Innovations in Systems and Software Engineering (2022) 18:283–299
https://doi.org/10.1007/s11334-020-00379-y

S.I. : ACITSEP

A pragmatic ensemble learning approach for effective software effort estimation

P. Suresh Kumar[1] · H. S. Behera[1] · Janmenjoy Nayak[2] · Bighnaraj Naik[3]

Received: 15 September 2020 / Accepted: 11 November 2020 / Published online: 26 January 2021

Abstract
The immense increase in software technology has resulted in the convolution of software projects. Software effort estimation is fundamental to commence any software project and inaccurate estimation may lead to several complications and setbacks for present and future projects. Several techniques have been following for ages of the software effort estimation. As the application of software is extensively increased in its size and complexity, the traditional methods aren't adequate to meet the requirements. To achieve the accurate estimation of software effort, in this paper, a gradient boosting regressor model is proposed as a robust approach. The performance is compared with regression models such as stochastic gradient descent, K-nearest neighbor, decision tree, bagging regressor, random forest regressor, Ada-boost regressor, and gradient boosting regressor by employing COCOMO'81 containing 63 projects and CHINA of 499 projects. The regression models are evaluated by the evaluation metrics such as MAE, MSE, RMSE, and R^2. From the results, it is evident that the gradient boosting regressor model is performing well by obtaining an accuracy of 98% with COCOMO'81 and 93% with CHINA dataset. The proposed method significantly performs better than all regression models used in comparison with both the datasets.

Keywords Software effort estimation · Ensemble learning · Gradient boosting · Machine learning · COCOMO

Abbreviations

SGD	Stochastic gradient descent
KNN	K-nearest neighbor
DT	Decision trees
BR	Bagging regressor
RFR	Random forest regressor
ABR	Ada-Boost regressor
GBR	Gradient boosting regressor
MRE	Magnitude relative error
MMRE	Mean magnitude of relative error
RMSE	Root mean square error
MAE	Mean absolute error
MdMRE	Median magnitude of relative error
MSE	Mean square error

✉ P. Suresh Kumar
reshu.suri@gmail.com

H. S. Behera
hsbehera_india@yahoo.com

Janmenjoy Nayak
mailforjnayak@gmail.com

Bighnaraj Naik
mailtobnaik@gmail.com

1 Department of Information Technology, Veer Surendra Sai University of Technology, Burla 768018, India

2 Department of CSE, Aditya Institute of Technology and Management (AITAM), Tekkali 532201, India

3 Department of Computer Application, Veer Surendra Sai University of Technology, Burla 768018, India

1 Introduction

Software effort estimation deals with estimating the software effort, which is essential to build a software project [1]. It considers estimates of schedules, a probable sum of cost and manpower essential in building a software project [2]. The software effort estimation approach will foretell the practical quantity of effort required to maintain and develop the software from undetermined, insufficient, noisy, and inconsistent data input [3]. The software effort estimation has drawn attention since the mid-1970s [4]. It was of utmost importance since inaccuracies in the predictions lead to complicated results such as overestimation and dissipation of resources, and underestimation may cause overabundance in their planned budgets [5]. With the boundless extension of software technology, the significance and bulkiness of the software are extended in size, collaterally its complexity also increased to predict the software project's effort accurately.

To estimate the software effort, the techniques that comprehend are algorithmic models, expert judgment, estimation by analogy, and machine learning. The algorithmic models comprise COCOMO, function points, SLIM, and use case points. Expert judgment is a method that analyzes and utilizes the expert's experiences in the estimation of the project. Estimation by analogy refers to that homogeneous historical projects which are compared to the current developmental models. Machine learning techniques are renowned for the last two decades in estimating software effort [6].

Moving further into the conceptualization of machine learning, the methods entail in machine learning builds regression models, which make use of prior projects and are subsequently employed to estimate the software project's effort. As software projects convolutions are increasing, the statistical methods and traditional parametric models do not often seem to be coherent to represent the correlation between the project features and the software effort [7]. So with the immense progress of software effort estimation practice, there should be techniques that can compute the effort of abruptly changing software, which keeps on updating the programming tools and skills. In this scenario, machine learning rather than traditional methods are preferable because those have the potentiality to access the historical data and learn from it and can adapt to the wide variations that occur in a software project [8]. Hence, such techniques are efficient to deal with the complex data and possessing high accuracy.

Periodically, there would be difficulty in the machine learning techniques to specify which effort estimators accomplishes best. Because, while comparing estimators based on the modified conditions, any estimator may change its place in the ranking. However, if we merge the estimates of multiple estimators, then the resulting method executes better than any single estimator. Taking the above assertion into consideration, we uncover the ensemble learning in machine learning, which makes use of multiple learning algorithms to estimate or predict better performance [9]. So, integrating ensemble learning for the software effort estimation process can lead to better accuracy in estimating the effort by making use of many algorithms and predicting the good result of all.

The primary goal of this paper is to estimate the effort of software using ensemble learning technique, and various machine learning techniques. The highlights of the research work are referenced as follows:

- Proposed gradient boosting regressor (GBR) for estimating the software effort of large-scale projects.
- Other ensemble learning methods such as Ada-Boost regressor (ABR), and bagging regressor (BR) and machine learning models such as stochastic gradient descent (SGD), *K*-nearest neighbor (KNN), decision trees (DT) are used to compare the performance with the proposed method.
- Simulation is carried out by using datasets such as COCOMO'81, and CHINA.
- Considered four performance metrics to analyze the performance of each regressor.

The rest of the paper is structured as follows: Section 2 shows the literature review of techniques in estimating the software project's effort. Section 3 describes the proposed model along with the framework. Section 4 presents the Experimental setup along with describing the datasets COCOMO'81, and CHINA, performance measures and the environmental setup. Section 5 shows the result analysis accompanying plots; Sect. 6 presents the conclusion of the work.

2 Literature review

Estimating software effort has always remained a challenging task for machine learning researchers. Singal et al. [10] applied differential evaluation (DE) algorithm on COCOMO II and COCOMO models to estimate the effort of a software. In this experiment, COCOMO81 and NASA'93 datasets were implemented and the evaluation metric used for comparison is MMRE. The effectiveness of the differential evolution algorithm was investigated in enhancing the parameter values for traditional algorithmic models such as COCOMO II and COCOMO. The cost driver values were upgraded by implementing the DE approach for both the models, which resulted in boosting the accuracy in software effort estimation.

The accurate effort estimation in agile software development projects can help in the planning of a sprint, which leads to optimal results. Malgonde and Chari [11] experimented with seven algorithmic approaches such as support vector machine, ridge regression, artificial neural network, *K*-nearest neighbor, decision tree, linear regression, and Bayesian networks to determine the method that gives better accuracy in estimating the software effort, and uniformly neither of them outperformed. So, they implemented an ensemble-based approach to predict the effort, the data on which they executed are from 24 software development projects. The considered performance metrics for evaluation are mean absolute error (MAE), mean balanced error (MBE), and root mean square error (RMSE). The proposed ensemble-based algorithm outperformed the other similar approaches in predicting the software effort.

Abdelali et al. [12] built a random forest (RF) model and experimentally optimized the performance by modifying the key parameters to estimate the accurate software effort. The datasets used are ISBSG, Tukutuku, and COCOMO. The evaluation was handled through the 30% hold-out validation method. To evaluate and determine the well-performed technique, three performance metrics such as median magnitude of relative error (MdMRE), mean magnitude of relative error (MMRE), and Pred (0.25) are used. The obtained RF model is collated with the classical regression tree and the results proved that the enhanced RF model performed well than the regression tree model.

Fuzzy models in predicting software effort are experimented by Nassif et al. [13]. Three fuzzy logic models were implemented and compared in estimating software effort and in aid of designing of these three fuzzy logic models namely Sugeno, Mamdani with constant output, and Sugeno with linear output, regression analysis were carried out. The dataset utilized for training and testing the models is ISBSG dataset and the performance metrics used for evaluation are MAE, MBRE, mean inverted balance relative error, standardized accuracy, and Scott–Knott. The comparative analysis among the fuzzy models designed in assistance with regression analysis showed that the Sugeno fuzzy model with linear output resulted better.

Pospieszny et al. [14] proposed implementations with the machine learning algorithms in predicting the effort of software projects effectively by averaging the ensemble of machine learning algorithms including neural networks, generalized linear models, support vector machines with cross-validation. The methods are tested with ISBSG dataset and verified with metrics such as MAE, MMRE, mean squared error (MSE), RMSE, MMER, mean balanced relative error (MBRE), and PRED in evaluating the proposed model. The efficacy of the model helped in predicting the software effort accurately within the time duration.

The other works of estimating software effort using different techniques by employing well-known datasets along with the real-time industry projects and the performance metrics evaluated to determine the best model that predicts the accurate software effort are shown in Table 1.

3 Proposed methodology

The proposed method's framework is the inclusion of various independent methods. Primarily, data collection is done as the initial step of our framework and the succeeding steps are data preprocessing, data cleaning, and data visualization. In our framework, we compared various machine learning techniques such as SGD, KNN, DT, BR, RFR, ABR, and GBR are considered for software effort estimation. The data are split into training and testing in the ratio of 80:20. Our proposed framework is represented in Fig. 1.

