Software Effort Estimation through Ensembling of Base Models in Machine Learning using a Voting Estimator

Beesetti Kiran Kumar[1], Saurabh Bilgaiyan[2], Bhabani Shankar Prasad Mishra[3]
PhD Scholar, KIITs Deemed to be University, Bhubaneswar, Odisha, India[1]
Assistant Professor, Department of Information Technology, ANITs, Vishakhapatnam, India[1]
Assistant Professor, School of Computer Engineering, KIITs Deemed to be University, Bhubaneswar, Odisha, India[2]
Professor, School of Computer Engineering, KIITs Deemed to be University, Bhubaneswar, Odisha, India[3]

***Abstract*—For a long time, researchers have been working to predict the effort of software development with the help of various machine learning algorithms. These algorithms are known for better understanding the underlying facts inside the data and improving the prediction rate than conventional approaches such as line of code and functional point approaches. According to no free lunch theory, there is no single algorithm which gives better predictions on all the datasets. To remove this bias our work aims to provide a better model for software effort estimation and thereby reduce the distance between the actual and predicted effort for future projects. The authors proposed an ensembling of regressor models using voting estimator for better predictions to reduce the error rate to over the biasness provide by single machine learning algorithm. The results obtained show that the ensemble models were better than those from the single models used on different datasets.**

Keywords*—Machine learning; software effort estimation; voting; regression; evolutionary algorithms

I. INTRODUCTION

For a given project, the effort estimation of software is always a burdensome task. For an extended period, team and finance managers strive to precisely calculate the effort, cost, and time while helping evaluate the project's schedule and budget parameters [1]. It is very tough to predict those specifications during the early stages of the project, where the scope of every module has yet to be marked, and when we still have no conclusive evidence for the ultimate functional requirements of the product [2]. Most frequently, insufficient knowledge of the affecting factors and the possible risks that can happen, or the work deadline fears, and conventional effort estimation methods [3], which are widely accepted by the opinions of various software domain experts, may sometimes lead to erroneous estimates . Because of these, the software product may not be delivered in time with the expected non-functional requirements. Though there are frequent improvements in the up-gradation of software development standards, surveys [4] show that only a quarter of the total number of beginnings is successful. These issues, which result in going beyond the budget or schedule, may lead to its termination [5]. Though the usage of agile methodology [6] reduced some concerns, project omissions are still occurring because of not having access to all the country's projects. For a country that just has a limit to its country's projects, obtaining success in the projects is still a problem. All the managers of the project are claiming otherwise. The causes are the poor skillsets of the teams on the project and less bonding with stakeholders. There are high chances of more project success if the effort is well predicted from the beginning. But once the project's parameters are set, it's not a good idea to increase either the budget or the schedule because that could lead to risks that are hard to predict.

The client's approval for that scenario must be independent of the chosen process model for project development, time, and cost determination. For this case, some simple and easy conventional techniques that experts accept, such as PERT, CPM, etc., are primarily deployed [7]. Researchers started working on methods depending on software lines of code, and functional points as the previously mentioned techniques are vulnerable [8]. In various ways, the software parameters try to be up to date with the improved technologies. However, those techniques struggle to keep pace in the fastest-growing world, specifically with the reusability components and software dependencies that have already gone so far in their enhancement.

After considering all these drawbacks, researchers dug deep for efficient predictive techniques for effort, especially in data science areas [9]. This area is highly trusted, proving its potential with uncertain and unstructured data. Hence, it is believed that they can find the effort and duration way better than existing models. In other methods, they consider patterns in the previous data and do not rely on human influences, making them unique in their work. The factors behind this are systematic research that builds the best model for prediction to reduce the error rate for data. Biased models have been generated for some time. Their work is limited to a particular dataset, repeatedly underfitting or overfitting using varied ensemble techniques. The critical role is preparing data that has a crucial impact on the model, but divergent methods are to be used. Even the individual algorithms may not give an improved score, which is evident from various journals. The same can be repeated with effort and cost prediction when working with data and building models with those algorithms [10].

Though using all project parameters produced better results in the literature, some works included proper feature

selection strategies, eliminating irrelevant features for less computation usage and creating a unique effect with fewer features than the others. Some of them are the Genetic Algorithm [11], PSO [12], and WCO [13].

This work aims to improvise the existing models on reliable machine learning algorithms for the best effort prediction. An averaging ensemble of various regressors proposes a hybrid model for this aim. Also, the proposed work conducted experiments on various datasets such as cocomo81, china, Desharnais, Maxwell, Kemerer, and Albrecht etc. to evaluate the behavior of the model with varying datasets.

II. Related Work

Amini et al. [14], in their paper combined two techniques, namely embedded and wrapper methods. The main motto of their writing is to integrate GA into regularized learning to improve prediction accuracy in regression problems. The outcome of their study reduced the dimension feature space by over 80% without affecting the accuracy. De Carvalho et al. [15] proposed an Extreme learning machine for forecasting software efforts. For selecting the best features, the Pearson correlation coefficient is used for feature selection. Extreme Learning Machines (ELM) are used with different numbers of hidden layers in their work. The ELM model values are compared with the models mentioned in the literature, namely LR, SVM, KNN, and MLP. The metric evaluated for comparison is MAE.

Ghosh et al. [16] proposed the binary variant of SFO for selecting features. This work compared ten state-of-the-art techniques and declared that BSFO based on adaptive hill climbing had shown better reliability. Carbonera et al. [17] surveyed over 120 studies and indicated that this study encouraged the researchers to minimize the space in the effort estimation. Chhabra et al. [18] worked in soft computing Fuzzy model along with PSO. This Fuzzy logic improved the existing COCOMO technique. The metric followed is MRE for result comparison. Ghatasheh et al. [19] proposed a firefly algorithm to optimize software effort. The results were better than the conventional models used earlier.

