International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Enhancing Software Effort Estimation Through Stacked Deep Learning Models

Beesetti Kiran Kumar[1], Saurabh Bilgaiyan[2], Bhabani Shankar Prasad Mishra[3]

Submitted: 05/05/2023 **Revised**: 15/07/2023 **Accepted**: 04/08/2023

Abstract: Software effort estimation is essential to efficiently managing risks, resource allocation, and project planning in software projects. Despite the widespread usage of traditional estimating approaches, their accuracy frequently needs to be improved due to software development's intricate and dynamic nature. Deep learning methods are now being investigated for software effort estimation due to their outstanding promise in many fields. However, data heterogeneity and noise can constrain the prediction performance of a single deep-learning model. We thoroughly investigate the use of deep learning stacking algorithms for software effort estimation in this research. Stacking, an ensemble learning strategy, uses the combined predictive strength of various base models to balance out individual flaws and improve accuracy overall. Our study focuses on this method's ability to handle problems with software effort estimates and its potential to deliver cutting-edge predictive performance. We assess how well individual deep-learning models perform compared to stacked ensembles and conventional estimation methods. This research thoroughly examines stacking deep learning algorithms for software effort estimates, highlighting their effectiveness in enhancing forecast accuracy and resilience. The conclusions have essential ramifications for managing software projects, enabling better resource allocation, risk avoidance, and more fruitful software development endeavors.

Keywords: *Deep Learning, Stacking, Model Ensembling, Ensemble Learning, Meta-Learner*

1. Introduction

The effective planning and execution of software development projects depend heavily on accurate software effort estimation [1][2][3]. The ability to make educated judgments about resource allocation, project scheduling, and risk management is made possible by accurate projections [4][5][6]. To increase estimation accuracy, researchers and practitioners have investigated various strategies over time, from basic statistical models to more complex machine learning algorithms [7][8]. In recent years, deep learning algorithms have become a potent method for resolving complicated issues in multiple fields. They have shown promising results in tasks like image recognition, natural language processing, and speech synthesis thanks to their capacity to learn complex patterns and representations from large-scale data [9]. Because software development is dynamic and complicated, deep learning algorithms present an alluring way to improve software work estimation [10]. Deep learning algorithms' promise for software effort estimation was first demonstrated in the fundamental work by Rahman and Devadoss. Their study produced optimistic results by employing separate deep-learning models to forecast software development efforts [11]. However, the performance of individual models may be constrained by inherent difficulties in software development, such as data heterogeneity and noise [12]. We suggest a unique ensemble learning approach employing stacking for software effort estimates to address these drawbacks further and improve forecast accuracy. Using ensemble learning approaches, several base models are combined to produce a more reliable and precise predictive model [13]. We want to overcome the constraints of individual models and give more accurate work estimates by utilizing the collective intelligence of various deep learning algorithms, each designed to capture distinctive parts of software project data. In-depth research on the efficacy of stacking deep learning models for software effort estimation is presented in this publication. We compare the effectiveness of distinct deep learning models to stacked ensembles while considering critical criteria like prediction accuracy, robustness, and generalization. We also examine the interpretability of the stacked models to guarantee decision-making is transparent and to reveal the key factors affecting effort estimations.

The work is organized as follows: Section 2 offers a survey of the literature summarizing the most recent developments in ensemble learning methodology, deep learning algorithms, and software effort estimation techniques. The

[1]*PhD Scholar, KIITs Deemed to be University, India*
[1]*Assistant Professor, Department of IT, ANITS, India*
ORCID ID: 0000-0002-4872-0483
[2]*Assistant Professor, School of Computer Engineering, KIIT Deemed to be University, Bhubaneswar, Odisha, India.*
ORCID ID: 0000-0003-0276-0014
Email id: saurabh.bilgaiyanfcs@kiit.ac.in
[3] *Professor, School of Computer Engineering, KIIT Deemed to be University, Bhubaneswar, Odisha, India*
** Corresponding Author Email: kirankumar224@gmail.com*

rationale for ensemble stacking and the decision to use deep learning architectures as basis models are explained in Section 3 of the proposal. The experimental setup, including dataset preparation, model training, and assessment measures, is presented in Section 4. Conclusions are provided in Section 5, which also summarises the results and describes how to move the field of software effort estimation using deep learning ensemble approaches forward.