3.1 Gradient boosting

In ensemble learning, the sets of learning machines are trained and combined to execute a similar task, to enhance predictive performance. Ensemble learning techniques have drawn the attention of software effort estimation communities as they constantly contributing better performance over single learning techniques [15]. Gradient boosting is an ensemble learning algorithm of machine learning. The primary concept of boosting is to append the new base models constantly to the ensemble of learning machines. At every following iteration of the learning process, a new weak base-learning model is set and trained based upon the errors of the earlier iterations of the entire ensemble. Gradient boosting was developed by Friedman [16]. It solves the minimization problem by using a gradient descent method and develops a model for prediction in the configuration of an ensemble of learning machines that includes weak prediction models consistently with decision trees.

Gradient boosting regressor is the abstraction of gradient boosting and entails a weak learner, a loss function, and an additive model. Loss function to be optimized, and it may be differentiable depending upon the kind of problem being solved, which is a prerequisite of gradient boosting. Decision trees are utilized as the weak learners as they permit their outputs to be added, also allowing the next immediate model outputs to be appended and to rectify the residuals in the predictions. The greedy approach is used to build the trees that chooses the best split to minimize the loss. A gradient descent method is utilized to minimize the loss by appending the decision trees. After evaluating the loss, a new tree adjoins to employ the gradient descent procedure to minimize the loss and the output of the new tree is subsequently appended to the existing ensemble of trees to improve the accuracy of final output [17].

Table 1 Literature study in software effort estimation

Contribution	Author proposed method	Datasets	Evaluation factors	Performance	Limitation	References
Compared various machine learning techniques such as Naïve Bayes, Random Forest, Artificial Neural Networks, and Support vector machine. ANN performed well compared to all other techniques	NB, RF, ANN, and SVM	Student dataset	Effort	Effort—39.148	Only tested with some existing ML methods	[30]
COCOMO II model is compared with COCOMO II-Cuckoo, COCOMO II-Whale optimization	COCOMO II model with humpback Whale optimization, Cuckoo	NASA 93	MMRE, MRE	MMRE—50.6122	Less accuracy	[31]
Estimating software effort using NN	BPNN	COCOMO 81 (63), Desharnais (81)	MMRE		High complexity and less accuracy	[32]
Improvised Genetic Algorithm designed for optimizing COCOMO II model's coeffecients to enhance effort estimation	Genetic algorithms	COCOMO II	MMRE, MRE	MMRE—33.96	The COCOMO II model is dependent on only time constraint and ignores all other requirements	[33]
Introduced optimization technique called OIL and compared performance results of CART-DE, and CART-FLASH	Optimized inductive learning, CART-DE, CART-FLASH	Kemerer, albrecht, ISBSG 10, Finnish, Miyazaki, Maxwell, Desharnais, Kitchenham, China	AR, MRE, MAE, SA	MRE—59%	Less accuracy and dependent parameter optimization	[34]
Compared various fuzzy logic models such as Mamdani, Sugeno with constant output, and sugeno with linear output in regression analysis	Regression fuzzy logic	ISBSG	MAE, MBRE, MIBRE, SA, ME	MAE—3613.7, MBRE—181.8, MIBRE—62.5, SA—38.5	Low estimation in major cases	[13]
Optimized hybrid model based on COCOMO	COCOMO	Real software projects	MMRE, PRED	MMRE—0.21, PRED—0.75	Less accuracy	[35]

Table 1 continued

Contribution	Author proposed method	Datasets	Evaluation factors	Performance	Limitation	References
Random forest is applied by varying various parameters and the results are compared with regression trees.	Random forest	ISBSG R8, Tukutuku and COCOMO	Pred(0.25), MMRE and MdMRE	Pred(0.25)—40, MMRE—1.29, MdMRE—0.37	Less accuracy	[12]
Compared various machine learning models such as SVM, MLP, GLM, and Ensemble average approach	SVM, MLP, GLM, Ensemble averaging	ISBSG	ME, MAE, MSE, RMSE, MMRE, PRED(0.25), PRED(0.3), MMER, MBRE	ME—0.02, MAE—0.19, MSE—0.07, RMSE—0.27, MMRE—0.13, PRED(0.25)—76.91%, PRED(0.3)—81.19%, MMER—0.47, MBRE—0.16	Less accuracy, moderate error rate	[14]
Examined the performance of DT, SGB, RF with existing approaches.	DT, SGB, RF	21 software projects	MMER, MdMER, $PRED_{25}$ (50) (75) (100)	MMER—0.1632, MdMER—0.1151, $PRED_{25}$—85.71, PRED(50)(75)(100)—95.23,	Use of Small dataset and less accuracy	[18]
Predicting software project's effort using function point, SVM and ANN MODELS	SVM, ANN	Real software projects	RMSE, RAE, RRSE, MAE	RMSE—5.893, RAE—9.158%, RRSE—13.263%, MAE—3.7723	Use of traditional models and obtained moderate error rate	[22]
Multi-objective software effort estimation	Confidence guided effort estimator	CHINA, Desharnais, Finnish, Miyazaki, Maxwell	MAE, SA	SA—0.90	Less accuracy	[36]
Estimating software effort through generalized regression NN	Generalized regression NN	COCOMO	MMRE, MdMRE	MMRE—2.1442, MdMRE—78.47	Less accuracy	[15]

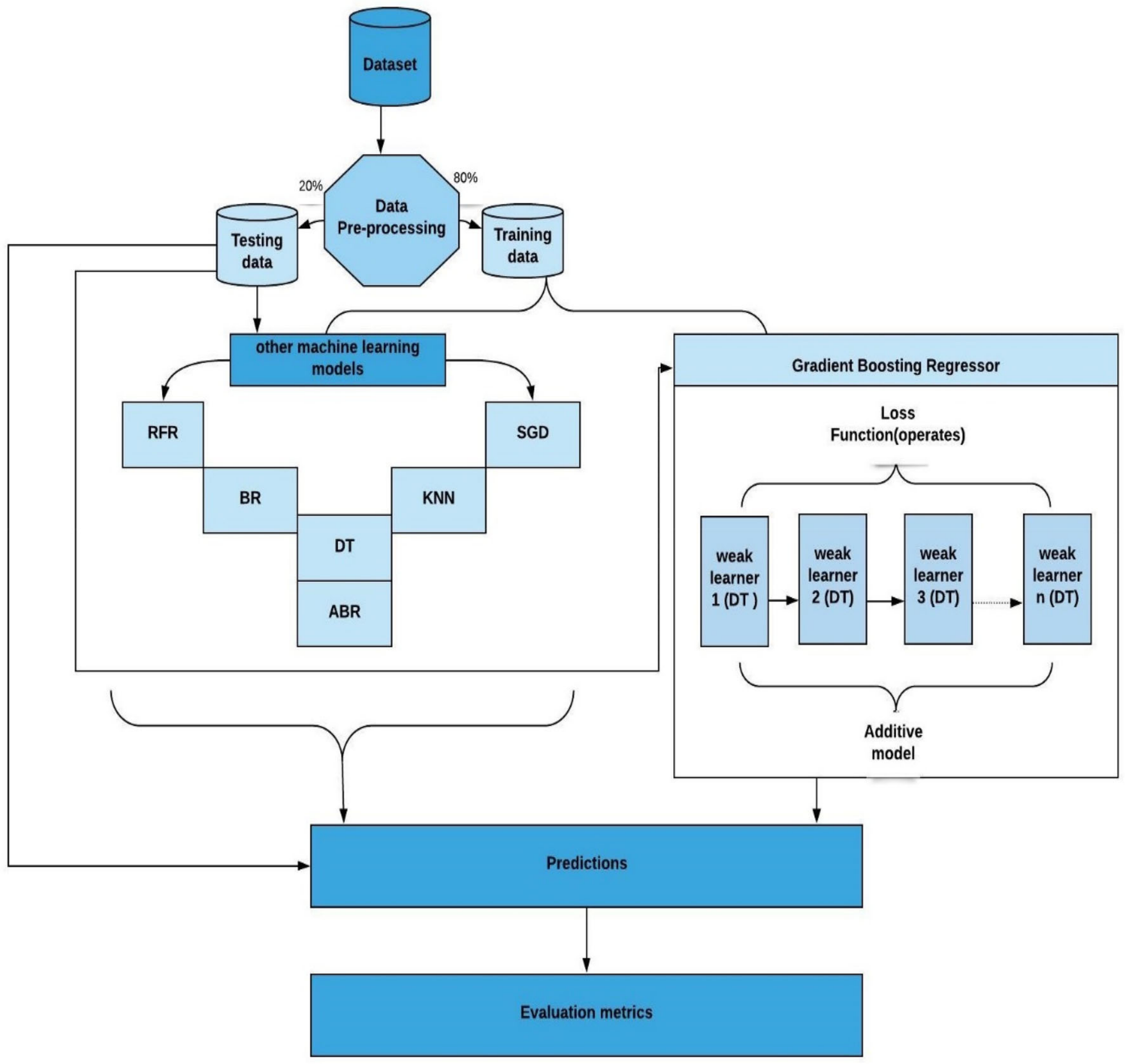

Fig. 1 The framework of the proposed model

The procedure of the proposed Gradient boosting regressor is as follows:

Gradient boosting algorithm constructs trees according to the input values and calculates the residuals based on the observed values and improves the accuracy by considering the predicted outputs of the previous trees. In the above algorithm, input considered are independent variables and the dependent variable of all the samples and loss function is considered to calculate the residuals. In step 1, $F_0(x)$ function is initialized with a value obtained by the summation of the loss function of all the samples. In step 2, the residuals are calculated by taking the derivative of the loss function to that of the predicted values for every tree and a regression tree is fitted for all the residuals. The output is calculated for each leaf in the tree by minimizing the summation of their residuals and that is the average of residuals in each leaf, and are stored. Now, the model is updated by adding the summation of the output to the previous output, and this iteration continues for h number of trees until we obtain the estimated accuracy as described in step 3.

