

The process of automatically and often integrating code changes into a shared source code repository is referred to as continuous integration (CI), which is an acronym for the activity. CD, which stands for continuous delivery and/or deployment, is a process that consists of two parts: the integration of code changes, testing of those changes, and delivery of those changes. When it comes to automatic production deployment, continuous delivery falls short of automatic deployment, whereas continuous deployment is responsible for automatically releasing updates into the production environment.

When it comes to modern application development, the aim is to enable multiple developers to collaborate on different features of the same app simultaneously. Nevertheless, when an organization is structured to consolidate all branches of source code on a designated day (referred to as "merge day"), the resulting task can be laborious, requiring manual effort and consuming a significant amount of time. That's because when a developer works independently and makes changes to an application, there is a possibility of conflicts with concurrent changes made by other developers. This issue may get more complicated still if every developer has personalized their own local integrated development environment (IDE), instead of the team reaching a consensus on a cloud-based IDE. CI is a solution that addresses the challenge of managing multiple branches of an application in development simultaneously, minimizing the risk of conflicts.

Effort estimation is a crucial exercise in the software industry, as it can significantly enhance the team's parameter quality. Often, companies suffer significant losses due to the neglect of important parameters during the software development and delivery process. There are traditional design and analysis models, such as the COCOMO model and others, that estimate the cost and efforts in an abstract manner. This will not meet the requirements for the project management process. As an answer to this, many authors propose a model of software effort estimation using artificial intelligence and machine learning. Many models work on a single method of neural network models, which is sometimes not enough to provide good results. Hence, the ensembling of multi-deep learning models to enhance the software effort estimation is a good choice. Some of the authors also discuss this in detail by introducing many novel ideas that eventually unleash the possibilities of the software effort estimation process.

[1] Zakrani Abdelali et al. examine the recent surge in research focused on software development effort estimation (SDEE) utilizing machine learning (ML) methodologies. The objective of these studies has been to enhance the precision of estimates and gain insights into the methodologies employed for generating such estimates. Among the various machine learning techniques, decision tree-based models have garnered significant scholarly interest due to their capacity for generalization and comprehensibility. Nevertheless, there is a scarcity of research that has examined the application of random forest (RF) in the assessment of software labor. This work presents the design and empirical optimization of a Random Forest (RF) model through the manipulation of its major parameters. A comparison is made between the performance of the Random Forest (RF) and the classical regression tree (RT). A 30% hold-out validation approach was employed to conduct the evaluation, utilizing three datasets: ISBSG R8, Tukutuku, and COCOMO. In order to determine the most precise methods, the author employed three well-established accuracy metrics: Pred (0.25), MMRE, and MdMRE.

[2] Ricardo Britto et al. put forward Two prominent developments in the software industry of the 21st century are Agile Software Development (ASD) and Global Software Development (GSD). Several studies in the literature document the implementation of agile methods or the adoption of GSD by software businesses. The role of effort estimating in software project management is of significant importance. In the context of a GSD (Global Software Development) environment, firms employ the agile technique to conduct effort estimation. This paper presents a study that combines the results of two Systematic Literature Reviews (SLRs) on effort estimation. One SLR focuses on the ASD context, while the other examines the GSD context. The aim is to provide an overview of the current state of effort estimation in agile global software development (ASD).

This research study's Part 2 entails a comprehensive analysis of previous works as literature review. Part 3 explores the proposed works steps, whereas part 4 identifying study's findings. Section 5 offers opportunities for further improvements as well as it concludes the research paper.

II. LITERATURE SURVEY

[3] The bi-objective software project effort estimation algorithm was introduced and analyzed by Federica Sarro et al. The algorithm's main innovation resides in its integration of confidence intervals as a means to direct a

multi-objective evolutionary trajectory. The findings of the author demonstrate that the newly developed algorithm exhibits superior performance compared to existing methods, hence aligning it with the established benchmarks for industrial human-expert-based best practices in effort estimating. The findings of the author also offer support for the notion that the multi-objective aspect of the author's strategy is responsible for the notable enhancement in performance. In addition to the intrinsically appealing enhancements in performance, the author posits that developers and their managers may also find the inclusion of confidence intervals in effort estimations to be advantageous, since they serve to limit the level of uncertainty associated with the estimation.