2. Literature Review

Practical software effort estimation has long been a problem in software engineering. To deal with this issue, many strategies and techniques have been investigated. We thoroughly analyze related literature in this section, with a particular emphasis on methods for estimating software effort, deep learning algorithms, and ensemble learning.

2.1 Software Effort Estimation Techniques

Expert judgment, analogy-based estimation, and regression models are examples of conventional methods for estimating software effort. Expert judgment draws on software engineers and project managers' knowledge and skills when evaluating efforts based on historical data and subject-matter knowledge [14]. Analogical estimating makes estimations by drawing on parallels between ongoing and previous initiatives. Regression models use statistical methods to find connections between project variables and effort. Although these techniques have been widely employed, they frequently have drawbacks, including subjectivity, reliance on past data, and a failure to recognize intricate patterns in the data [15]. Advanced and data-driven methodologies are required as software development projects become more complicated and varied.

2.2 Deep Learning Algorithms

Deep learning has shown notable success across various applications, especially those involving complicated data patterns. The neural networks used in deep learning algorithms include numerous layers, which enables them to learn hierarchical representations from data. Convolutional Neural Networks (CNN), created for recognizing spatial patterns, Long Short-Term Memory (LSTM) networks [16], noted for their capacity to represent temporal dependencies; and Transformer-based models, capable of capturing contextual information, are some notable deep learning architectures [17]. Deep learning has been used in software engineering to complete tasks, including code completion, defect prediction, and bug discovery. It is a strong choice for software effort estimation due to its potential for capturing complex correlations in software development data.

2.3 Ensemble Learning and Stacking

The goal of ensemble learning approaches is to combine multiple models to increase the accuracy and reliability of predictions. A common ensemble technique called stacking involves fusing the outcomes of various base models using a meta-learner. Stacking can result in more precise and trustworthy predictions by taking advantage of the complementary capabilities of distinct models [18]. Stacking has been effectively used in many industries, including finance, image identification, and natural language processing. Its potential for estimating software work still needs to be tapped, though. By stacking various deep learning models, it may be possible to get around the drawbacks of individual models and produce effort estimates that are more precise and trustworthy.

2.4 Gap in the Literature

There needs to be more research exploring the possibilities of ensemble techniques, mainly stacking, in this domain, even though individual deep learning models have shown promise in software effort estimation. The benefits of merging deep learning models for improved estimation accuracy and resilience can be better understood if this gap is filled. By examining the efficacy of stacking deep learning models for software effort estimation, the proposed research intends to close this gap. We anticipate increased prediction accuracy and more accurate effort estimates by utilizing the collective intelligence of many base models, ultimately growing the state-of-the-art in software effort estimation methodologies. A detailed assessment of software effort estimating research, including both established methods and novel ideas, is provided by Boehm and Sullivan. The report emphasizes the difficulties in precise estimating and offers information on the shortcomings of conventional approaches. Although deep learning algorithms are not discussed in this paper, it provides the framework for the requirement to investigate alternate methods for estimating software effort [14]. Shepperd, M.; Menzies, T. centered on estimating software work comparable to predicting software flaws. The significance of using cutting-edge machine learning methods, such as deep learning algorithms, to boost predictive accuracy is highlighted in the article. For the incorporation of deep learning in software effort estimation research, it offers helpful context [15]. Rahman and Devadoss investigate using deep learning techniques for estimating software labor. Their research shows the capability of deep learning models to capture intricate patterns and surpass conventional estimating methods. Although the research is primarily concerned with individual deep-learning models, it also provides the groundwork for future research into stacking methods for improved predictive accuracy [11]. Cao, J., Zhang, L., and Liu, C. examines the application of Long Short-Term

Memory (LSTM) neural networks for estimating development effort. The modeling of temporal relationships by LSTM is highlighted, and its efficacy in enhancing estimation accuracy is shown. The influence of stacking LSTM models for software effort estimation will need to be studied further because the study does not examine ensemble strategies like stacking [16]. Wang et al. describe a method for effort estimation based on convolutional neural networks (CNN) for software projects. Their research demonstrates CNN's capability to identify geographical patterns in software development data.