The framework of the proposed model is presented in Fig. 1.

4 Experimental setup

COCOMO'81, and CHINA datasets were employed to evaluate and compare the software effort using various regression models. The COCOMO dataset has been widely used in research studies to estimate the effort of software by employing traditional algorithms and also with the machine learning

Algorithm: Gradient boosting regressor model

Input: The input taken is $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function considered is $G(y_i, F(x))$ where x_i represents input variables and y_i represents the corresponding output variable i.e., dependent variable and the differentiable loss function is $G(y_i, F(x))$

The loss function is defined as $\frac{1}{2}(\text{Actual} - \text{Predicted})^2$

1. A function $F_0(x)$ is initialized with a value by minimizing the summation of the loss function $G(y_i, \beta)$ by considering the summation of the loss function and $\underset{\beta}{\text{argmin}}$ is used to minimize the summation of loss function consisting of y_i as observed values and β as predicted values. i.e., we are initializing the model with a constant value.

$$F_0(x) = \underset{\beta}{argmin} \sum_{i=1}^{n} G(y_i, \beta) \tag{1}$$

2. Consider the decision trees i.e., weak learners in our case; starting from first tree (h) to last tree (H) for $h = 1$ to H:
   (a) Computation of residuals is carried out by derivative of loss function with respect to the derivative of predicted value; for every tree we are trying to build, and with every sample $(i = 1, 2, 3, \ldots)$ given by

$$q_{ih} = -\left[\frac{\partial G(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{h-1}(x)} \tag{2}$$
$$\text{for } i = 1, \ldots, n$$

   The residuals are called as pseudo residuals, and the derivative $\frac{\partial G(y_i, F(x_i))}{\partial F(x_i)}$ is called as Gradient.

   (b) After calculating the residuals, a regression tree to be fitted for pseudo-residuals q_{ih} values and keep count of the terminal regions i.e., leaves in the trees Q_{jh} for $j = 1, \ldots, j_h$ where h is tree.

   (c) Compute the output values for each leaf in the tree by minimizing the summation of loss function by using $\underset{\beta}{\text{argmin}}$ for each sample in the particular leaves of the tree i.e., $x_i \in Q_{ij}$. The output values are the average of residuals for each leaf and are stored in β_{jh}

   For $j = 1, \ldots, j_h$ compute

$$\beta_{jh} = \underset{\beta}{\text{argmin}} \sum_{xi \in Qij} G(y_i, F_{h-1}(x_i) + \beta) \tag{3}$$

   (d) Now, update the model $F_h(x)$ by adding the previous output and summation of output residuals of the leaves of each tree to minimize the loss function. Here, υ is defined as learning rate. By adding the learning rate, the accuracy of the model can be increased.

   Update

$$F_h(x) = F_{h-1}(x) + \upsilon \sum_{j=1}^{j_h} \beta_j I(x \in Q_{jh}) \tag{4}$$

3. The loop will be continued for the no. of weak learners keep adding and the output will be the H^{th} tree which obtains the estimated level or highest accuracy Output $F_H(x)$.

Table 2 Effort multipliers and their description of COCOMO'81 dataset

Category	Effort multipliers	Description
Product	RELY	Required software reliability
	DATA	Database size
	CPLX	Product complexity
Platform	TIME	Execution time
	STOR	Main storage constraint
	VIRT	Virtual machine volatility
	TURN	Computer turnaround time
Personnel	ACAP	Analyst capability
	AEXP	Applications experience
	PCAP	Programmer capability
	VEXP	Virtual machine experience
	LEXP	Language experience
Project	MODP	Modern programming
	TOOL	Use of software tools
	SCED	Required development schedule

algorithms [18–21]. China dataset has been used in various models to estimate accurate software effort; for instance, employing in designing a model with an algorithm [22] and in comparing approaches to improve the defect and effort estimation models [23] and so on. These two datasets are publicly available in the PROMISE repository [24, 25].

COCOMO'81, proposed by Barry Boehm [26], comprising 63 software projects. There exist entire 17 numeric attributes, among them 15 are effort multipliers divided into four clusters such as product, platform, personnel, and project. The effort multipliers under their respective cluster are shown in Table 2. The two attributes other than the 15 effort multipliers are lines of code and actual development effort. The effort is estimated in person-months. The development effort is the dependent variable whereas, effort multipliers and lines of code are the independent variables.

CHINA dataset is of Chinese software projects, and it consists of 499 projects including 19 attributes for each project and unit of measure is done through function points and effort is estimated in Person–Hours [27]. Features and their description are shown in Table 3 [28].

Table 3 Features and description of the CHINA dataset

Features	Description
AFP	Adjusted function points
INPUT	Function points (UFP) of input
OUTPUT	Function points (UFP) of external output
ENQUIRY	Function points (UFP) of external enquiry
FILE	Function points (UFP) of internal logical files or entity references
INTERFACE	Function points (UFP) of external interface added
ADDED	Added functions count
CHANGED	Function points (CFP) of changed functions
DELETED	Function points (CFP) of deleted functions
PDR_AFP	–
PDR_UFP	Normalized level 1 productivity delivery rate
NPDR_AFP	Normalized productivity delivery rate
NPDU_UFP	Productivity delivery rate
RESOURCE	Team type
DEV.TYPE	Development type
DURATION	Total elapsed time for the project
N_EFFORT	–
EFFORT	Summary work effort

4.1 Performance measures

Developing models is not sufficient and needs to be evaluated to verify the accuracy and to know how precise the models are. In this experiment, to evaluate and compare the performance of the various regression models, we adopted four performance measures such as mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and coefficient of determination (R^2). The above regression metrics are imported from sklearn package.

The MAE is the mean absolute error that defines the absolute error i.e., the average error of true values and predicted values of all the samples included. The lower the MAE value obtained, the better is the model. The mean_absolute_error function imported from the sklearn metrics package evaluates the mean absolute error, the error metric is relative to the expected value of absolute error loss. The computation of MAE [22] is shown in Eq. (5)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{5}$$

The MSE is the mean squared error which is calculated by considering the averages of squares of the difference between the actual and predicted values of all samples. The lower the MSE value obtained, the better is the model. The mean_squared_error function imported from the sklearn metrics package evaluates mean square error, the error metric for the expected value of squared error. The computation of MSE [29] is shown in Eq. (6)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6}$$

The RMSE is the root mean squared error, evaluated by calculating the square root of the MSE, and it can also be defined as the standard deviation of the residuals. Like MAE and MSE, the RMSE value should be lower for a model to perform better. The RMSE is computed by importing the function mean_squared_error from the sklearn metrics package and setting the squared parameter to false. RMSE is described in Eq. (7) [18].

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{7}$$

In Eqs. (5)–(7), y_i is the actual values and $\hat{y}_i$ is the predicted values of corresponding actual values for a total of n.

The R^2 evaluation metric is the coefficient of determination. This evaluation metric gives a manifestation that how good a model fits a given dataset. It indicates how nearer the predicted values to the actual values. The R^2 value lies between 0 and 1. If the obtained value is 1, it indicates the model fits exactly to the dataset provided and if it is negative then it defines that model doesn't fit well to the dataset. Unlike, the MAE, MSE, and RMSE, the R^2 value should be higher i.e., nearer to 1 for a model to perform better. The r2_score function imported from the sklearn metrics package computes the coefficient of determination. It represents the proportion of variance that has been described by the independent variables in the model. The equation describes the R^2 is presented in Eq. (8) [29]

$$R^2\left(y, \hat{y}_i\right) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{8}$$

y_i is the actual value and $\hat{y}_i$ is predicted value of ith sample for a total of n samples and $\bar{y}_i = \frac{1}{n}\sum_{i=1}^{n} y_i$.

4.2 Environmental setup

Experimentation is done on the system setup of Lenovo with windows 10 pro 64-bit operating system Intel i5 processor and 6 GB RAM. The designing of regression models and evaluation of statistics is done using python and various frameworks such as NumPy and pandas modules. Visualization is done by using matplotlib, and seaborn libraries. We used python's package scikit-learn to build various machine learning and ensemble models. Experimentation is carried out by using COCOMO'81 and CHINA dataset.