Wani et al. [20] worked on ANN and PSO. The limitation in their work is that the combination is giving better results for only the cocomo81 dataset. The ANN showed fast training speed than MLP. This method ended with lesser MdMRE and MRE than other models. Ali et al. [21] used all bio-inspired feature selection strategies with Support Vector and Random Forest regressors. The evaluation metrics considered are Correlation Coefficient (CC), Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Relative Absolute Error (RAE), and Root Relative Squared Error (RRSP).

Kodmelwar et al. [22] modified a neural network for effort prediction. The proposed method uses java for the front-end tool. The metrics used for comparison are PRED, MRE, MAE, MMRE, and RE. Desai et al. [23] combined ANN with cuckoo optimization, and experiments were performed on various datasets. This hybrid combination worked well then all other literature models except for the ACO technique. Langsari et al. [24] optimized the parameters of the COCOMO II model using particle swarm optimization. PSO is a valuable strategy for resolving dataset uncertainty and optimizing the values. The author worked on the Turkish software industry dataset.

Hosni et al. [25] concentrated on parameter tuning ensemble using grid search optimization. The authors evaluated results over seven datasets to compare statistical measures, namely mean, median, and inverse ranked weighted mean. Used three algorithms, GS, PSO, and UC-Weka, and concluded that PSO gained over other approaches. Goyal et al. [26] proposed an SG5 neural network model trained on the Cocomo dataset and tested in the Kemerer dataset. It excelled over the traditional models. Padhy et al. [27] developed an Aging and survivability-related reusability optimization model, and the software metric estimation is done with the help of UML or Class diagrams. To overcome the limitations of ANN, some different Evolutionary Computing (EC) algorithms like Genetic Algorithms, Differential Evolution, and Particle Swarm Optimization (PSO) have been proposed. By implementing the above algorithms, the regression outputs are improved so that the results are significantly accurate and most effective.

Pospieszny et al. [28] proposed ensemble averaging with a 3-fold validation, namely SVM, MLP, and GLM, to predict both effort and duration. Here used the standard ISBSG dataset and considered the MMRE and PRED metrics. In their paper, Shekhar et al. [29] discussed various software cost estimation techniques and models. The authors classified these techniques into algorithmic and non-algorithmic, which helps the software team rule out the weaker methods and provides specific areas for considering an approach.

Venkatesh et al. [30] calculated the workforce to determine the cost and effort of the project, which outperformed other models, like regression models and neural networks. This work applied to several PROMISE datasets by considering RMSE as the root metric. Nassif et al. [31] worked on four different neural networks, the oldest projects used for training and the newest projects used for testing. Here ten-fold cross-validation is achieved. The author concluded that in 60% of datasets, CCNN performed better than other models, and on 40% of datasets, RBFNN performed better than others. Miandoab et al. [32] proposed a hybrid Algorithm using a particle swarm optimization algorithm and fuzzy logic.

Dizaji et al. [33] combined Ant Colony Optimization (ACO) and Lorentz transformation as Chaos Optimization Algorithm (COA). The meta-heuristic algorithms like ACO and COA are used to estimate the cost of the software. Mean Absolute Relative Error (MARE) is taken into consideration. Here the dataset is classified and distributed among the ACO and hybrid ACO and COA algorithms according to their functionalities. The results show that the performance is improved and efficient when the ACO algorithm is combined with COA.

III. METHOD

The proposed approach introduces a novel method of using ensemble techniques with voting for software development effort estimation. This approach combines the strengths of multiple models and leverages the diversity of their predictions to improve accuracy. By investigating the impact of different factors on the accuracy of the ensemble with voting, this approach can provide insights into how to optimize the performance of the ensemble for different datasets and problems. The proposed approach can also have practical applications for software development organizations, as it can help them to make more accurate and informed decisions about project planning and resource allocation. The proposed architecture is illustrated in Fig. 1.

- Collect historical project data: Gather historical project data including information on the size of the project, the number of developers involved, the complexity of the software, and the amount of time and resources required to complete the project.
- Preprocess the data: Preprocess the data to remove any outliers or errors, and to convert the data into a format that can be used by the ensemble models.
- Train multiple estimation models: Train multiple estimation models on the preprocessed data, such as linear regression, decision trees, neural networks, and support vector machines.
- Implement the voting algorithm: Implement the voting algorithm to combine the predictions from the multiple models. There are different types of voting algorithms such as majority voting, weighted voting, and threshold voting.
- Evaluate the ensemble with voting: Evaluate the accuracy of the ensemble with voting using a validation set of historical data that was not used during training. Compare the performance of the ensemble with voting against individual models and other ensemble techniques.
- Investigate the impact of different factors: Investigate the impact of different factors on the accuracy of the ensemble with voting, such as the number of models, the type of models, the voting algorithm, and the size and quality of the historical data.
- Apply the ensemble with voting to new data: Apply the ensemble with voting to new software development projects to assess its accuracy and reliability in real-world scenarios.

A. Data Preprocessing

Preprocessing of data involves data cleansing approaches. It has a clear positive impact on training the machine learning models. It reduces the dataset's noise by filling in missing values, removing duplicate records, dropping unnecessary columns, etc. Finally, it produces the data in its best representation to be used for model building. Without preprocessing, models might learn the noise as an underlying pattern, leading to overfitting or underfitting the data. Here, we dropped some attributes in our work, such as project ids, dates of projects, other categorical details, etc. We ignored the missing data records.

B. Normalization

Normalization is done as a second step, and it is essential to scale the features within a range for the model's performance. This normalization sets the feature scale from 0 to 1 and is implemented using the MinMax scalar in Python. In our datasets, we normalized all the input and output features.

$$x\_new = \frac{x - x\_min}{\text{x\_max} - x\_min} \quad (1)$$

C. 5 - Cross Fold Validation

Cross-fold validation is an interesting technique, which makes our model more reliable. Instead of considering a particular subset for training and the remaining part for testing, it uses the entire dataset for training and testing purposes. A five-fold validation usually splits the entire dataset into five equal sets or folds, where for every time, four sets are used for training, and one set is used for testing. This process is repeated for four (k-1) iterations, i.e., all possible combinations, and it will give the average score of all iterations.