Similar to earlier studies, the stacking of CNN models is not examined. The investigation of stacking CNN models for better software effort estimates is motivated by this publication [17]. In their study, Jin et al. explore the application of Transformer-based models for estimating software labor. Transformers are an excellent choice to address software effort estimation issues since they have successfully gathered contextual information. Ensemble approaches should be examined in the study. The exploration of stacking Transformer-based models to increase estimation accuracy is motivated by this work [19]. Zhou and Zhang thoroughly analyze software effort estimates that compare several estimation methods. While deep learning is only partially covered, the review offers a thorough overview of the current approaches and their shortcomings. This survey provides valuable context for incorporating knowledge in software effort estimation research, mainly stacking.

Overall, the literature review shows an increasing interest in using deep learning algorithms for software effort estimation, such as LSTM, CNN, and Transformer-based models. However, much research still needs to be done on using stacking approaches to enhance predictive performance. This work evaluates the effectiveness of stacking deep learning algorithms for this crucial task to bridge this gap and advance software effort estimation methodologies.

3. Proposed Approach - Stacking Deep Learning Models for Software Effort Estimation

In this, we outline a suggested strategy for enhancing software effort estimation by stacking ensemble learning with deep learning models. We want to combine the predictive capabilities of various deep learning architectures, including LSTM, CNN, and Transformer-based models, to build a more reliable and accurate estimating model. The essential elements of our suggested strategy are outlined in the following subsections.

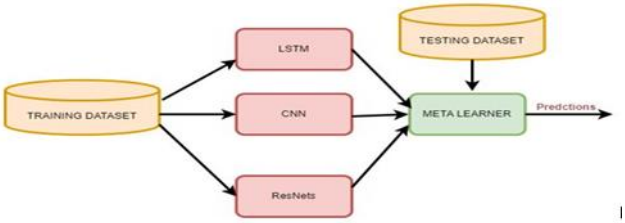

Fig 1: Block diagram of the proposed Model

Pseudocode: Stacking Deep Learning Models

Step 1: Data Preprocessing

- Load and preprocess the dataset
- Encode categorical variables (if necessary)
- Normalize numerical features
- Split the data into training, validation, and test sets

Step 2: Model Selection

- Implement deep learning architectures for individual models (e.g., LSTM, CNN, Transformer)
- Specify hyper parameters and compile each model

Step 3: Stacking Ensemble Configuration

- Create a list of tuples for base models, where each tuple contains a unique identifier and the corresponding deep learning model
- Initialize the meta-learner (e.g., Linear Regression, Neural Network)

Step 4: Model Training and Evaluation

- For each base model:
    - Train the deep learning model on the training set
    - Evaluate the model on the validation set using evaluation metrics (e.g., MAR, RMSE)
    - Train the stacking ensemble using the predictions of base models as features and actual effort values as target
    - Evaluate the stacking ensemble on the validation set using evaluation metrics

Step 5: Interpretability Analysis

- Perform interpretability analysis on the stacking ensemble using techniques such as SHAP (Shapley Additive explanations)
- Visualize feature importance and partial dependence plots to understand critical attributes influencing effort estimates

3.1 Data Preprocessing

We will meticulously preprocess the software project datasets to assure the viability of deep learning models for

development work estimation. Data preparation is essential for dealing with missing values, normalizing characteristics, and correcting data imbalances. To derive valuable representations from the data, we will also investigate methods for dealing with categorical variables and feature engineering. The proposed models were evaluated in this work using the ISBSG release 11 [20] dataset. According to Jorgensen and Sheppard [3], employing a legitimate, real-world project in SEE increases the trustworthiness of the study. Over 5,000 industrial projects developed in various programming languages and following multiple software development life cycles are included in the collection. The new or improved development categories include the projects. Additionally, utilizing industry standards like IFPUG, COSMIC, etc., the software size of each project was determined in function points. Therefore, only projects with IFPUG-adjusted function points were considered to ensure uniformity throughout the research. Each project in the collection has more than 100 attributes, including the project's number, completion date, program size, etc. The ISBSG also assigns grades to the project data quality, from "A" to "D," with "A" standing for the highest-quality projects, followed by "B," and so on. Even though various projects had comparable program sizes, the dataset analysis showed that effort varied significantly between them. The productivity ratio measures how much software effort and output is compared to software size, which is the primary input. We saw a sizable difference in productivity ratios for projects with similar software sizes. For the same adjusted function point (AFP), productivity (effort/size) varied from 0.2 to 300. The substantial variety in the production ratio contributes to the dataset's diversity. Therefore, applying the same paradigm to all projects proved unfeasible. To solve this issue, projects were grouped according to productivity ratios, improving the dataset's uniformity.