Table 4 Parameter settings of all considered methods w.r.t. respective datasets

Dataset	Technique	Parameter setting
COCOMO'81	SGD regressor	penalty: l1, random_state: 1, learning rate: adaptive, max_iter: 2,000,000
	K-nearest neighbors	n_neighbors: 3, weight: uniform, p: 1
	Decision tree	criterion: friedman_mse, max_depth: 2, min_samples_split: 10, random_state: 1, splitter: best, min_samples_leaf: 2
	Bagging regressor	n_estimators: 1, random_state: 1, max_samples: 50
	Random forest	n_estimators: 5, random_state: 1, criterion: mae
	Ada-boost regressor	n_estimators: 70, random_state: 1, loss: square
	Gradient boosting regressor	n_estimators: 69, random_state: 1, learning_rate: 0.3, max_leaf_nodes: 6
China	SGD regressor	loss: huber, random_state: 1, alpha: 0.05, epsilon: 2.9, average: true
	K-nearest neighbors	n_neighbors: 10, weight: uniform
	Decision tree	criterion: mse, max_depth: 9, max_leaf_nodes: 27, random_state: 1
	Bagging regressor	n_estimators: 54, random_state: 1, max_samples: 359, max_features: 12, bootstrap: True, warm_start: False
	Random forest	n_estimators: 60, random_state: 1, criterion: mae, max_depth: 12
	Ada-boost regressor	random_state: 1, n_estimators: 3, learning_rate: 0.18
	Gradient boosting regressor	loss: ls, random_state: 1, n_estimators: 96, subsample: 0.7, criterion: mae, ccp_alpha: 0.9

The COCOMO'81 and CHINA datasets comprise 63 and 499 projects, respectively. The datasets are split into 80% for training the model and 20% for testing the model. Various machine learning and ensemble learning algorithms such as stochastic gradient descent (SGD) regressor, K-nearest neighbors (KNN) regressor, decision tree (DT) regressor, bagging regressor (BR), random forest regressor (RFR), Adaboost regressor (ABR), gradient boosting regressor (GBR) were used to estimate the effort of software. For each regression model in correspondence with the particular dataset, with specified parameters that give the best score are shown in Table 4.

The evaluation metrics were computed and compared with the results obtained from different parametric settings, in the process of obtaining the highest accuracy for the regression models, and the proposed method. The evaluation of the gradient boosting model's accuracy with the different parameter settings are shown in Table 5 with the COCOMO'81 and CHINA datasets.

Table 5 shows the changes in evaluation metrics with different parameter settings applied to the gradient boosting regression model with the COCOMO'81 and CHINA dataset. The best score for the R^2 metric obtained is 0.98, and the least error value for MAE, MSE, and RMSE metrics is obtained as 184.8, 52,314.5, and 228.7, respectively, when the parameter settings are set to n_estimators: 69, learning rate: 0.3, max_leaf_nodes: 6. The parameter "n_estimators" defines the number of trees to be added to the regression model and from the table, it is evident that the increasing number of "n_estimators" increases the accuracy. The lower values were set to the "learning rate" parameter to make the model more robust, "max_leaf_nodes" parameter defines the minimum no. of leaf nodes in the tree. From the table, we can state that the minimum number of leaf nodes is required to boost the accuracy of the model.

For the proposed model, the best score of R^2 with 0.93, and the MAE, MSE, RMSE with 676.6, 3,252,196.6, and 1803.3, respectively, for CHINA dataset can be obtained with the parameter settings such as n_estimators: 96, subsample: 0.7, criterion: mae, ccp_alpha: 0.9. With the increase in "n_estimators," the accuracy is increased as "no. of trees" are appending and learning from the previous errors which result in the best score, "subsample" parameter is the fraction of samples to be considered for each tree. For the value is nearly less than 1, the results signify a robust model. 'criterion' parameter defines the split's quality, with this dataset, 'mae-criteria', results in good score than the other 'criteria'. 'ccp_alpha' parameter is utilized to reduce the cost-complexity, 'ccp_alpha' is set low according to the cost-complexity of the subtree, 'ccp_alpha' slightly less than 1 gives the best result with the CHINA dataset.

5 Result analysis

In this section, the performance of various machine learning algorithms is analyzed and evaluated using MAE, MSE, RMSE, and R^2 by employing COCOMO'81 and CHINA datasets. The statistical results calculated for each regression model concerning COCOMO'81 and CHINA datasets are tabulated in Tables 6 and 7 respectively.

Table 6 describes the performance metrics of our considered regression models in correspondence with the COCOMO'81 dataset. From the results obtained in Table 6,

Table 5 Parameter settings and performance metrics of gradient boosting regression model on COCOMO'81 and CHINA datasets

Dataset	Parameter setting	MAE	MSE	RMSE	R^2
COCOMO'81	n_estimators: 1 learning rate: 0.3 max_leaf_nodes: 6	690.6	757,082.3	870.1	0.73
	n_estimators: 20 learning rate: 0.3 max_leaf_nodes: 6	208.2	70,673.9	265.8	0.97
	n_estimators: 40 learning rate: 0.2 max_leaf_nodes: 8	304.4	286,469.2	535.2	0.89
	n_estimators: 54 learning rate: 0.2 max_leaf_nodes: 8	296.8	282,312.3	531.3	0.9
	n_estimators: 69 learning rate: 0.3 max_leaf_nodes: 6	184.8	52,314.5	228.7	0.98
CHINA	n_estimators:10 subsample:0.7 criterion: mae ccp_alpha:0.9	2247.2	20,018,975	4474.2	0.61
	n_estimators:50 subsample:0.7 criterion: mae ccp_alpha:0.9	828.9	4,414,217	2101	0.91
	n_estimators:50 subsample:0.3 criterion: mse ccp_alpha:0.9	1084.5	5,768,098	2401.6	0.88
	n_estimators:96 subsample:0.7 criterion: mae ccp_alpha:0.1	689.4	3,342,791	1828.3	0.93
	n_estimators:96 subsample:0.7 criterion: mae ccp_alpha:0.9	676.6	3,252,197	1803.3	0.93

Table 6 Performance metrics evaluated using COCOMO'81 dataset for various regression models

Regression model	MAE	MSE	RMSE	R^2
SGD	860.5	1,498,019.6	1223.9	0.47
KNN	397.9	481,104.3	693.6	0.83
DT	429	639,741.4	799.8	0.77
BR	153	109,995.3	331.6	0.96
RFR	402.6	293,674.5	541.9	0.89
ABR	229.4	69,936.2	251.7	0.97
GBR	184.8	52,314.5	228.7	0.98

Table 7 Performance metrics evaluated using CHINA dataset for various regression models

Regressor model	MAE	MSE	RMSE	R^2
SGD	2103.5	20,462,127.5	4523.5	0.6
KNN	2268.2	19,883,523.4	4459	0.61
DT	1268.7	6,325,892	2515.1	0.87
BR	903.9	5,786,313	2405.4	0.88
RFR	810.1	5,235,939.8	2288.2	0.89
ABR	1483.8	7,431,929.3	2726.1	0.85
GBR	676.6	3,252,196.6	1803.3	0.93

the order of mean absolute error of various regression models from least to highest error value is BR, GBR, ABR, KNN, RFR, DT, and SGD with values 153, 184.8, 229.4, 397.9, 402.6, 429, and 860.5, respectively. Similarly, the order of various regression models from the least to highest mean squared error values is GBR, ABR, BR, RFR, KNN, DT, and SGD with values 52,314.5, 69,936.2, 109,995.3, 293,674.5, 481,104.3, 639,741.4, and 1,498,019.6, respectively. For root mean squared error, the order of various regression models from least to highest error value is GBR, ABR, BR, RFR, KNN, DT, and SGD with values 228.7, 251.7, 331.6, 541.9, 693.6, 799.8, and 1223.9, respectively. Unlike MAE, MSE,