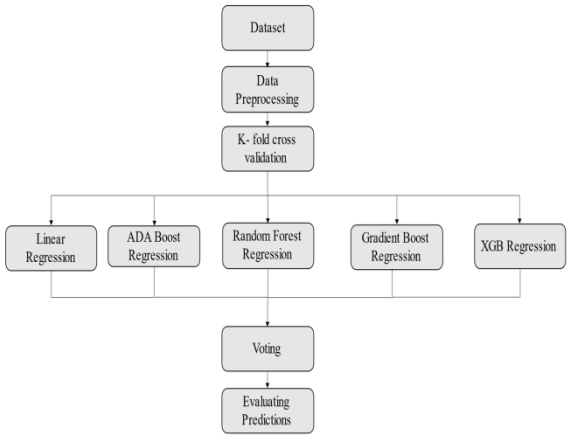


Fig. 1. Proposed architecture

D. Algorithms

In our work, we build a hybrid model with the help of five machine learning Regression algorithms. Each algorithm has a different structure in its implementation.

1) Linear regression: Linear Regression frames an equation for the given attributes to fetch the target variable. It assumes a linear relationship between the characteristics of a dataset. The equation is y = f (x), where y represents the output variable and x is the set of input attributes. This algorithm performs better than complex models when the dataset is linear.

2) Random forest: Random Forest is a bagging model. It constructs several trees for prediction. Every tree is constructed from a subset of the training data. Every tree will give some effort for a test set. All predictions are averaged to

get the final estimate of how much work needs to be done, lowering the result's error rate.

3) Boosting techniques: Every boosting algorithm has a base model. After each iteration, a new weak learner is added to the sequence of learners; every iteration model reduces the residual effort. We implemented three boosting models in our work: Ada Boost, Gradient Boost (GB), and Extreme Gradient Boost. AdaBoost handles missing data well and undergoes no overfitting. It has fewer parameters to tune when needed and is sensitive to outliers. Gradient Boosting is a sequence of tree learners robust to outliers, depending on residuals. XGB has been showing better results than GB as it includes the calculation of similarity weights.

E. Voting

Every algorithm is unique in its background processing of data. Hence, all algorithms can find their patterns of data. Here comes the idea of ensembling [34]. Ensembling is obtained by combining various models. Bagging, boosting, voting, etc., are some of the ensemble approaches [35]. Here we aggregated predictions of various models, i.e., averaged the output predictions of all models and produced one model closer to the actual effort than any individual model. We took the Linear Regression, Decision Tree Regression, Random Forest Regression, Support Vector Regression, and Neural Network Regression outputs, calculated the average of all the values, and compared them with the actual effort in the test dataset. The results are given for evaluation metrics.

F. Pseudo Code

```
# Step 1: Collect historical project data
data = load_data()
# Step 2: Preprocess the data
data = preprocess_data(data)
# Step 3: Train multiple estimation
models
models = []
for i in range(num_models):
    model = train_model(data)
    models.append(model)
# Step 4: Implement the voting algorithm
def ensemble_predict(models, input):
    predictions = [model.predict(input)
for model in models]
    return voting_algorithm(predictions)
# Step 5: Evaluate the ensemble with
voting
validation_set = load_validation_set()
ensemble_accuracy = 0
for input, target in validation_set:
    ensemble_prediction =
ensemble_predict(models, input)
    ensemble_accuracy +=
evaluate_prediction(ensemble_prediction,
target)
ensemble_accuracy /= len(validation_set)
# Step 6: Investigate the impact of
different factors
# For example, vary the number of models,
the type of models, the voting algorithm,
and the size and quality of the
historical data.

# Step 7: Apply the ensemble with voting
to new data
new_data = load_new_data()
for input in new_data:
    ensemble_prediction =
ensemble_predict(models, input)
process_prediction(ensemble_prediction)
```

IV. Evaluation Criteria

In problems like predicting continuous values, we calculate the error rate given as the difference between the actual and predicted values. For our problem statement, we looked at the MAE (Mean Absolute Error), MSE (Mean Squared Error), and RMSE (Root Mean Square Error) metrics, which are used to compare models.

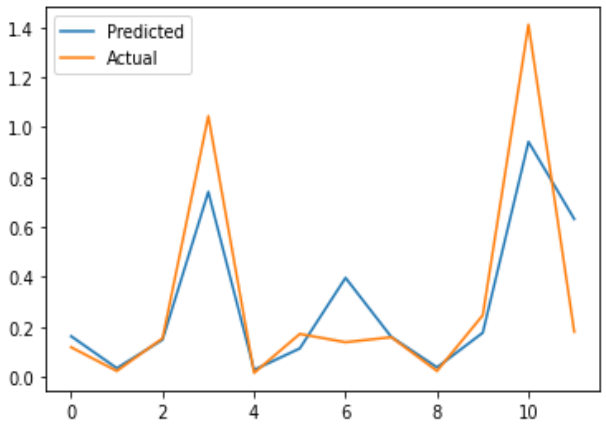
$$MAE = \frac{\sum abs(y_{actual} - y_{predicted})}{n} \quad (2)$$

$$MSE = \frac{\sum (y_{actual} - y_{predicted})^2}{n} \quad (3)$$

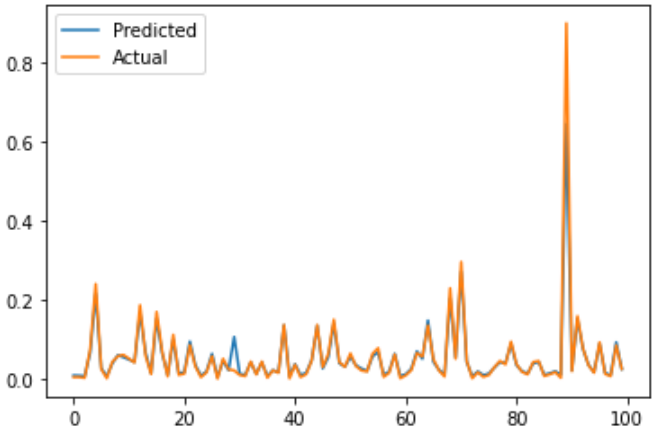
$$RMSE = \sqrt[2]{\frac{\sum (y_{actual} - y_{predicted})^2}{n}} \quad (4)$$