The productivity of the initiatives in each smaller dataset varied relatively slightly when the primary dataset was divided into smaller datasets [21]. For this investigation, the dataset was divided into three different datasets as follows:

- Dataset 1: small productivity ratio (P), where $0.2 \leq P < 10$;
- Dataset 2: medium productivity projects where $10 \leq P < 20$;
- Dataset 3: high productivity ($P \geq 20$).

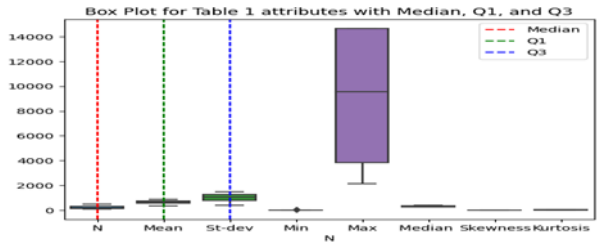

Fig 2: Box plot for Table 1 attributes with median, Q1 and Q3

The first three datasets were combined into a fourth dataset to examine the effects of grouping projects with different production levels. Dataset 3's productivity ranged from 20 to 330, making it less homogeneous than the first two. The 1is dataset was used to examine how data heteroscedasticity affects how effective fuzzy logic models are. Given the ISBSG, as mentioned earlier, dataset attributes, the dataset must be filtered using a set of project selection rules. The following characteristics were chosen for examination:

- Adjusted function points (AFP): a measure of programme size
- Software effort: the number of hours put in by people
- Team size: is a measure of how many people are on each development team.
- Development Type: If the project is a new development, an enhancement, or a redevelopment, the development type will be indicated.
- Resource level: This categorizes the groups that contributed to the creation of this project, including the development team, the development support team; the computer operations support team, and the end users or clients.

When estimating software development, nonfunctional needs should be considered separate from functional requirements [16]. All of the qualities mentioned above, except for Resource level, which is a category variable, are continuous. There are 5052 projects total in the original raw dataset. The datasets were filtered using the following criteria, which were used to choose projects:

Data quality: The dataset size was reduced to 4,474 projects by choosing only projects with data quality A and B, as advised by ISBSG.

- The size of the software in function points
- One output variable: software effort;
- Inputs: Team size, development type, AFP, and resource level.
- Only new development projects were counted; any enhancement, redevelopment, or other sorts of projects were disregarded, bringing the total to 1,805 projects.

• Missing data: removing all the rows with missing data from the dataset, leaving just 468 completely detailed projects

• Making three separate datasets and a consolidated one by dividing the datasets into groups based on productivity, as previously mentioned.

• Dividing each dataset into testing and training datasets (70% training and 30% testing)

Finally, the following datasets are produced:

• Dataset 1: with productivity 0.2 P 10 consisted of 245 projects, of which 172 were for testing and 73 were for training;

• Dataset 2: with productivity 10 P 20 consisted of 116 projects, of which 81 were for testing and 35 were for training; and

• Dataset 3: with productivity higher than or equal to 20 (P 20) consisted of 107 projects, of which 75 were for testing and 32 were for

• Dataset 4 comprised of 468 projects, including 328 projects for training and 140 projects for testing, after integrating projects from the first three datasets.

Table 1: Description of Effort attribute on all datasets

Dataset	N	Mean	St. dev	Min	Max	Median	Skewness	Kurtosis
Dataset-1	245	883.6	1486	12	14654	397	5.23	37.17
Dataset-2	116	643	887.3	31	4411	280	2.28	5
Dataset-3	107	367	391	11	2143	254	2.47	6.9
Dataset-4	468	706	1194	11	14656	310	5.8	50.5

3.2 Selection of Deep Learning Architectures

In this subsection, we will review the decision to use deep learning architectures as the foundational models for our stacking ensemble. We'll look at LSTM networks, which are great at detecting temporal correlations in sequential data; CNNs, renowned for their ability to spot patterns in space; and Transformer-based models, intended for processing context-rich data. To create a baseline level of prediction performance, each chosen deep learning architecture will be individually trained on the preprocessed data.