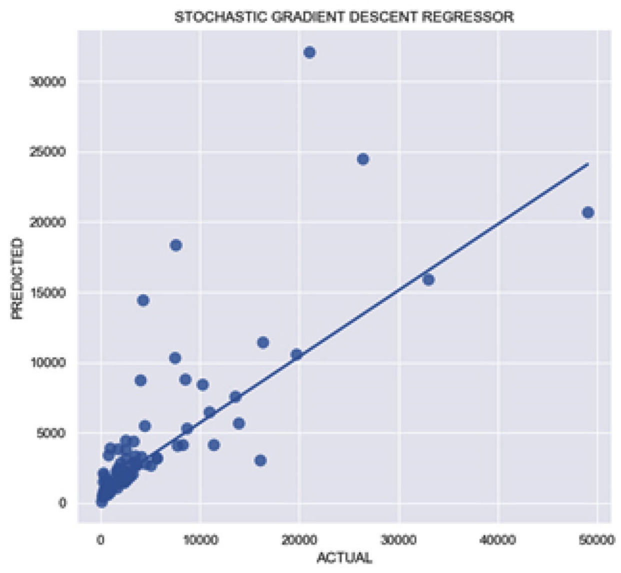

Fig. 2 Actual versus predicted using SGD in China dataset

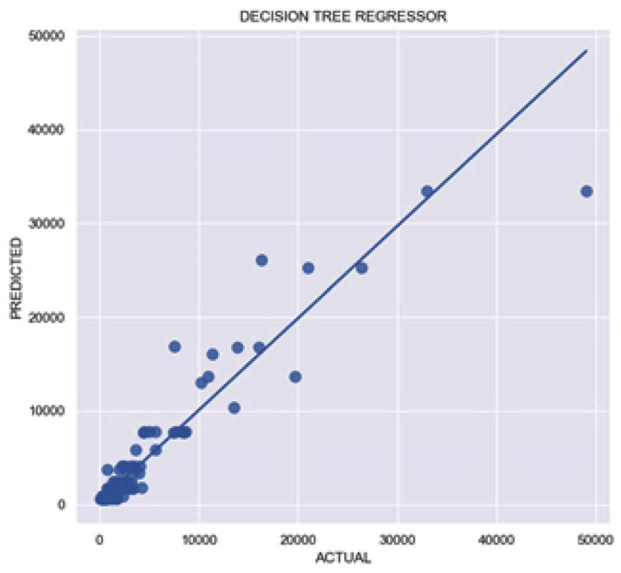

Fig. 4 Actual versus predicted using DT in China dataset

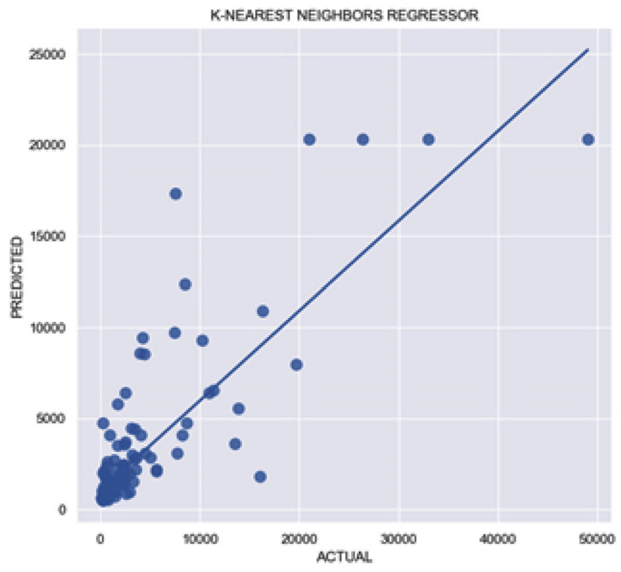

Fig. 3 Actual versus predicted using KNN in China dataset

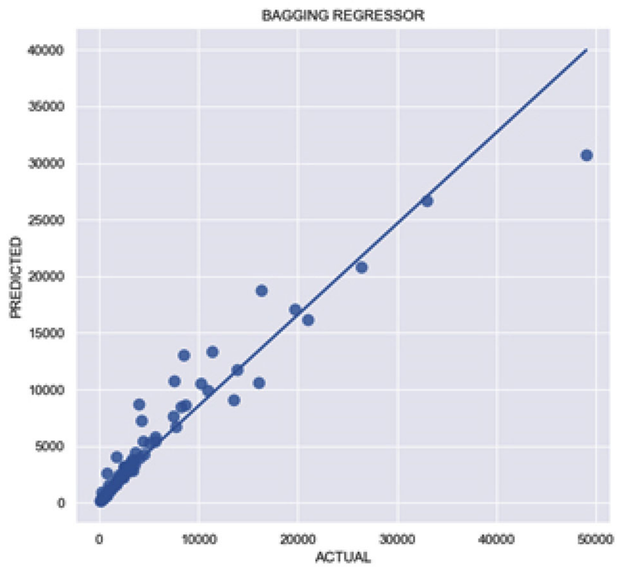

Fig. 5 Actual versus predicted using BR in China dataset

RMSE which needs to possess low values to be the accurate model, for the R^2 metric, the higher the value and closer to 1.0 is the most efficient model. From the Table 6, the order of different regression models that acquired the highest R^2 values are GBR, ABR, BR, RFR, KNN, DT, and SGD with the scores 0.98, 0.97, 0.96, 0.89, 0.83, 0.77, and 0.47, respectively. So, it is evident from the results of Table 6 that the GBR attains optimal performance when compared with other regression models using the COCOMO'81 dataset.

Table 7 represents the performance metrics of regression models using the CHINA dataset. The order of the MAE value of the regression models from low to a high value is GBR, RFR, BR, DT, ABR, SGD, and KNN with values 676.6, 810.1, 903.9, 1268.7, 1483.8, 2103.5, and 2268.2, respectively. Similarly, for the MSE values, the order of regression models from low to high are represented as GBR, RFR, BR, DT, ABR, KNN, and SGD with values 3,252,196.6, 5,235,939.8, 5,786,313.0, 6,325,892.0, 7,431,929.3, 19,883,523.4, and 20,462,127.5, respectively. Correspondingly, the RMSE values of various regression models from low to high values are in the order of GBR, RFR, BR, DT, ABR, KNN, and SGD with the values 1803.3, 2282.2, 2405.4, 2515.1, 2726.1, 4459.0, and 4523.5, respectively. For R^2 metrics, the regression models with the highest

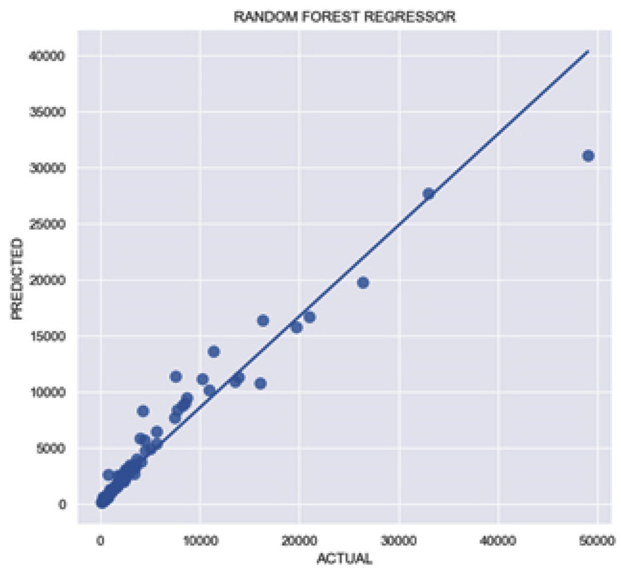

Fig. 6 Actual versus predicted using RFR in China dataset

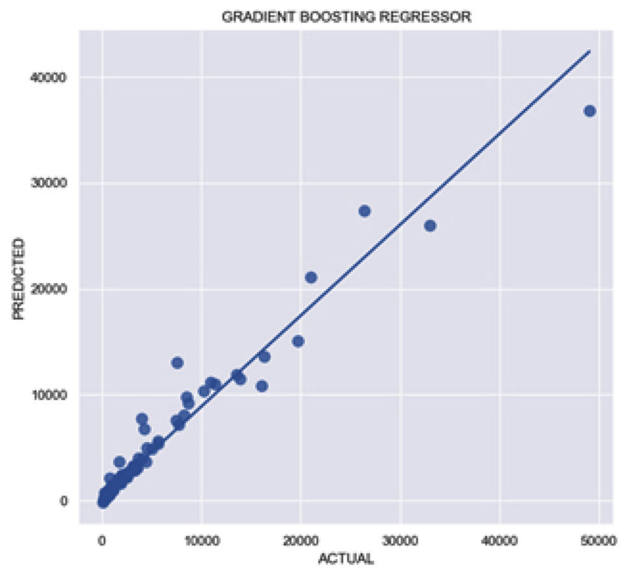

Fig. 8 Actual versus predicted using GBR in China dataset

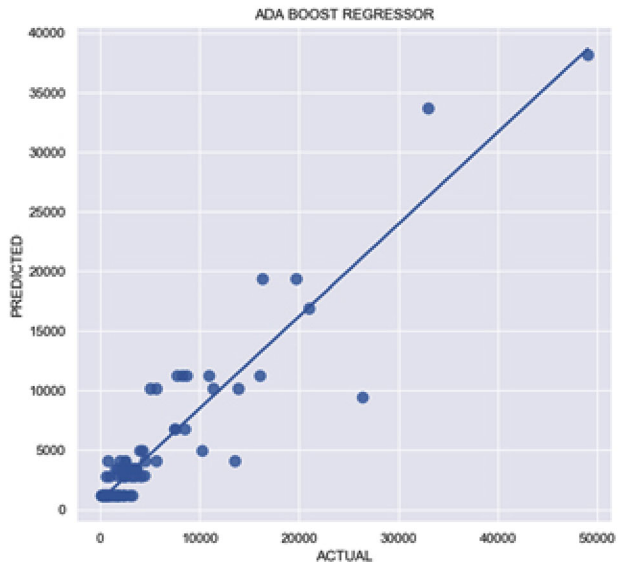

Fig. 7 Actual versus predicted using ABR in China dataset

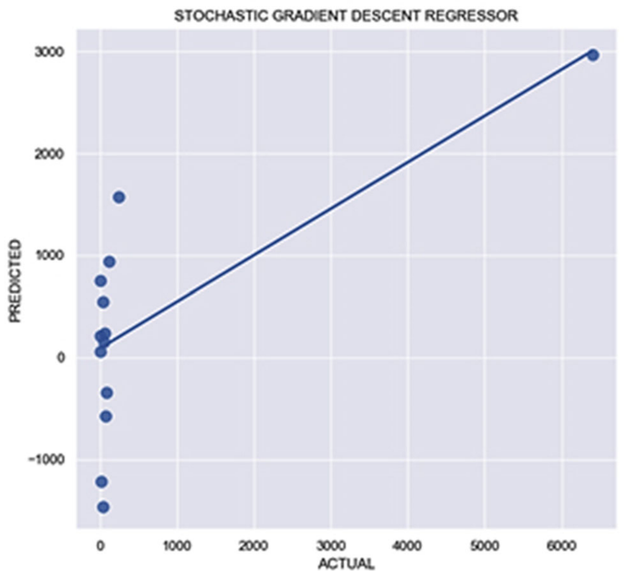

Fig. 9 Actual versus predicted using SGD in COCOMO'81 dataset

scores nearer to 1.0 are in the order of GBR, RFR, BR, DT, ABR, KNN, and SGD containing the values as 0.93, 0.89, 0.88, 0.87, 0.85, 0.61, and 0.60, respectively. So from the results, it is noticed that the gradient boosting regression model outperformed the other models with the CHINA dataset.