V. Results and Discussion

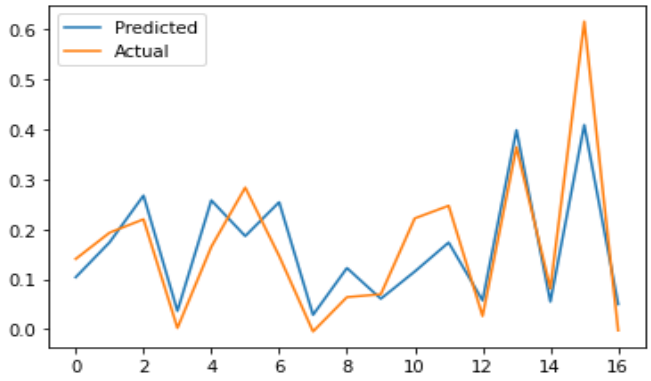
Below, Fig. 2 represents the deviation between actual and predicted effort values on all datasets, where the X-axis represents the record number. The Y-axis represents the effort of the record. Fig. 2(a), 2(d) on Cocomo81 and Maxwell show a notifiable difference in peak effort values. The values are closer to the China dataset in Fig. 2(b). Fig. 2(c) and 2(f) show a constant gap between actual and predicted values. Fig. 2(e) on Albrecht shows a considerable difference.



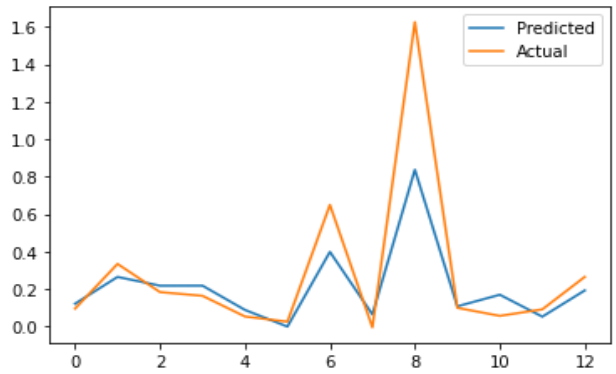
(a)



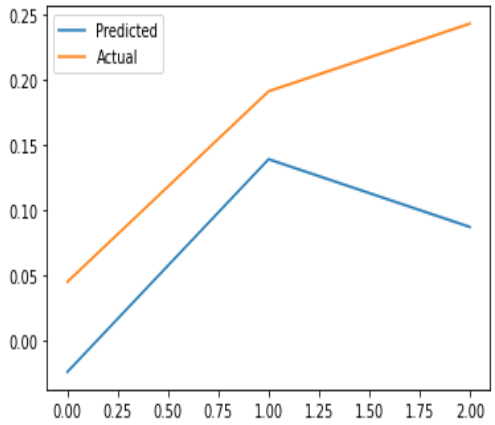
(b)



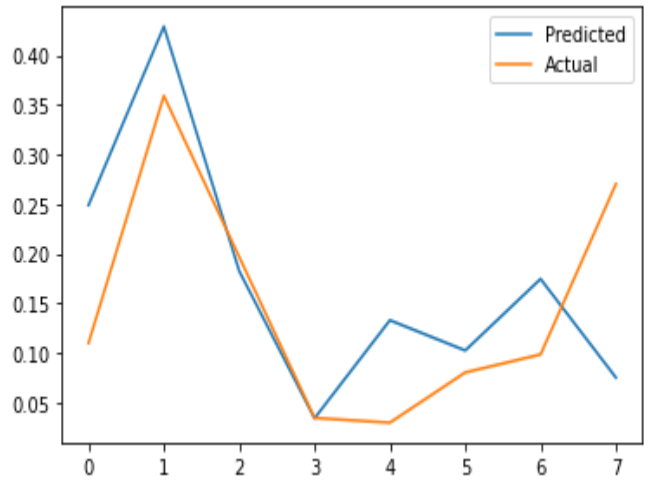
(c)



(d)



(e)



(f)

Fig. 2. (a) Cocomo81 actual vs. predicted effort (b): China's actual vs. predicted effort (c) Desharnais actual vs. predicted effort (d) Maxwell actual vs. predicted effort (e) Kemerer actual vs. predicted effort (f) Albrecht actual vs. predicted effort

Fig. 2(a) represents a line graph drawn to show the deviations between actual effort and the predicted effort by our proposed model in the COCOMO81 dataset. This graph shows a noticeable difference at the peak points.

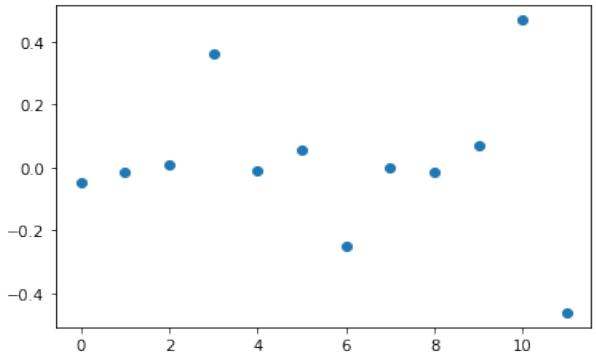
Fig. 2(b) represents a line graph drawn to show the deviations between actual effort and the predicted effort by our proposed model in the China dataset. We can see that the predicted line has come close to the actual line in many places.

Fig. 2(c) represents a line graph drawn to show the deviations between actual effort and the predicted effort by our proposed model in the Desharnais dataset. In this graph, records 10 and 11 have a significant deviation, whereas records 12 to 14 have the least deviation.