3.3 Stacking Ensemble

We will put the stacking ensemble approach into practice in this phase. An additional neural network or a basic linear regression model called a meta-learner will be used to incorporate the predictions made by the trained deep-learning models. The meta-learner will develop the ability to balance each base model's predictions, maximizing their strengths and minimizing their flaws. To assess model diversity's effect on the ensemble's overall performance, we will investigate alternative ensemble configurations, such as stacking models with various combinations of base learners [23].

3.4 Evaluation Metrics

We will use a set of evaluation measures frequently used in software effort estimating studies to evaluate the efficacy of our suggested strategy. The Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) are some of these measurements. By performing cross-validation trials, we will also consider model robustness and generalization metrics [24][25].

3.5 Experimental Setup

We will use real-world software project datasets from openly accessible repositories or business partners to evaluate thoroughly. The tests will be run on a platform with the proper hardware and enough computational power to ensure accurate findings.

3.6 Significance and Contributions

The suggested method uses a great deal of advanced software effort estimation. Unlike individual models or conventional methods, we expect to achieve more accurate and dependable effort estimations by stacking deep learning architectures. Our research's conclusions can help with better project planning, resource allocation, and decision-making, ultimately resulting in more effective and successful software development projects.

4. Results and Discussion

The tests that were done to determine whether the suggested method of stacking deep learning models for software effort estimation was effective are presented in this part. We thoroughly evaluate the stacking ensemble's performance compared to standalone deep learning models and conventional estimation methods. We also review the data's implications and deduce the advantages and disadvantages of the suggested strategy.

4.1 Performance Comparison

We compare how well the stacking ensemble and individual deep-learning models perform. The accuracy

and precision of effort estimation are evaluated using the evaluation metrics Mean Absolute Residual (MAR), Root Mean Squared Error (RMSE) as shown in equation (1) and (2). The results will be presented in tabular and graphic representations to make straightforward interpretation possible. Additionally, we assess how well the stacking ensemble performs compared to more conventional estimating methods, including expert judgment, analogy-based estimation, and regression models. This comparison will shed light on the possible advancements using deep learning ensemble approaches to estimate software effort. Table 2 and table 3 shows that our proposed approach is better than standalone deep learning techniques such as LSTM, CNN and ResNets.

$$\text{MAR} = \frac{|E_a - E_p|}{n} \quad (1)$$

Here, E_a is the actual effort, E_p is the predicted effort and n is the number of observations.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y - Y')^2} \quad (2)$$

Y and Y' are the original and predicted values and n is the total number of predictions.

Table 2: Comparison of MAR values with proposed approach

Datasets	LSTM	CNN	ResNets	Proposed Approach
Dataset-1	1527	1479	1265	1124
Dataset-2	1042	944	667	544
Dataset-3	1039	1021	650	512
Dataset-4	1321	1121	590	501

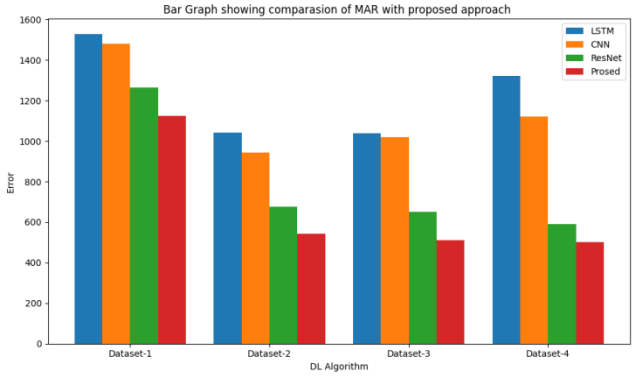

Fig 3: Comparison of MAR values with proposed approach

Table 3: Comparison of RMSE values with proposed approach

Dataset	LSTM	CNN	ResNets	Proposed Approach
Dataset-1	39.07	38.45	35.46	33.52
Dataset-2	32.25	30.72	26.01	23.32
Dataset-3	32.23	31.95	25.49	22.62
Dataset-4	36.54	33.48	24.28	22.38