[4] The objective of Nevena Rankovic et al. was to enhance the precision of software evaluation techniques. This article presents a comparative analysis of the effects of nonparametric model designs, namely two artificial Neural Network architectures, on software effort and costs estimation. The study focuses on comparing these architectures with classic parametric models, such as COCOMO2000. The comparison was conducted using publicly available datasets, namely COCOMO81, COCOMO2000, and Kemerer databases. There is a correlation between the performance metrics and the planning criteria in the decision-making process. This relationship could potentially be characterized by weights. The weights assigned to the scheduling criteria indicate their respective levels of significance. The operational policy is delineated by the factors and their corresponding weights. In this empirical investigation, a backpropagation Neural Network was selected to ascertain the relationship between global performance metrics pertaining to operational policies and the scheduling of work orders.

[5] Swarnima Singh Gautam et al. conducted research on many aspects to enhance software effort estimations, with a specific emphasis on improving dataset quality. This study focuses on examining the efficacy of outlier removal in enhancing the estimation performance of five machine learning (ML) techniques, namely Support Vector Regression, Random Forest, Ridge Regression, K-Nearest Neighbor, and Gradient Boosting Machines, for the purpose of software development effort estimation (SDEE). The author presents a new method called Golden Section based Adaptive Discretization (GSAD) to determine the most suitable number of outliers for the SDEE dataset. The findings indicate the significance of removing an ideal amount of outliers in order to enhance estimations. Furthermore, a comparative analysis was conducted between the outcomes derived from the implementation of the GSAD technique and the IQR and Cooks' distance based outlier identification methods across four datasets, namely ISBSG Release 2021, UCP, NASA93, and China.

[6] Huynh Thai hoc et al. The objective of this project is to construct prediction models by employing an ensemble methodology. It is anticipated that this will enhance the efficiency of the model in comparison to the solo model. The regression methods employed in this study, including XGBoost, RF, and HGB, demonstrate strong predictive capabilities. To assure impartiality and optimize the selection of regression approaches within the technical ensemble, a mixture of regression models is employed. A generator is designated as the ultimate regressor, while the remaining generators are picked as the preceding generators. The blend of the generators is achieved by the stacking method.

Ho Le Thi Kim Nhung et al. The objective of this work is to enhance the accuracy of estimations by making modifications to the Author's OCF approach. The ExOCF approach, as proposed, draws inspiration from the potential of employing a conventional estimation procedure to address the issue of human error impact in the analysis of the UCM. Additionally, it aims to streamline the fundamental principles of the UCP that are inherent in the OCF method. The author employed Multi-Layer Regression (MLR) models using historical project data points in order to construct regression models and mitigate errors during the integration phase or recursion. The proposed methodology enhances the capacity of the OCF approach to estimate the amount of software and reduces the forecast error.

[8] In their study, Khan et al. examine the significance of effort estimation as a crucial activity in ensuring the successful delivery of solutions in software engineering projects. The paper's primary contributions to the existing body of work on software effort estimation can be categorized into two key aspects. This work investigates the utilization of meta-heuristic algorithms in order to establish a coherent and satisfactory parametric model for estimating software labor. Furthermore, this study introduces a Deep Neural Network (DNN) model that utilizes meta-heuristic algorithms to optimize Deep Learning (DL) architectures for software effort estimates. The aim is to explore the advantages of incorporating nature-inspired meta-heuristic algorithms in this context. This study utilizes the Grey Wolf Optimizer (GWO) and StrawBerry (SB) meta-heuristic algorithms to develop a sound and satisfactory parametric model for estimating software effort.