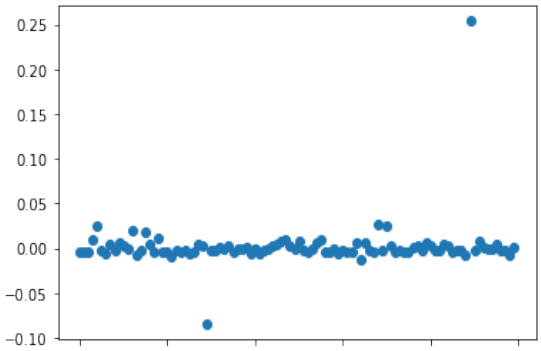
Fig. 2(d) represents a line graph drawn to show the deviations between actual effort and the predicted effort by our proposed model in the Maxwell dataset. This graph shows a noticeable difference at the peak points.

Fig. 2(e) represents a line graph drawn to show the deviations between actual effort and the predicted effort by our proposed model in the Kemerer dataset. As the test set records are meager, they show a significant deviation, but the deviation range is 0.05.

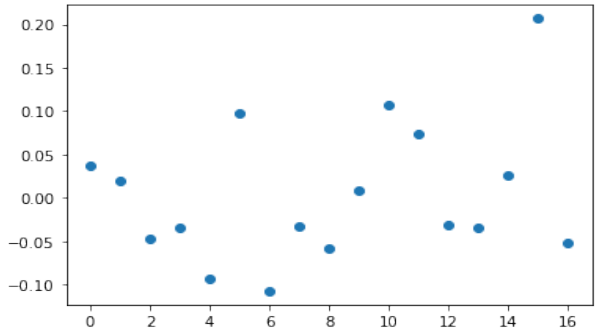
Fig. 2(f) represents a line graph drawn to show the deviations between actual effort and the predicted effort by our proposed model in the Albrecht dataset, where the X-axis represents the record number and the Y-axis represents the effort of the record. Fig. 3 represents the residuals graphs between actual and predicted effort values on all datasets. Fig. 3(a) and 3(d) on Cocomo81 and Maxwell shows a notifiable difference in peak effort values. Fig. 3(b) of the China dataset shows values closer to 0 ("zero"). Fig. 3(c) and 3(f) show a constant gap between actual and predicted values. There is a significant difference in Albrecht's Fig. 3(e).



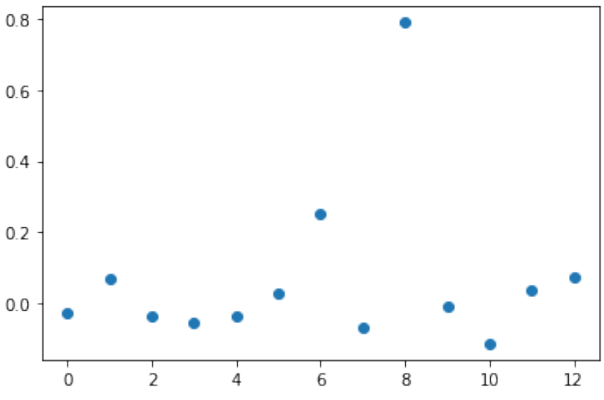
(a)



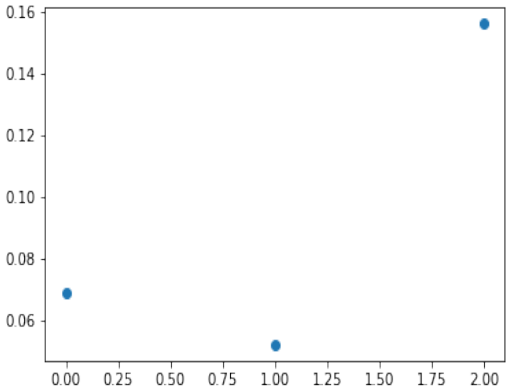
(b)



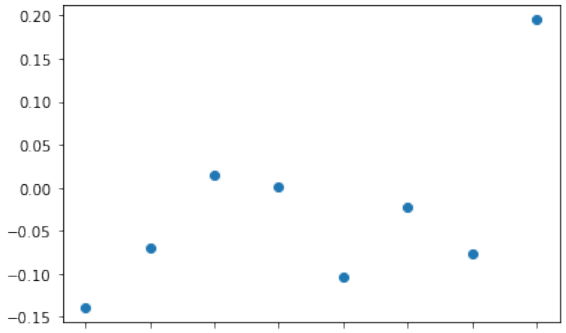
(c)



(d)



(e)



(f)

Fig. 3. (a) Cocomo81 prediction residuals (b) China prediction residuals (c) Desharnais prediction residuals (d) Maxwell prediction residuals (e) Kemerer prediction residuals (f) Albrecht prediction residuals

The above Fig. 3(a) represents a graph that shows the residuals between actual effort and the predicted effort of the data records of the COCOMO81 dataset ranging from-0.4 to +0.4, and most of the data points are present in the range of-0.2 to +0.2.

The above Fig. 3(b) represents a graph that shows the residuals between actual effort and the prediction effort of the data records of the China dataset ranging from-0.10 to +0.25. In the presented graph, most data points are nearer to 0, indicating that the proposed model is working much more efficiently in the China dataset.

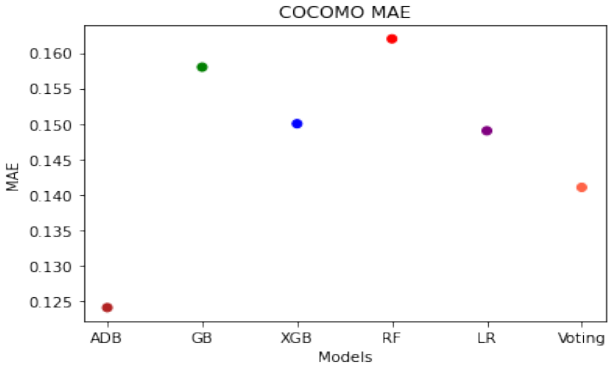
The above Fig. 3(c) represents a graph that shows the residuals between actual effort and the predicted effort of the data records of the Desharnais dataset, ranging from-0.10 to +0.20. In this graph, most of the data points are below point 0. That means the proposed model predicted values are less than the actual values.