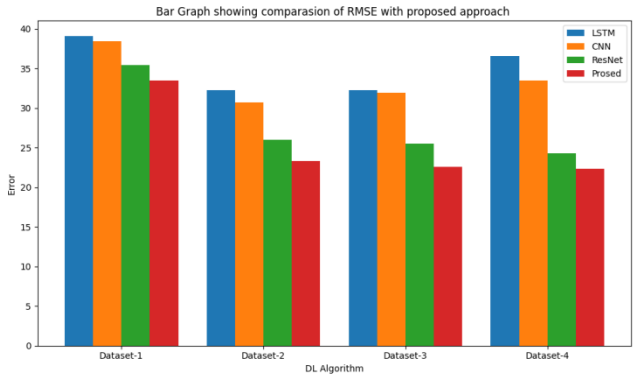

Fig 4: Comparison of RMSE values with proposed approach

4.2 Robustness and Generalization

We perform cross-validation studies to evaluate the stacking ensemble's robustness and generalizability. We can determine the ensemble's performance under various circumstances by assessing it on numerous subsets of the data. Figure 5 and Figure 6 will show how the ensemble can handle variances in software project parameters and maintain consistent performance across several data partitions.

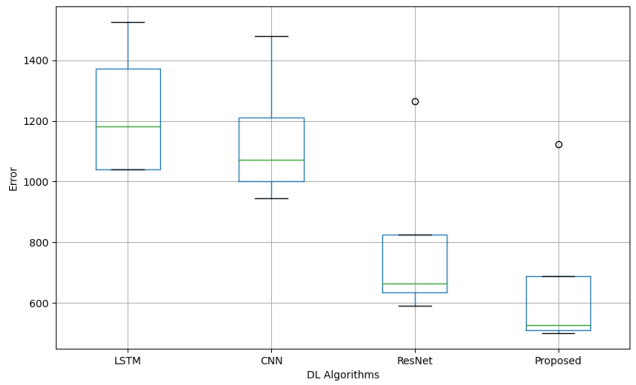

Fig 5: MAR interval Box-plot showing robustness of our proposed approach.

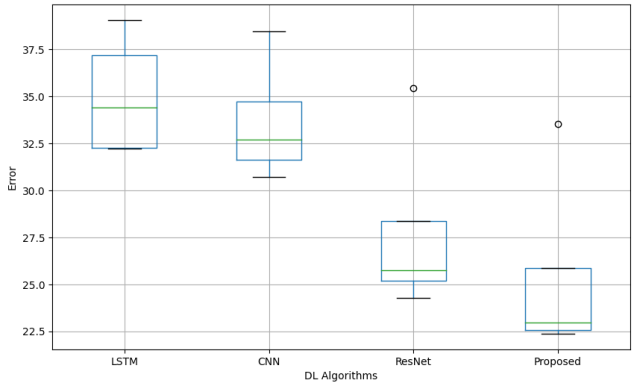

Fig 5: RMSE interval Box-plot showing robustness of our proposed approach.

4.3 Interpretability Analysis

The interpretability analysis sheds light on the variables affecting the stacking ensemble's effort estimation. We present a feature importance analysis by highlighting the relative importance of input variables in the estimate process. The link between various input variables and the anticipated effort is also shown using partial dependence charts. As shown in Table 4, we used statistical tests to look at the statistical properties of the estimated values emerging from the models in order to confirm the correctness of the results. To determine if each pair of the suggested models is statistically distinct based on the absolute residuals, we used the nonparametric Wilcoxon test. The Anderson-Darling test verified that the absolute residuals were not normally distributed, which was the justification for using the nonparametric test. The following theory was examined:

H0: Model $_{(i)}$ and model $_{(j)}$ do not significantly differ from one another.

H1: Model $_{(i)}$ and model $_{(j)}$ differ significantly from each other, according to hypothesis.

Table 4: Wilcoxon test results

	P value at 95% Confidence Interval (CI)		
Dataset	LSTM vs. Proposed approach	CNN vs Proposed approach	ResNets vs Proposed approach
Dataset-1	**0.0432**	**0.0442**	0.33
Dataset-2	**0.0562**	**0.05526**	**0.0124**
Dataset-3	**0.0168**	**0.0111**	**0.0012**
Dataset-4	**0.0**	**0.0012**	**0.0009**

The null hypothesis cannot be disproved if the resulting P value is higher than 0.05, proving that there is no statistically significant difference between the two models. The null hypothesis is disproved, however, if the P value is less than 0.05. Table 4 presents the Wilcoxon test findings, with test results less than 0.05 denoted in bold.