[9] No In their study, Mohammed Alsheikh et al. examine Planning the software project is one of the steps in the Software Development Life Cycle. One additional duty during the project planning phase involves the estimation of the software work. The process of software effort estimation involves the determination of the number of workers necessary for the successful completion of a software project. Numerous scholars have directed their

attention towards this domain with the aim of enhancing the accuracy of software effort estimation, employing a combination of algorithmic and non-algorithmic methodologies.

The Constructive Cost Model (COCOMO) is the most commonly employed approach. Nevertheless, the COCOMO model exhibits a constraint pertaining to the accuracy of software work estimation. Meta-heuristic algorithms are commonly favored in the context of parameter optimization due to their ability to yield solutions that are close to optimal while maintaining an acceptable cost. The objective of this study is to improve the accuracy of effort estimation by adjusting the coefficients of three COCOMO-based models. Additionally, the study aims to evaluate the effectiveness of Grey Wolf Optimization (GWO) in determining the optimal value of effort estimation. This will be achieved by incorporating other algorithms proposed by the author, such as Zebra Optimization (ZOA), Moth-Flame Optimization (MFO), Prairie Dog Optimization (PDO), and White Shark Optimization (WSO), using the NASA18 dataset.

[10] In their study, Sánchez-García et al. examine the significance of software testing (ST) as a crucial step within the software development life cycle (SDLC). They highlight that the work dedicated to ST is frequently quantified as a proportion of the whole SDLC effort. Regrettably, the literature indicates that the percentage of effort in ST varies between 10% and 60%. The majority of machine learning methods and metaheuristics in the literature have primarily focused on estimating the overall effort of the Software Development Life Cycle (SDLC), without specifically addressing any particular phase of the SDLC, such as testing. Hence, the present work aims to examine the utilization of the Software Testing Effort Prediction (STEP) technique, namely the Gradient Boosting (GB) machine learning regression model, which has been enhanced using Differential Evolution (DE). The prediction accuracy of the GB is compared to that achieved through the optimization of Particle Swarm Optimization (PSO) and Genetic Algorithms (GA).

[11] Petr Silhavy et al. This study examines the impact of the kernel function on the movement of weighted windows across seven distinct datasets. Tests have been conducted on window sizes of 25%, 50%, 75%, and 100%. The chosen % window size ensures repeatability across datasets of varying sizes. The Windows technique has been subjected to a comparative analysis with spectral clustering, wherein the number of clusters ranged from 1 to 4, with 1 representing a nonclustered alternative. In the regression model, the kernel function has been employed as a weighting function. A total of 192 models based on windows and 192 models based on clustering were examined.

[12] In their study, Mohammad Muhairat et al. examine the advantages of outsourcing software development work activities in software development projects, including the reduction of development costs and time. The effective management of the implementation of this plan is a crucial attribute, both in its own right and in relation to its consequences. Precise work estimating is essential for the success of software development projects, particularly in projects that are deployed globally. Hence, it is imperative to ascertain and examine the fundamental aspects that impact the precision of effort estimating techniques. This study aims to examine the effort estimating methods of COCOMO II, SLIM, and ISBSG.

[13] The findings of this benchmarking investigation, as described by P. Subitsha et al., provide partial confirmation of the conclusions obtained in prior studies. Basic and comprehensible methods such as Ordinary Least Squares (OLS) with logarithmic modification of characteristics and target exhibit comparable or superior performance compared to non-linear techniques. Furthermore, the Cocomo formal model demonstrated comparable performance to the ordinary least squares (OLS) model with log transformation when used to the Coc81 and Cocnasa datasets. The Cocomo model was used to collect these two data sets. Nevertheless, this model necessitates a distinct collection of characteristics and cannot be employed on data sets that do not adhere to this stipulation. While the absolute performance variations may be minimal, even a slight variation in estimation performance can lead to increased frequency and magnitude of project cost overruns in the context of software development. Therefore, even minor variations might hold significance in terms of cost and operational considerations. An additional finding suggests that the choice of an appropriate estimating technique can exert a substantial influence on the overall performance.