From Tables 6 and 7, the results obtained i.e., by comparing various regression models and analyzing the different evaluation metrics, we concluded that our proposed regression model, gradient boosting regressor performed well among the other models with both the datasets. The GBR outperformed the other models by obtaining a score of 0.98 with the COCOMO'81 dataset. While using the CHINA dataset, the GBR model obtained a higher score of 0.93 when compared with the other regression models.

Figures 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15 describes the scatter plots for CHINA and COCOMO'81 datasets. Figures 2, 3, 4, 5, 6, 7 and 8 shows the scatter plots of regression models associated with the CHINA dataset and Figs. 9, 10, 11, 12, 13, 14, and 15 visualize the scatter plots of regression models with the COCOMO'81 dataset. Scatter plots are used to identify the relationship between the variables and are used to examine the patterns. In this case, plots of true and predicted values show a strong, linear, and

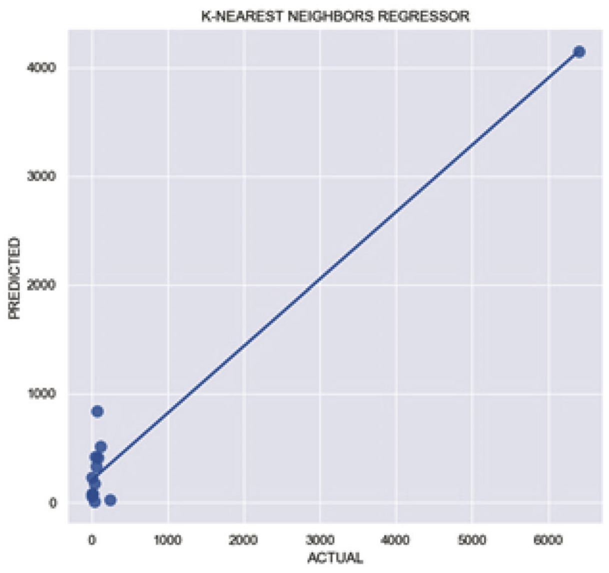

Fig. 10 Actual versus predicted using KNN in COCOMO’81 dataset

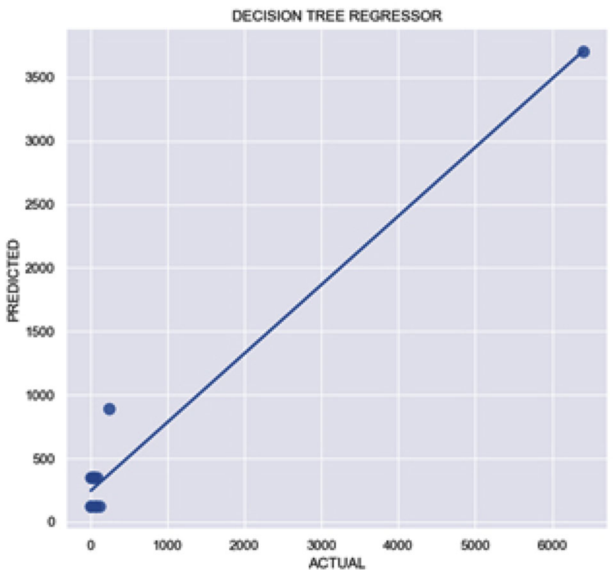

Fig. 12 Actual versus predicted using BR in COCOMO’81 dataset

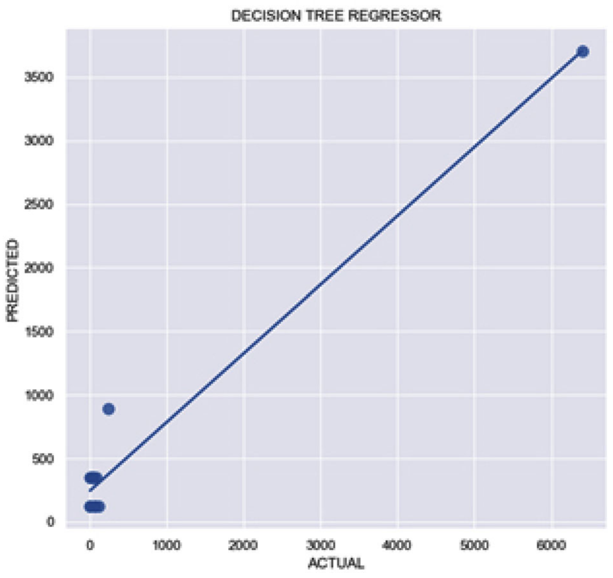

Fig. 11 Actual versus predicted using DT in COCOMO’81 dataset

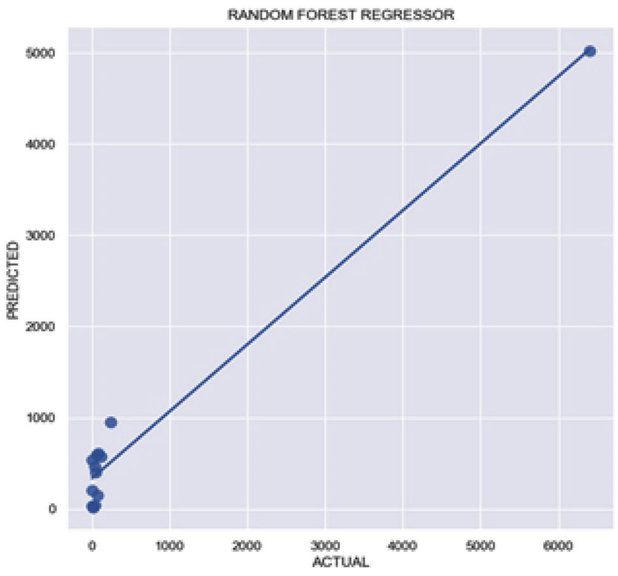

Fig. 13 Actual versus predicted using RF in COCOMO’81 dataset

positive relationship between the variables. The line passing through the origin in all the scatter plots gives a better explanation of how well the model fits. So, if the points are closer to the line and the outliers are lesser, then the model is the best model.

The proposed GBR model gives the best fitline with both the datasets when compared with other models. COCOMO’81 dataset possesses lesser projects compared to the CHINA dataset as we compared only 20% of data for testing. So, the data points plotted for the COCOCMO’81 dataset are fewer in number. Though they are lesser in number, data points are closely fitted to the regression line expressing the best fit.

For better visualization and understanding, we plotted boxplots for each evaluation metrics of various regression models such as MAE, MSE, RMSE, and R^2 using both the datasets. Figures 16, 17, 18 and 19 compares the performance of evaluation metrics of distinct regression models with both the datasets. The boxplots of the evaluation metrics show how the errors and performance of regression models are distributed. The boxplots represent the median value, and the quartiles represent the lower and higher values. We have

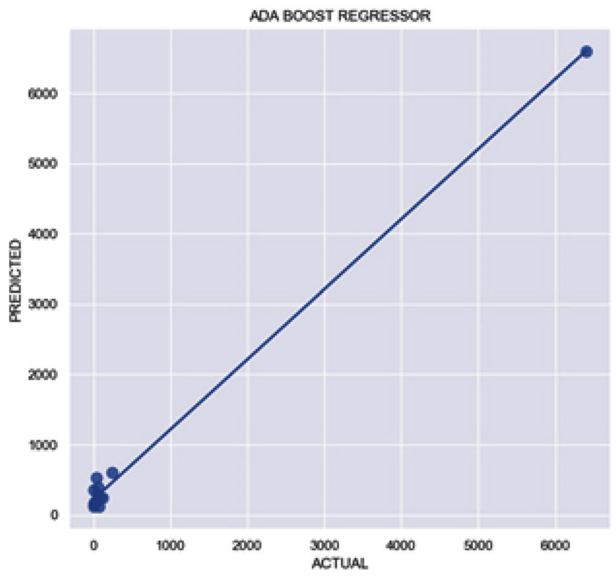

Fig. 14 Actual versus predicted using ABR in COCOMO'81 dataset

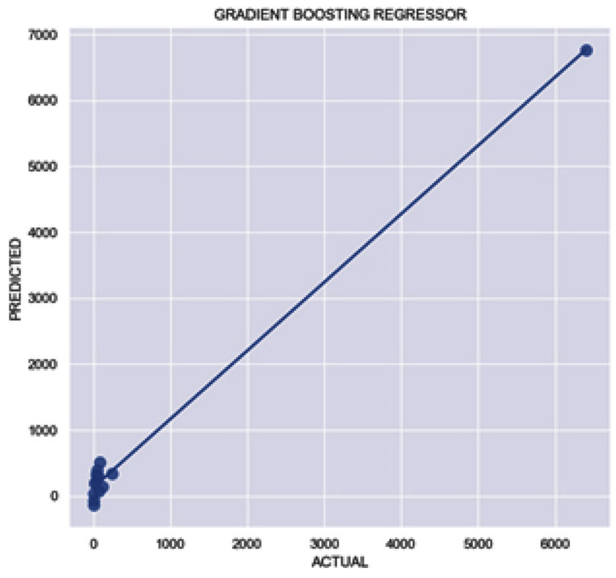

Fig. 15 Actual versus predicted using GBR in COCOMO'81 dataset

considered the evaluation metrics of both the datasets to have a detailed understanding of a better regression model.