Fig. 3(d) shows a graph of the residuals between actual effort and predicted effort of the Maxwell dataset data records, ranging from -0.2 to +0.8. According to this graph, the proposed model prediction is much closer to the actual values based on working on this dataset.

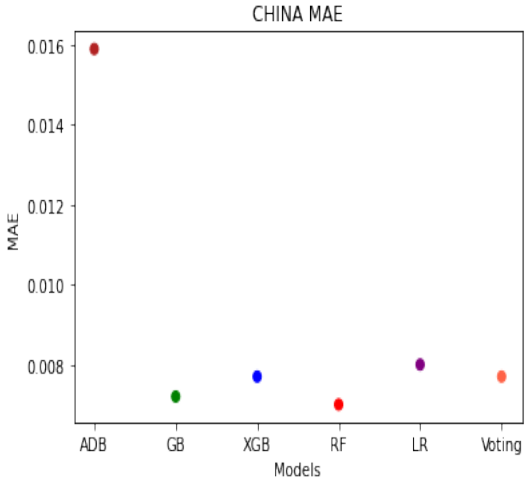
Fig. 3(e) depicts a graph displaying the residuals between actual effort and predicted effort of the Kemerer dataset data records, ranging from 0.05 to +0.16.

The above Fig. 3(f) represents a graph that shows the residuals between actual effort and the predicted effort of the data records of the Albrecht dataset ranging from-0.15 to +0.20.

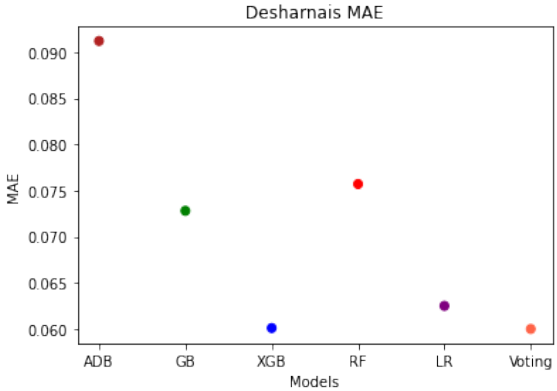
Below, Fig. 4 shows the bar plots of all the implemented models, representing the mean absolute error on all six datasets. Fig. 4(a) the voting model outperformed GB, XGB, RF, and LR except for ADB. Fig. 4(b) shows that, except for RF, voting showed less residual than all others. Fig. 4(c), (d), and (f) voting models are reliable. From all the above comparisons, we concluded that voting is a constant performer. On all datasets, the models behave randomly, whereas voting shows an upvote constantly.



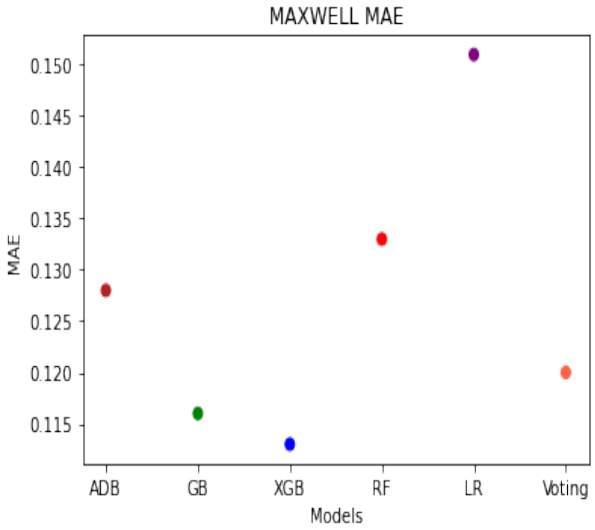
(a)



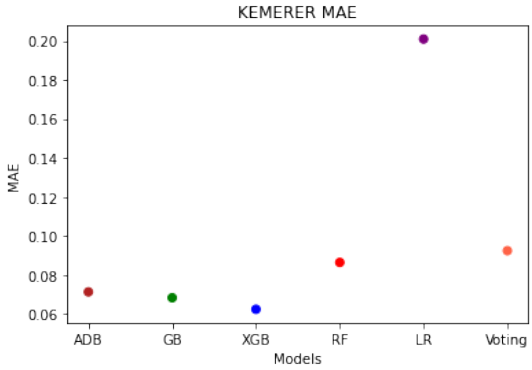
(b)



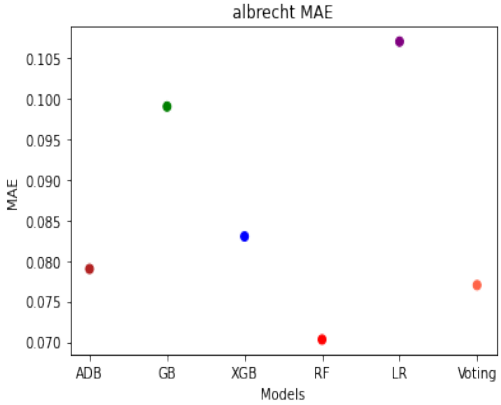
(c)



(d)



(e)



(f)

Fig. 4. (a) COCOMO81 mean absolute error (b) CHINA mean absolute error (c) DESHARNAIS mean absolute error (d) MAXWELL mean absolute error (e) KEMERER mean absolute error (f) ALBRECHT mean absolute error

The graphical representation of Mean Absolute Error for various models that are worked on the COCOMO dataset is shown in Fig. 4(a), with the ADB model giving the slightest error followed by voting and the Random Forest giving the highest error among the models presented.

Fig. 4(b) shows a graphical representation of the Mean Absolute Error for various models tested on the CHINA

dataset. The RF model produces the lowest error, and the ADB produces the highest error.

Fig. 4(c) shows a graphical representation of the Mean Absolute Error for various models tested on the Desharnais dataset, with the Voting and XGB models producing the lowest error and the ADB having the highest error.

The graphical representation of Mean Absolute Error for various models that are worked on the MAXWELL dataset is shown in Fig. 4(d), with the XGB model giving a minor error and the LR model giving the highest error among the models presented.