[14] In this study, Poonam Rijwani et al. emphasize the importance of accurately predicting software development costs. This is crucial for making informed management decisions and precisely estimating the amount of effort and time required for a project. This information is essential for project managers, system analysts, and developers. A variety of software cost estimation methodologies are accessible, encompassing algorithmic approaches, analogy-based estimation, expert judgment, top-down methodology, and bottom-up methodology. There is no inherent superiority or inferiority of any certain method over another. The selection of estimation methodologies for a particular project is contingent upon the project's environment. Based on the merits and drawbacks of the strategies, you can select some methods to employ. For established projects and project components, the author should employ the expert judgment method or analogy method, provided that

similarities can be identified. This approach is efficient and dependable in such situations. However, for larger and less familiar projects, it is advisable to utilize an algorithmic model such as COCOMO2.0, which is expected to be accessible in early 1997.

III. PROPOSED MODEL

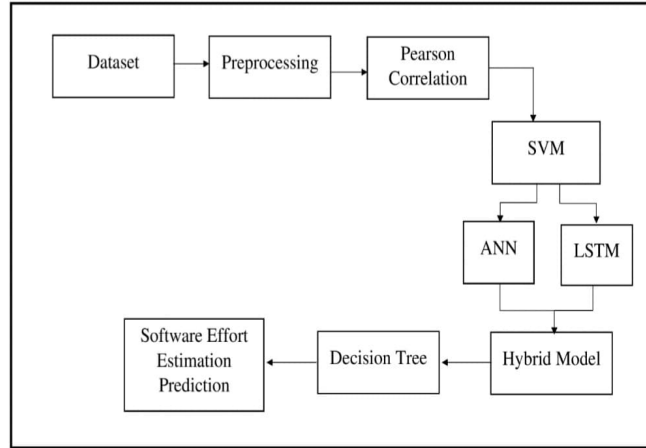


Figure 1: Overview of the proposed model

In Figure 1 above, the proposed system for software effort estimation is shown graphically. A series of actions are followed in order to deploy the model.

Phase 1: Dataset Collection and Preprocessing- The process of software effort estimation is evaluated by using the datasets from the URL: <https://www.kaggle.com/datasets/tamannashermin/software-effort-estimation-datasets>. This repository consists of archives of datasets from which china.csv is selected for the experiment. The dataset consists of 20 attributes such as id, ID, AFP, Input, Output, Enquiry, File, Interface, Added, Changed, Deleted, PDR_AFP, PDR_UFP, NPDR_AFP, NPDU_UFP, Resource, Dev.Type, Duration, N_effort and Effort.

Once the dataset is obtained, it usually goes through the pre-processing stage. As part of the pre-processing step, the attributes are collected in a two-dimensional list after reading the dataset from the specified path. The dataset has been successfully imported from the specified path using the Pandas library in Python. Utilizing a two-dimensional list data, the initial parameters of the attributes are estimated to describe the characteristics of the dataset, including the mean and standard deviation. Using this procedure, System is currently analyzing the dataset to collect attribute information for different data types, including strings and floats. Our goal is to calculate the entropy of the dataset's data types to obtain proper preprocessed list.

Phase 2: Pearson Correlation - The Pearson Correlation values are used to determine which traits have the lowest correlation. Pearson Correlation is used to establish the correlation between the attributes. As a result, a correlation matrix is produced, which may be helpful in determining the ideal attribute combination. This conclusion means that the qualities with lower correlation are disregarded and fresh correlation values are computed. The Pearson correlation is calculated using Equation 1, which is shown below.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \text{----- (1)}$$

Where,

x_i =values of x (independent) variable, y_i = values of y (Dependent) variable, $\bar{x}$ = mean of x variable values and $\bar{y}$ = mean of y variable values

The dataset is compiled into a double-dimensional list during the correlation step, following which a few superfluous columns are eliminated from the list. The resulting list is then passed to the next data segmentation stage, as the following step will demonstrate.