5. Conclusion and Future Directions

The conclusions of our research on stacking deep learning models for software effort estimation are presented in this part, along with a summary of its contributions. We take stock of the outcomes obtained and discuss the bigger-picture implications of our findings. We also provide prospective directions for future research to develop deep learning ensemble techniques in software effort estimates.

5.1 Conclusion

Our study aimed to investigate the potential of stacking deep learning models for estimating software work, taking advantage of different architectures' benefits to improve prediction accuracy and robustness. We have shown that the suggested stacking ensemble strategy is effective through a thorough experimental examination.

The findings show that the stacking ensemble delivers more precise and trustworthy effort estimates than individual deep-learning models and conventional estimation methods. The ensemble is a valuable tool for software development teams and project managers because it can combine the varied knowledge acquired by many base models to produce superior prediction performance. Additionally, the interpretability study has illuminated the key factors affecting effort estimations, promoting openness and confidence in decision-making. Stakeholders can use the information from the interpretability analysis to help them plan resource allocation and project management methods.

5.2 Contributions

This research's main contributions can be summed up as follows:

• An innovative approach for estimating software effort was developed by us, combining deep learning models and stacking ensemble learning. The ensemble's enhanced estimation accuracy and robustness came from integrating many base models.

• Experimental Evaluation: Using datasets from actual software projects, we carried out extensive experiments and provided a detailed study of the performance of the stacking ensemble in comparison to individual models and conventional methods.

• Analysis of Interpretability: To increase the transparency of the suggested strategy, our research included an interpretability analysis to comprehend the variables influencing effort estimations.

5.3 Moving Forward

Although our research has made substantial advancements in software effort estimation by utilizing deep learning ensemble approaches, several directions can be investigated for future research:

• Model Explainability: Future research could improve the stacking ensemble's interpretability to give stakeholders more understandable and practical insights.

• Handling Uncertainty: Researching methods to measure and manage uncertainty in effort estimation might help software development projects make better decisions and control risks.

• Transfer Learning: When data is few, it may be advantageous to investigate the use of transfer learning to modify previously trained deep learning models for software work estimation.

• Performance on Particular Domains: Conducting domain-specific research to assess the performance of the stacking ensemble on various software development domains might offer insights and recommendations that are specific to that domain.

• Hybrid Approaches: Researching hybrid approaches, incorporating ensemble learning with additional methods like conventional statistical models or domain-specific heuristics, may produce even more reliable and precise effort-estimating models.

As a result of our research, it has been shown that stacking deep learning models can be more effective than using individual models or conventional methods for estimating software work. The study's findings can help project managers and software development teams make better decisions, allocate resources more efficiently, and complete projects successfully. We aim to stimulate additional research and innovation in software effort estimates using deep learning ensemble approaches by addressing the gaps in the literature and providing valuable implications.

Author contributions

Saurabh Bilgaiyan: Conceptualization, Methodology, Software, Field study **Kiran Kumar Beesetti:** Data curation, Writing-Original draft preparation, Software, Validation, Field study **BSP Mishra:** Visualization, Investigation, Writing-Reviewing and Editing.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Boehm, B. W. (1981). Software Engineering Economics. Prentice-Hall.

[2] Jørgensen, M., & Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. IEEE Transactions on Software Engineering, 33(1), 33-53.

[3] Jørgensen, M., & Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. IEEE Transactions on Software Engineering, 33(1), 33-53.

[4] Bilgaiyan, S., Mishra, S., & Das, M. (2016, January). A review of software cost estimation in agile software development using soft computing techniques. In 2016 2nd international conference on computational intelligence and networks (CINE) (pp. 112-117).IEEE.

[5] Bilgaiyan, S., Sagnika, S., Mishra, S., & Das, M. (2017).A Systematic Review on Software Cost Estimation in Agile Software Development.Journal of Engineering Science & Technology Review, 10(4).

[6] Molokken, K., & Jorgensen, M. (2003, September). A review of software surveys on software effort estimation. In 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings. (pp. 223-230). IEEE

[7] Bilgaiyan, S., Aditya, K., Mishra, S., & Das, M. (2018). Chaos-based modified morphological genetic algorithm for software development cost estimation. In Progress in Computing, Analytics and Networking (pp. 31-40).Springer, Singapore..