Fig. 4(e) shows a graphical representation of the Mean Absolute Error for various models tested on the Kemerer dataset. The XGB model produces the lowest error, and the LR model produces the highest error.

Fig. 4(f) shows a graphical representation of the Mean Absolute Error for various models tested on the Albrecht dataset. The RF model produces the lowest error, and the LR model produces the highest error.

We normalized the literature results and compared them with the obtained model's results (see Tables I-VI)

TABLE I. COCOMO81 DATASET

Model	MAE	MSE	RMSE
Linear Regression	0.1499	0.0393	0.1984
AdaBoost	0.1248	0.0395	0.1989
Random Forest	0.1627	0.0680	0.2608
Gradient Boosting	0.1587	0.0662	0.2574
XGB	0.1509	0.0680	0.2665
Ali et al. [10]	0.1652	-	0.4322
Voting	0.1466	0.0527	0.2297

TABLE II. CHINA DATASET

Model	MAE	MSE	RMSE
Linear Regression	0.0080	0.0005	0.0231
AdaBoost	0.0159	0.0013	0.0363
Random Forest	0.0070	0.0008	0.0286
Gradient Boosting	0.0072	0.0008	0.0295
XGB	0.0077	0.0009	0.0308
Hosni et al. [14]	0.0099	-	-
Voting	0.0077	0.0007	0.0270

TABLE III. DESHARNAIS DATASET

Model	MAE	MSE	RMSE
Linear Regression	0.0625	0.0070	0.0841
AdaBoost	0.0912	0.0107	0.1037
Random Forest	0.0757	0.0095	0.0976
Gradient Boosting	0.0728	0.0075	0.0866
XGB	0.0601	0.0065	0.0790
De Carvalho et al., [2]	0.0562	0.0078	0.0880
Hosni et al. [14]	0.0664	-	-
Voting	0.0627	**0.0061**	**0.0783**

TABLE IV. MAXWELL DATASET

Model	MAE	MSE	RMSE
Linear Regression	0.1519	0.0571	0.2390
AdaBoost	0.1287	0.0503	0.2243
Random Forest	0.1333	0.0739	0.2719
Gradient Boosting	0.1166	0.0565	0.2378
XGB	0.1131	0.0544	0.2334
Voting	0.1221	0.0555	0.2356

TABLE V. KEMERER DATASET

Model	MAE	MSE	RMSE
Linear Regression	0.2009	0.0462	0.2151
AdaBoost	0.0714	0.0085	0.0922
Random Forest	0.0865	0.0074	0.0865
Gradient Boosting	0.0684	0.0066	0.0815
XGB	0.0625	0.0079	0.0893
Ali et al. [10]	0.1113	-	0.2200
Hosni et al. [14]	0.0866	-	-
Voting	0.0925	0.0160	0.1031

TABLE VI. ALBRECHT DATASET

Model	MAE	MSE	RMSE
Linear Regression	0.1078	0.1977	0.1406
AdaBoost	0.0790	0.0113	0.1064
Random Forest	0.0786	0.0077	0.0878
Gradient Boosting	0.0995	0.0168	0.1298
XGB	0.0835	0.0115	0.1073
Ali et al. [10]	0.0856	-	0.1196
Voting	**0.0775**	0.0099	0.0996

Our work includes testing the voting regressor on six datasets. From the above tables, observations in all datasets voted on, showed excellent performance in minimizing the actual and predicted effort error. On the COMO81 dataset, absolute error is the minimum for voting, and squared errors are minor for linear regression. On COCOMO81, China, Desharnais, Kemerer, Maxwell, and Albrecht had excellent performances. Finally, we concluded that all dataset implementations support voting, which makes voting more reliable and robust. Voting followed by linear regression shows that the datasets have a linear relationship between the attributes of the projects.

VI. Conclusion

We studied various existing research papers on software effort estimation in this work. In the early days, we relied on many conventional approaches, considering the line of codes, functional points, CPM and PERT, etc., or merely relying on the people's judgment that has ample experience in software project effort determination. Because extensive developments in project building consider multiple parameters in every project, these techniques might not be feasible anymore with rapid results in software projects. And at the same time, machine learning has gained momentum in recent decades in various domains. And there is some work taking place in software engineering through machine learning. Therefore, our work aims to provide a robust machine learning model for effort calculation. We successfully used the machine learning ensembling concept to predict software development efforts. We considered every parameter for the effort estimation. Based on our research, the ensembling of models outperformed other single models. We recorded a lower error rate from the ensemble model comparatively. The average of different predictors positively impacted the output, which shows the vital role played in optimizing software effort estimation in the machine learning area. The input dataset dramatically affects how well the machine learning algorithm works, and in our work, models performed very well with our datasets.

References

[1] Ramesh, M. R., & Reddy, C. S. (2016). Difficulties in software cost estimation: A survey. International Journal of Scientific Engineering and Technology, 5(1), 10-13.

[2] Hareton, L., and Zhang F: “Software Cost Estimation”, Department of Computing, Hong Kong Polytechnic University.http://paginaspersonales.deusto.es/cortazar/doctorado/articulos/leung-andbook.pdf, accessed 24th Nov 2019.

[3] Rajeswari, K., &Beena, D. R. (2018). A Critique on Software Cost Estimation. International Journal of Pure and Applied Mathematics, 118(20), 3851-3862.

[4] Bull Survey, 1998, Failure Causes. http://www.itcortex.com/Stat_Failure_Cause.htm#surveys, Retrieved on 1stJune 2013.

[5] KPMG Canada, 1997, Failure Causes. http://itcortex.com/Stat_Failure_Cause.htm#surveys. Retrieved on 2ndJune 2013.

[6] Ziauddin, Shahid Kamal Tipu, ShahrukhZia, “An Effort Estimation Model for Agile Software Development”, Advances in Computer Science and its Applications (ACSA), Vol. 2, No. 1, 2012, ISSN 2166-2924.

[7] Boehm, B. W. (2017, May). Software cost estimation meets software diversity. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C) (pp. 495-496). IEEE.