Phase 3: Support vector machine- Support vector machine is used to classify the attributes that were obtained during the past phase of preprocessing. For this purpose, the Sklearn library in Python has been used to classify the attributes for the SVM model. The X and Y lists are generated for the decided attributes to fit the model and identify their classification values. Based on these classification values, the attributes indicating “effort” is labeled as 1 for yes and 0 for no. This is based on the threshold effort values that the system set for the

estimation. After labeling the dataset, it's crucial to estimate the frequency of each class to guarantee a balanced distribution. System oversamples the labeled classes to obtain the proper list, ensuring a balanced distribution.

Phase 4: Artificial Neural network – The minMaxScaler function is applied to change the characteristics, rescale them to a given range. This estimator meticulously modifies and changes each feature to fit inside the designated range on the training set, for example, between zero and one. With the largest data point corresponding to the highest value and the smallest to the minimum value, MinMaxScaler progressively scales outliers into a predetermined range. Equations 2 and 3 can be utilized to express the minmaxscaler transformation.

$$X_{std} = \frac{(x - x.min(axis=0))}{(x.max(axis=0) - x.min(axis=0))} \text{-----}(2)$$

$$X_{scaled} = X_{std} * (max - min) + min \text{---}(3)$$

where min, max = feature_range.

After the scaling process all the (n-1) attributes are listed in X list and Nth attributes (i.e effort) is listed Y list. To divide our data into training and testing sets, we utilize the train_test_split() function. We should separate our data into features (X) and labels (y). The dataframe is split into four sections: y_train, y_test, X_train, and X_test. The model is trained and fitted using the X_train and y_train sets. To evaluate the precision of the model's predictions, we can make use of the X_test and y_test sets. Explicitly testing the size of the test sets and the train can be done. It is advisable to ensure that our train sets are larger than the test sets.

Set for Training: The set of information used to train the model is commonly known as the training dataset. The dataset used to train the model. The model carefully analyzes and acquires knowledge from this data. To ensure a precise evaluation of the final model's performance, a portion of the training dataset is assigned as the test set. The proposed system allocates 33% of our data for test sets and 67% for training sets. During the process of fine-tuning the model's hyper parameters, it is common to use a validation set, which is a portion of data that is extracted from the training set. It is used to assess the effectiveness of the model.

After the process is finished, data scaling is implemented to establish the minimum and maximum values of each feature as zero and one, respectively. The data is transformed using MinMax Scaler to ensure it falls within a predefined range, typically ranging from 0 to 1. Adjusting features to a specified range alters the data. It maintains the integrity of the original distribution while scaling the values to a predetermined range. A method called MinMaxScaler() is provided by the Sklearn preprocessor specifically for this purpose. When utilizing the MinMaxScaler() method, it generates outcomes for lists like train_X, test_X, train_Y, and test_Y. These four lists are used in the application of the Artificial Neural network, as explained.

During this stage, the sequential() method of Keras is utilized to set up an artificial neural network object with a sequential type for the lists train_X, test_X, train_Y, and test_Y. The concept behind sequential API sequencing involves arranging the Keras layers in a sequential manner. ANNs typically consist of layers that are organized in a sequential manner, allowing data to pass through each layer until it reaches the output layer. Then, the 'uniform' option is utilized to initialize a dense layer with 11 kernel units in an input layer. Initializers determine the initial random weights for the Keras layers. This layer also includes a "relu" activation function with a 20 input dimension. Then, the output layer is expanded to incorporate a dense layer with a single unit of kernel, which is initialized using the "uniform" parameter. The activation function used to strengthen the output layer is sigmoid.

The generated ANN model is built using the "adam" optimizer, which utilizes the Adam technique. Adam optimization is a method that utilizes stochastic gradient descent to adaptively estimate first- and second-order moments. The method employed by the Adam algorithm is highly efficient in terms of computation, requires minimal memory, and remains unaffected by diagonal rescaling of gradients. It is particularly well-suited for tackling problems that involve large amounts of data and parameters. During the compilation of the model, the binary cross entropy parameter is utilized to assess the model's accuracy, considering the presence of two distinct classes, namely "efforts estimated" and "efforts not estimated". The ANN neural network undergoes training for 100 epochs, utilizing a batch size of 100.