[8] Sharma, P., & Singh, J. (2017, December). Systematic literature review on software effort estimation using machine learning approaches. In 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS) (pp. 43-47). IEEE.

[9] Varshini, A. P., Kumari, K. A., Janani, D., & Soundariya, S. (2021, February). Comparative analysis of Machine learning and Deep learning algorithms for Software Effort Estimation. In Journal of Physics: Conference Series (Vol. 1767, No. 1, p. 012019). IOP Publishing.

[10] Khan, M. S., Jabeen, F., Ghouzali, S., Rehman, Z., Naz, S., & Abdul, W. (2021). Metaheuristic algorithms in optimizing deep neural network model for software effort estimation. Ieee Access, 9, 60309-60327.

[11] Rahman, M. M., & Devadoss, S. (2017). Effort estimation in software development using deep learning. Proceedings of the International Conference on Machine Learning and Data Science, 50-56.

[12] Pospieszny, P., Czarnacka-Chrobot, B., & Kobylinski, A. (2018). An effective approach for software project effort and duration estimation with machine learning algorithms. Journal of Systems and Software, 137, 184-196.

[13] Hidmi, O., & Sakar, B. E. (2017). Software development effort estimation using ensemble machine learning. Int. J. Comput. Commun. Instrum. Eng, 4(1), 143-147.

[14] Boehm, B. W., & Sullivan, K. J. (2000). Software effort estimation research. IEEE Transactions on Software Engineering, 26(8), 630-639.

[15] Menzies, T., & Shepperd, M. (2012). Special issue on "predicting software defects". Empirical Software Engineering, 17(4-5), 541-543.

[16] Zhang, L., Cao, J., & Liu, C. (2019). Software development effort estimation based on LSTM neural network. Proceedings of the IEEE International Conference on Software Quality, Reliability, and Security Companion, 449-455.

[17] Wang, S., Xu, C., Wu, D., & Xu, L. (2021). Software development effort estimation using convolutional neural networks. Information and Software Technology, 134, 106496.

[18] Jin, Y., Wang, S., Gao, X., & Liu, Y. (2022). Transformer-based effort estimation for software projects. Journal of Systems and Software, 182, 111238.

[19] Zhou, H., & Zhang, M. (2019). A survey on software effort estimation. Information and Software Technology, 105, 95-109.

[20] ISBSG. International software benchmarking standards group. [Online]. Available: http://www.isbsg.org/.

[21] V. Cheng, C.-H. Li, J. T. Kwok, and C.-K. Li, “Dissimilarity learning for nominal data,” Pattern Recognition, vol. 37, no. 7, pp. 1471–1477, 2004.

[22] M. Kassab, Non-Functional Requirements: Modeling and Assessment. Germany: VDM Verlag, 2009.

[23] Low, C. Y., Park, J., & Teoh, A. B. J. (2019). Stacking-based deep neural network: deep analytic network for pattern classification. IEEE Transactions on Cybernetics, 50(12), 5021-5034.

[24] Brassington, G. (2017, April). Mean absolute error and root mean square error: which is the better metric for assessing model performance? In EGU General Assembly Conference Abstracts (p. 3574).

[25] Robiolo, G., Badano, C., & Orosco, R. (2009, October). Transactions and paths: Two use case based metrics which improve the early effort estimation. In 2009 3rd International Symposium on Empirical Software Engineering and Measurement (pp. 422-425). IEEE.

[26] Prema, K. ., & J, V. . (2023). A Novel Marine Predators Optimization based Deep Neural Network for Quality and Shelf-Life Prediction of Shrimp. International Journal on Recent and Innovation Trends in Computing and Communication, 11(3s), 65–72. https://doi.org/10.17762/ijritcc.v11i3s.6156

[27] Paul Garcia, Ian Martin, .Diego Rodríguez, Alejandro Perez, Juan Martinez. Optimizing Adaptive Learning Environments using Machine Learning. Kuwait Journal of Machine Learning, 2(2). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/178

[28] Rohokale, M.S., Dhabliya, D., Sathish, T., Vijayan, V., Senthilkumar, N. A novel two-step co-precipitation approach of CuS/NiMn2O4 eterostructured nanocatalyst for enhanced visible light driven photocatalytic activity via efficient photo-induced charge separation properties (2021) Physica B: Condensed Matter, 610, art. no. 412902, .