[8] Santanu Kumar Rath, “Use Case Point Approach Based Software Effort Estimation using Various Support Vector Regression Kernel Methods”, January 2014.

[9] Ali BouNassif , Mohammad Azzeh , Ali Idri , and Alain Abran, Hindawi , Software Development Effort Estimation Using Regression Fuzzy Models, Computational Intelligence and Neuroscience Volume 2019.

[10] C. E. L. Peixoto, J. L. N. Audy and R. Prikladnicki, "Effort Estimation in Global Software Development Projects: Preliminary Results from a Survey," 2010 5th IEEE International Conference on Global Software Engineering, Princeton, NJ, 2010, pp. 123-127, doi: 10.1109/ICGSE.2010.22.

[11] B Rajesh Kumar Singh, A.K.Misra,”Software Effort Estimation by Genetic Algorithm Tuned Parameters of Modified Constructive Cost Model for NASA Software Projects”, International Journal of Computer Applications 59(9):22-26, December 2012.

[12] Kumar, G., & Bhatia, P. K.. Empirical assessment and optimization of software cost estimation using soft computing techniques. In Advanced Computing and Communication Technologies (pp. 117-130). Springer, Singapore, 2016.

[13] Ahmad, S. W., & Bamnote, G. R.. Whale–crow optimization (WCO)-based Optimal Regression model for Software Cost Estimation. Sādhanā, 44(4), 94, 2019.

[14] Amini, F., & Hu, G. (2021). A two-layer feature selection method using a genetic algorithm and elastic net. Expert Systems with Applications, 166, 114072.

[15] De Carvalho, H. D. P., Fagundes, R., & Santos, W. (2021). Extreme Learning Machine Applied to Software Development Effort Estimation. IEEE Access, 9, 92676-92687.

[16] Ghosh, K. K., Ahmed, S., Singh, P. K., Geem, Z. W., & Sarkar, R. (2020). Improved binary sailfish optimizer based on adaptive β-hill climbing for feature selection. IEEE Access, 8, 83548-83560.

[17] Carbonera, C. E., Farias, K., & Bischoff, V. (2020). Software development effort estimation: a systematic mapping study. IET Software, 14(4), 328-344.

[18] Chhabra, S., & Singh, H. (2020). Optimizing design of a fuzzy model for software cost estimation using particle swarm optimization algorithm. International Journal of Computational Intelligence and Applications, 19(01), 2050005.

[19] Ghatasheh, N., Faris, H., Aljarah, I., & Al-Sayyed, R. M. (2019). Optimizing software effort estimation models using the firefly algorithm. arXiv preprint arXiv:1903.02079.

[20] Wani, Z. H., & Quadri, S. M. K. (2019). An improved particle swarm optimization-based functional link artificial neural network model for software cost estimation. International Journal of Swarm Intelligence, 4(1), 38-54.

[21] Ali, A., & Gravino, C. (2019, December). Using Combinations of Bio-inspired Feature Selection Algorithms in Software Efforts Estimation: An Empirical Study. In 2019 13th International Conference on Open Source Systems and Technologies (ICOSST) (pp. 1-8). IEEE.

[22] Kodmelwar, M. K., Joshi, S. D., & Khanna, V. (2018). A deep learning modified neural network is used for efficient effort estimation. Journal of Computational and Theoretical Nanoscience, 15(11-12), 3492-3500.

[23] Desai, V. S., & Mohanty, R. (2018, October). ANN-Cuckoo optimization technique to predict software cost estimation. In 2018 Conference on Information and Communication Technology (CICT) (pp. 1-6). IEEE.

[24] Langsari, K., & Sarno, R. (2018). Optimizing effort parameter of COCOMO II using particle swarm optimization method. Telkomnika, 16(5), 2208-2216.

[25] Hosni, M., Idri, A., Abran, A., & Nassif, A. B. (2018). On the value of parameter tuning in heterogeneous ensembles effort estimation. Soft Computing, 22(18), 5977-6010.

[26] Goyal, S., & Parashar, A. (2018). Machine learning application to improve COCOMO model using neural networks. International Journal of Information Technology and Computer Science (IJITCS), 3, 35-51.

[27] Padhy, N., Singh, R. P., & Satapathy, S. C. (2018). Software reusability metrics estimation: algorithms, models and optimization techniques. Computers & Electrical Engineering, 69, 653-668.

[28] 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.

[29] Shekhar, S., & Kumar, U. (2016). Review of various software cost estimation techniques. International Journal of Computer Applications, 141(11), 31-34.

[30] Venkataiah, V., Mohanty, R., Pahariya, J. S., & Nagaratna, M. (2017). Application of ant colony optimization techniques to predict software cost estimation. In Computer Communication, Networking and Internet Security (pp. 315-325). Springer, Singapore.

[31] Nassif, A. B., Azzeh, M., Capretz, L. F., & Ho, D. (2016). Neural network models for software development effort estimation: a comparative study. Neural Computing and Applications, 27(8), 2369-2381.

[32] Miandoab, E. E., & Gharehchopogh, F. S. (2016). A novel hybrid algorithm for software cost estimation based on cuckoo optimization and k-nearest neighbors algorithms. Engineering, Technology & Applied Science Research, 6(3), 1018-1022.

[33] Dizaji, Z. A., & Gharehchopogh, F. S. (2015). A hybrid of ant colony optimization and chaos optimization algorithms approach for software cost estimation. Indian Journal of science and technology, 8(2), 128.

[34] Mahmood, Y., Kama, N., Azmi, A., Khan, A. S., & Ali, M. (2021). Software Effort Estimation Accuracy Prediction of Machine Learning Techniques: A Systematic Performance Evaluation. *ArXiv*. https://doi.org/10.48550/arXiv.2101.10658

[35] Marco, R., Ahmad, S. S. S., & Ahmad, S. (2022). Bayesian hyperparameter optimization and Ensemble Learning for Machine Learning Models on software effort estimation. International Journal of Advanced Computer Science and Applications, 13(3).