The equations 4 and 5 below display the ReLU and Sigmoid functions respectively.

$$f(x) = \max(0, x) \text{ --- (4)}$$

Where, x is any positive value
s

$$S(X) = \frac{1}{(1+e^{-x})} \text{ —————(5)}$$

Where,
X is the input to a neuron
f(x) = Relu Activation Function
S(x) = Sigmoid Activation Function
e= Euler's Number

The summary of the constructed ANN model can be seen in the below figure 2.

Model: "sequential"		
Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 11)	242
dense_1 (Dense)	(None, 1)	12
=====		

Figure 2: ANN Model Summary

Phase 5: Long short term memory (LSTM) - The LSTM neural network requires a few input parameters: test_x, test_y, train_x, and a scalar normalization object. Providing the LSTM model that makes use of a one-dimensional space with a single feature, ten units of samples, and a TRUE return sequence. The model is defined by parameters such as the shape of train_X1 along its first and second dimensions. Afterwards, a Dense layer is included with a kernel size of 1 and an activation function of "relu". The dense layer of a densely connected neural network uses activation functions on neurons to effectively gather new information. For this scenario, the fundamental LSTM neural network is comprised of two dense layers. For one-dimensional data, a single kernel with a size of 1 is utilized. When constructing a neural network, a batch size of 100 and 100 epochs are used, with the shuffle parameter set to false. Figure 3 presents the summary of the constructed LSTM model.

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
lstm (LSTM)	(None, 10)	1280
dense_2 (Dense)	(None, 1)	11
=====		

Figure 3: LSTM Model Summary

Phase 6: Software Effort estimation prediction through Decision making of hybrid model- In this phase of proposed model, the prediction values of both ANN and LSTM models are evaluated for their best results to correlate them using the hybrid technique. The obtained data from LSTM and ANN is used to apply the if-then rules of decision tree to evaluate the software effort estimation prediction data efficiently.

IV. RESULTS AND DISCUSSIONS

The construction of the designed approach for software effort estimation prediction involves using a Windows laptop with a core i5 CPU and 8 gigabytes of RAM. Python and the Spyder integrated development environment (IDE) are used in the development of the proposed model. As previously stated, the datasets from the URL: <https://www.kaggle.com/datasets/tamannashermin/software-effort-estimation-datasets> are obtained. This repository consists of archives of datasets from which china.csv is selected for the experiment. The dataset consists of 20 attributes

A series of 10 experiments were conducted for software effort estimation prediction. One way to assess the accuracy of the given model is by utilizing the Root Mean Square Error (RMSE). When assessing the discrepancy between two continuously connected metrics, the RMSE method simplifies the process. The models

analyze the levels of accuracy and inaccuracy in software effort estimation identification. Equation 6 is utilized to calculate the discrepancy once these data have been evaluated.

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (x_{1,i} - x_{2,i})^2}{n}} \quad \text{---(6)}$$

Where,

$\sum$ - Summary

$(x_1 - x_2)^2$ - Disparities Squared for the total of the differences

N - The Count of Trails

The RMSE for the recorded experiment is displayed in table 1, while figure 4 illustrates the corresponding graph. An average MAE of 2090.4 was obtained for the conducted experiment, with an RMSE of 45.72.