The boxplot of MAE in Fig. 16 shows that the box of our proposed model GBR is at the bottom, compared to the other regressor models which explain that the MAE values of the GBR in both the datasets are comparatively lesser than the other models and are identical in the visualization of GBR in Figs. 17 and 18 i.e., box plot of MSE and boxplot of RMSE are also similar as that of MAE boxplot showing that the MSE and RMSE values of GBR model are lesser than the other models. The boxplot of R^2 shown in Fig. 19 provides an important insight into how better is our proposed model

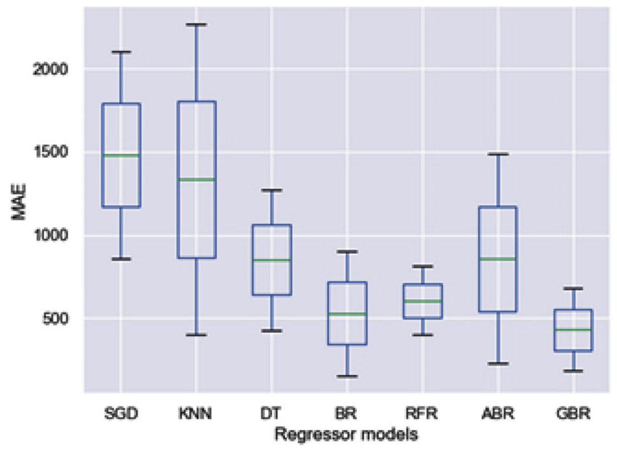

Fig. 16 MAE in COCOMO'81 and China dataset

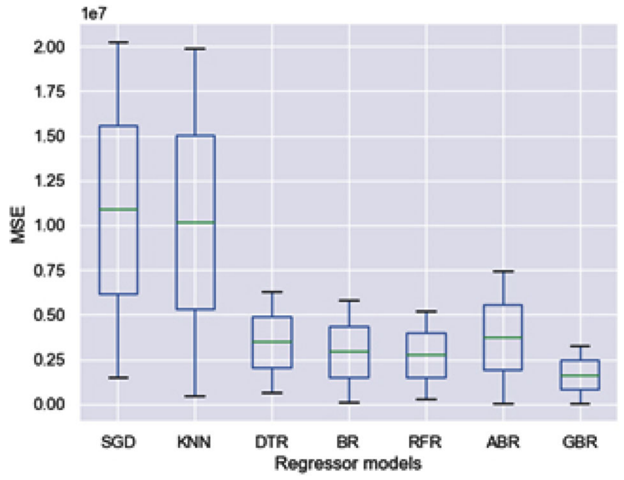

Fig. 17 MSE in COCOMO'81 and China dataset

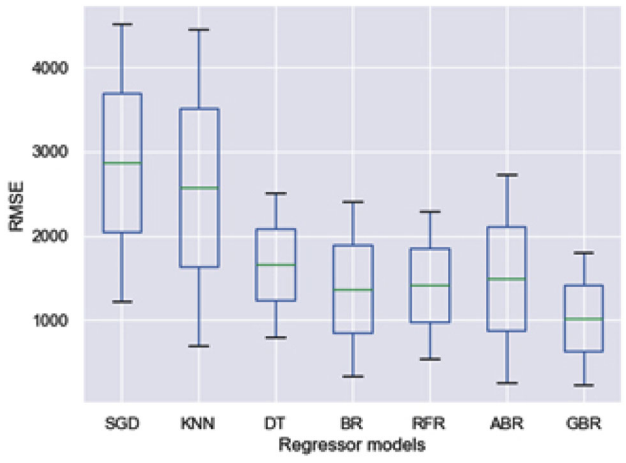

Fig. 18 RMSE in COCOMO'81 and China dataset

performing compared to other models. The score should be close to 1.0 for a better model in R^2 evaluation metrics. Therefore, the values closer to 1.0 are considered as better models.

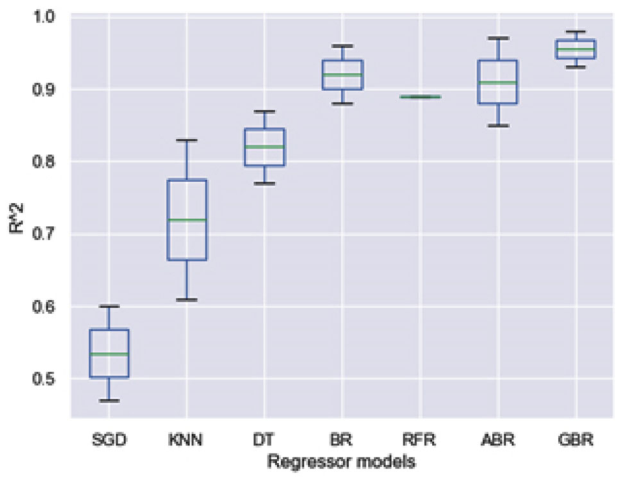

Fig. 19 R^2 in COCOMO'81 and China dataset

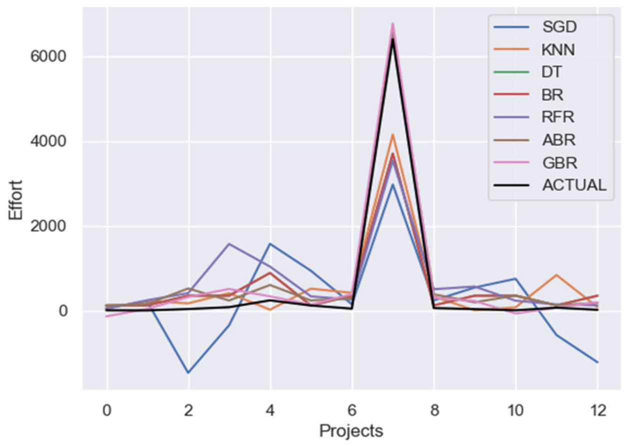

Fig. 20 Actual effort and predicted efforts of regression models in the case of COCOMO'81

So, unlike the other three evaluation metrics, the visualization is different for the R^2 box plot.

Figures 20 and 21 describes the predicted effort of each regression model for the two datasets comparing with the actual effort. These plots show us how each regression model predicted effort is contrasted with the actual effort. We can get a better understanding of each regression model performance for the particular dataset. Figure 20 shows the line plot of regression models effort with the actual effort of the COCOMO'81 dataset, the line of GBR is competent with the actual effort line of the dataset i.e., the predicted effort of this proposed model is relatively closer to the actual effort. Figure 21 shows that the line plot of regression models effort regarding the actual effort of the CHINA dataset, the plot explains that the GBR's effort line is adjacent to the actual effort line of the dataset. Hence, the GBR model's effort is comparatively better than the other models.

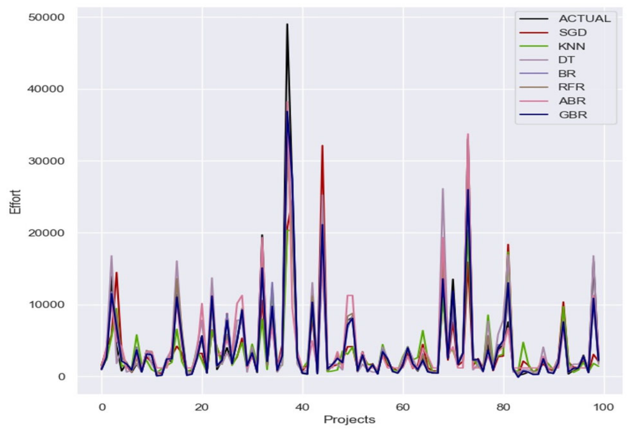

Fig. 21 Actual effort and predicted efforts of regression models in the case of china dataset

All in all, we can conclude that by considering different evaluation metrics, our proposed regression model i.e., gradient boosting regressor performed well compared to the other models and for better comprehension, we examined the visualizations of each regression models for each dataset and our proposed model showed a better fit, and we also considered the visualization of evaluation metrics of both the datasets, which showed a better performance of the proposed model. Finally, the gradient boosting regressor model performs well with the COCOMO'81 and CHINA datasets compared to the SGD, KNN, DT, BR, RFR, and ABR.

Table 8 shows previous works on software effort estimation using various techniques. We showed the results concerning the COCOMO and CHINA datasets. The results proved that when these models were compared with our proposed model, our proposed model outperformed the other previous models in estimating software effort.