TABLE I: RMSE RECORDED DATA

Trails	Properly Efforts Estimated	MAE
1	451	2401
2	450	2500
3	460	1600
4	444	3136
5	474	676
6	465	1225
7	430	4900
8	453	2209
9	464	1296
10	469	961

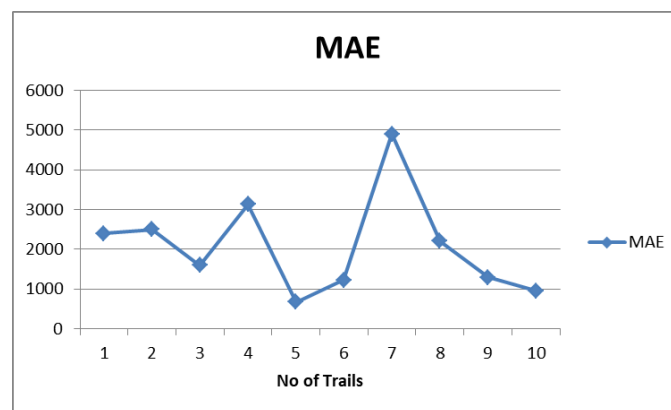


Figure 4: MAE Results for different trails

The obtained RMSE is compared with that of [7]. In [7] which offers a parametric software effort estimation model that is based on optimizing correction factors and multiple regression models. The goal of this model is to minimize estimate error and the influence of unsystematic noise, both of which have not been taken into consideration in prior studies. On the basis of the Use Case Points-based elements, the suggested method makes use of the Least Squared Regression models as well as the Multiple Linear Regression models. Experimental

research has been carried out by the authors in order to evaluate the accuracy of the suggested method's estimation and to compare it with three other ways that are related prior to this one. In [7], the lowest average Root mean square error (RMSE) achieved by Optimizing Correction Factors is 54.679. The RMSE of the model utilizing the ANN and LSTM is approximately 45.72. This comparison clearly demonstrates the superiority of our approach over the results obtained in [7]. This is due to the combination of two deep learning models, ANN and LSTM, which results in a favorable RMSE. We can find the comparison table and accompanying graph in table 2 and figure 5, respectively.

TABLE II: RMSE COMPARISON

Methods	RMSE
Optimizing Correction Factors	54.679
ANN-LSTM	45.72

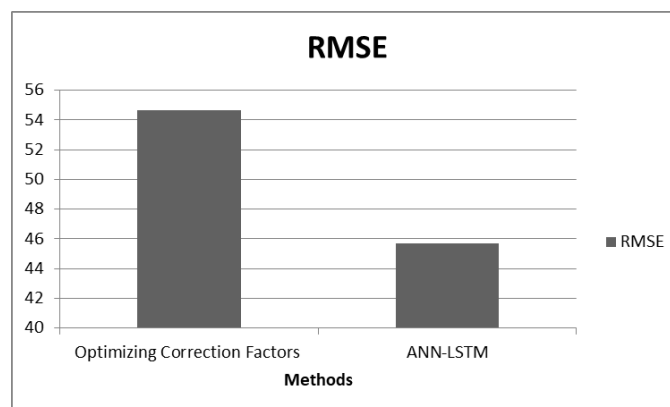


Figure 5: RMSE Comparison of ANN –LSTM with OCF method

V. CONCLUSION AND FUTURE SCOPE

This paper proposed ensembling technic of two deep learning models like ANN and LSTM for the prediction of software effort estimation. Many models rely on a single neural network model approach, which may not always yield satisfactory results. Therefore, the combination of multiple deep learning models to enhance the estimation of software efforts is a viable option. The system is using Support Vector Machine to classify the data in order to estimate software efforts. Using this classification protocol, we have implemented LSTM and ANN Neural network. The predictions' values are provided to the hybrid model, enabling it to determine the software effort estimation. The obtained results for the software effort estimation is measured for the Root mean square error parameter (RMSE). The RMSE is recorded for the ten trails of experiments and obtain a average MAE of 2090.4 with an RMSE of 45.72.

Enhancing the software effort estimation model allows for its application to a wide range of datasets, including live data provided by top officials in the software development team. This enables the model to provide accurate estimation parameters, thereby improving the development process.

REFERENCES

- [1] Zakrani abdelalia, Hain Mustaphaa, Namir Abdelwahedb, "Investigating the use of random forest in software effort estimation," in ScienceDirect Access, Data Sciences (ICDS 2018). 10.1016/j.procs.2019.01.042.
- [2] Ricardo Britto, Muhammad Usman, Emilia Mendes, "Effort Estimation in Agile Global Software Development Context," in Springer Access T. Dingsøyr et al. (Eds.): XP 2014 Workshops, LNBIP 199, pp. 182–192, 2014. Springer International Publishing Switzerland 2014
- [3] F. Sarro, A. Petrozziello and M. Harman, "Multi-objective Software Effort Estimation," 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, USA, 2016, pp. 619-630, doi: 10.1145/2884781.2884830.
- [4] N. Rankovic, D. Rankovic, M. Ivanovic and L. Lazic, "A New Approach to Software Effort Estimation Using Different Artificial Neural Network Architectures and Taguchi Orthogonal Arrays," in IEEE Access, vol. 9, pp. 26926-26936, 2021, doi: 10.1109/ACCESS.2021.3057807.

- [5] S. S. Gautam and V. Singh, "Adaptive Discretization Using Golden Section to Aid Outlier Detection for Software Development Effort Estimation," in *IEEE Access*, vol. 10, pp. 90369-90387, 2022, doi: 10.1109/ACCESS.2022.3200149.
- [6] H. T. Hoc, R. Silhavy, Z. Prokopova and P. Silhavy, "Comparing Stacking Ensemble and Deep Learning for Software Project Effort Estimation," in *IEEE Access*, vol. 11, pp. 60590-60604, 2023, doi: 10.1109/ACCESS.2023.3286372.
- [7] H. L. T. K. Nhung, V. Van Hai, R. Silhavy, Z. Prokopova and P. Silhavy, "Parametric Software Effort Estimation Based on Optimizing Correction Factors and Multiple Linear Regression," in *IEEE Access*, vol. 10, pp. 2963-2986, 2022, doi: 10.1109/ACCESS.2021.3139183.
- [8] M. S. Khan, F. Jabeen, S. Ghouzali, Z. Rehman, S. Naz and W. Abdul, "Metaheuristic Algorithms in Optimizing Deep Neural Network Model for Software Effort Estimation," in *IEEE Access*, vol. 9, pp. 60309-60327, 2021, doi: 10.1109/ACCESS.2021.3072380.
- [9] J. N. M. Alsheikh and N. M. Munassar, "Improving Software Effort Estimation Models Using Grey Wolf Optimization Algorithm," in *IEEE Access*, vol. 11, pp. 143549-143579, 2023, doi: 10.1109/ACCESS.2023.3340140.
- [10] A. J. Sánchez-García, C. López-Martín and A. Abran, "Gradient Boosting Optimized Through Differential Evolution for Predicting the Testing Effort of Software Projects," in *IEEE Access*, vol. 11, pp. 135235-135254, 2023, doi: 10.1109/ACCESS.2023.3337809.
- [11] P. Silhavy and R. Silhavy, "Evaluating Kernel Functions in Software Effort Estimation: A Comparative Study of Moving Window and Spectral Clustering Models Across Diverse Datasets," in *IEEE Access*, vol. 11, pp. 126335-126351, 2023, doi: 10.1109/ACCESS.2023.3329369.
- [12] Mohammad Muhairat, Saleh Aldaajeh, Rafa E. Al-Qutaish, "The Impact of Global Software Development Factors on Effort Estimation Methods," *European Journal of Scientific Research* ISSN 1450-216X Vol.46 No.2 (2010), pp.221-232, EuroJournals Publishing, Inc. 2010
- [13] P. Subitsha, J. Kowski Rajan, "Artificial Neural Network Models For Software Effort Estimation," *H*, VOL 2, ISSUE 4 76 ISSN 2347-4289.
- [14] Poonam Rijwani, Sonal Jaina, Dharmesh Santani, "Software Effort Estimation: A Comparison Based Perspective," Volume 3, Issue 12, December 2014.