6 Conclusion

Software effort estimation is very significant and necessary for software projects. There are many machine learning models and traditional algorithms such as COCOMO, SLIM, functional points are used of software effort estimation. In this paper, a gradient boosting regressor method is proposed for effective software effort estimation. Also, we studied the problem of estimating the effort of software projects by adopting SGD, KNN, DT, BR, RFR, and ABR by employing two datasets COCOMO'81 and CHINA. We evaluated the performance of these models by using MAE, MSE, RMSE, and R^2. For the model to be an accurate model, the MAE, MSE, RMSE should possess low and from the results, it is observed that the gradient boosting algorithm achieved low value compared to the rest of the models. Whereas, for the R^2 metric, the model is considered as an efficient

Table 8 Evaluated values of previous paper models in estimating software effort

S. no.	Author(s)	Year	Intelligent method	Datasets	Evaluation factors	References
1	Azzeh, Elsheikh and Alseid	2014	Optimized analogy-based estimation	CHINA	MMRE—24.7, $PRED_{25}$—80.7, MdMRE—24.6, MMER—16.2, MBRE—23.3.	[37]
				COCOMO	MMRE—50.1, $PRED_{25}$—20.2, MdMRE—26.3, MMER—58.0, MBRE—97.3	
2	Uzun, Erkaymaz and Yapici	2018	ANN	COCOMO	R^2—0.85, MMRE—420	[38]
3	Kumari and Pushkar	2016	Multi-objective genetic algorithm	COCOMO	MMRE—0.52, PRED—0.72	[39]
4	Liu, Xiao and Zhu	2019	Localized neighborhood-based feature selection (LFS)	CHINA	MAR—414, MMRE—0.11	[40]
5	Abdelali, Mustapha and Abdelwahed	2019	Random forest	COCOMO	MMRE—1.40, MdMRE—0.55, PRED(0.25)%—30.43	[12]
6	Azzeh	2011	Model tree-based estimation analogy	CHINA	MMRE—34.9, MdMRE—10.9, PRED—67.1	[41]
				COCOMO	MMRE—21.7, MMRE—21.7, MdMRE—21.9, PRED—60.0	
Proposed method (GBR)				COCOMO	MAE—184.8, MSE—52,314.5, RMSE—228.7, R^2—0.98	
				CHINA	MAE—676.6, MSE—5,252, 196.6, RMSE—1803.3, R^2—0.93	

model if the value is higher and closer to 1.0. The proposed method obtained R^2 values such as 0.98 and 0.93 with the COCOMO'81 and CHINA dataset, respectively. The study proved that the gradient boosting regressor performance is outstanding in terms of COCOMO'81 and CHINA datasets concerning all performance measures. In future work, some other ensemble learning models may be adopted to estimate the effort of software projects and special attention may be attained toward the large-sized projects.

Compliance with ethical standards

Conflict of interest The authors declare that this manuscript has no conflict of interest with any other published source and has not been published previously (partly or in full). No data have been fabricated or manipulated to support our conclusions.

References

1. Al Yahya M, Ahmad R, Lee S (2010) Impact of CMMI based software process maturity on COCOMO II's effort estimation. Int Arab J Inf Technol 7(2):129–137
2. Attarzadeh I, Mehranzadeh A, Barati A (2012) Proposing an enhanced artificial neural network prediction model to improve the accuracy in software effort estimation. In: Proceedings of 2012 4th international conference computer intelligence, communication system networks, CICSyN 2012, pp 167–72
3. Suresh Kumar P, Behera HS (2020) Role of soft computing techniques in software effort estimation: an analytical study. Adv Intell Syst Comput 999:807–831
4. Baskeles B, Turhan B, Bener A (2007) Software effort estimation using machine learning methods. In: 2007 22nd international symposium on computer and information sciences [Internet]. IEEE, pp 1–6. Available from http://ieeexplore.ieee.org/document/4456863/
5. Kocaguneli E, Tosun A, Bener A (2010) AI-based models for software effort estimation. In: Proceedings of 36th EUROMICRO conference software engineering and advanced applications, SEAA, pp 323–326
6. Nassif AB, Capretz LF, Ho D (2012) Software effort estimation in the early stages of the software life cycle using a cascade correlation neural network model. In: Proceedings of 13th ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing. SNPD 2012, pp 589–594
7. Huang J, Li YF, Xie M (2015) An empirical analysis of data preprocessing for machine learning-based software cost estimation. Inf Softw Technol 67:108–127
8. Arslan F (2019) A review of machine learning models for software cost estimation. Rev Comput Eng Res 6(2):64–75
9. Kocaguneli E, Menzies T, Keung JW (2012) On the value of ensemble effort estimation. IEEE Trans Softw Eng 38(6):1403–1416
10. Singal P, Kumari AC, Sharma P (2020) Estimation of software development effort: a differential evolution approach. Proc Comput Sci 167(2019):2643–2652
11. Malgonde O, Chari K (2019) An ensemble-based model for predicting agile software development effort. Empir Softw Eng 24:1017–1055
12. Abdelali Z, Mustapha H, Abdelwahed N (2019) Investigating the use of random forest in software effort estimation. Proc Comput Sci 148:343–352

13. Nassif AB, Azzeh M, Idri A, Abran A (2019) Software development effort estimation using regression fuzzy models. Comput Intell Neurosci. https://doi.org/10.1007/978-94-007-7506-0_7
14. Pospieszny P, Czarnacka-Chrobot B, Kobylinski A (2018) An effective approach for software project effort and duration estimation with machine learning algorithms. J Syst Softw 137:184–196
15. Minku LL, Yao X (2013) Ensembles and locality: insight on improving software effort estimation. Inf Softw Technol 55(8):1512–1528. https://doi.org/10.1016/j.infsof.2012.09.012
16. Friedman H, Greedy J (2001) Function approximation: a gradient boosting machine. Ann Stat. https://doi.org/10.2307/2699986
17. Keprate A, Ratnayake RMC (2017) Using gradient boosting regressor to predict stress intensity factor of a crack propagating in small bore piping. In: IEEE international conference on industrial engineering management, pp 1331–1336
18. Aljahdali S, Sheta AF, Debnath NC (2016) Estimating software effort and function point using regression. In: Support vector machine and artificial neural networks models. Proceedings of IEEE/ACS international conference on computing system applications, AICCSA
19. Reddy PVGDP, Sudha KR, Sree PR, Ramesh SNSVSC (2010) Software effort estimation using radial basis and generalized regression. Neural Netw 2(5):87–92
20. Minku LL, Yao X (2011) A principled evaluation of ensembles of learning machines for software effort estimation. In: ACM international conference on proceeding series
21. Dave VS, Dutta K (2011) Comparison of regression model, feed-forward neural network and radial basis neural network for software development effort estimation. ACM SIGSOFT Softw Eng Notes 36(5):1
22. Sarro F, Petrozziello A, Harman M (2016) Multi-objective software effort estimation. In: Proceedings of international conference on software engineering, pp 619–30
23. Bettenburg N, Nagappan M, Hassan AE (2012) Think locally, act globally: improving defect and effort prediction models. IEEE Int Work Conf Min Softw Repos, pp 60–69
24. Boehm B (1981) Software engineering economics. Available from http://promise.site.uottawa.ca/SERepository/datasets/cocomo81.arff
25. No Title. Available from http://promise.site.uottawa.ca/SERepository/datasets-page.html
26. Boehm BW (1984) Software engineering economics. IEEE Trans Softw Eng 10(1):4–21
27. Bosu MF, Macdonell SG (2019) Experience: quality benchmarking of datasets used in software effort estimation. J Data Inf Qual 11(4):1–38
28. Menzies T, Butcher A, Cok D, Marcus A, Layman L, Shull F et al (2013) Local versus global lessons for defect prediction and effort estimation. IEEE Trans Softw Eng 39(6):822–834
29. Li X, Li W, Xu Y (2018) Human age prediction based on DNA methylation using a gradient boosting regressor. Genes (Basel) 9(9):424
30. Singh AJ, Kumar M (2020) Comparative study on effort estimation using different data mining techniques. Int J Sci Technol Res 9(4):3005–3010
31. Fadhil AA, Alsarraj RG (2020) Exploring the whale optimization algorithm to enhance software project effort estimation. In: 2020 6th international engineering conference "sustainable technology and development" (IEC) [Internet]. IEEE, pp 146–51. Available from https://ieeexplore.ieee.org/document/9122918/
32. Suresh Kumar P, Behera HS (2020) Estimating software effort using neural network: an experimental investigation. In: Advances in intelligent systems and computing. Springer ,Singapore, pp 165–80. https://doi.org/10.1007/978-981-15-2449-3_14
33. Xia T, Krishna R, Chen J, Mathew G, Shen X, Menzies T (2018) Hyperparameter optimization for effort estimation. Available from http://arxiv.org/abs/1805.00336
34. Saljoughinejad R, Khatibi V (2018) A new optimized hybrid model based on COCOMO to increase the accuracy of software cost estimation. J Adv Comput Eng Technol 4(1):27–40
35. Satapathy SM, Rath SK (2017) Empirical assessment of machine learning models for agile software development effort estimation using story points. Innov Syst Softw Eng 13(2–3):191–200. https://doi.org/10.1007/s11334-017-0288-z
36. Satapathy SC, Govardhan A, Srujan Raju K, Mandal JK (2015) Emerging ICT for bridging the future. In: Proceedings of the 49th annual convention of the computer society of India (CSI), vol 1. Advanced intelligent system computing, vol 337, pp 19–30
37. Azzeh M, Elsheikh Y, Alseid M (2014) An optimized analogy-based project effort estimation. Int J Adv Comput Sci Appl 5(4):6–11
38. Uzun R, Erkaymaz O, Yapici İŞ (2018) Comparison of artificial neural network and regression models to diagnose of knee disorder in different postures using surface. Electromyography 31(1):100–110
39. Kumari S, Pushkar S (2016) A framework for analogy-based software cost estimation using multi-objective genetic algorithm. Lect Notes Eng Comput Sci 2225:508–515
40. Liu Q, Xiao J, Zhu H (2019) Feature selection for software effort estimation with localized neighborhood mutual information. Cluster Comput 22(1):6953–6961
41. Azzeh M (2011) Model tree based adaption strategy for software effort estimation by analogy. In: Proceedings of 11th IEEE international conference on computing information technology, pp 328–335