Bayesian Hyperparameter Optimization and Ensemble Learning for Machine Learning Models on Software Effort Estimation

Robert Marco[1]*, Sakinah Sharifah Syed Ahmad[2], Sabrina Ahmad[3]
Department of Informatics, Universitas Amikom Yogyakarta, Yogyakarta, Indonesia[1]
Faculty of Information & Communication Technology, Universiti Teknikal Malaysia Melaka, Malaysia, Melaka, Malaysia[2, 3]

***Abstract*—In recent decades, various software effort estimation (SEE) algorithms have been suggested. Unfortunately, generating high-precision accuracy is still a major challenge in the context of SEE. The use of traditional techniques and parametric approaches is largely inaccurate because they produce biased and subjective accuracy. Meanwhile, none of the machine learning methods performed well. This study applies the AdaBoost ensemble learning method and random forest (RF), on the other hand the Bayesian optimization method is applied to determine the hyperparameters of this model. The PROMISE repository and the ISBSG dataset were used to build the SEE model. The developed model was comprehensively compared with four machine learning methods (classification and regression tree, k-nearest neighbor, multilayer perceptron, and support vector regression) under 3-fold cross validation (CV). It can be seen that the RF method based on AdaBoost ensemble learning and bayesian optimization outperforms this approach. In addition, the AdaBoost-based model assigns a feature importance rating, which makes it a promising tool in software effort prediction.**

***Keywords*—Bayesian optimization; adaboost ensemble learning; random forest; software effort estimation**

I. Introduction

Software effort estimation (SEE) is a method of estimating the amount of time it will take to build a software system in person-months or person-hours [1][2]. Uncertainty and imprecision characterize software effort estimation environments [3][4]. The topic of SEE, on the other hand, has been characterized as a regression problem in general [5]. Various SEE models in the literature still show considerable performance deviations and are extremely dataset-dependent.

SEE has previously been accomplished via expert judgment, analogy, decomposition and recomposition, and parametric techniques [6]. Meanwhile, standard SEE methodologies can produce erroneous estimates due to human bias and subjectivity [7]. The Machine learning (ML) algorithm is very effective in modeling uncertainty as a better decision-making process [8]. However, no single learning method is ideal for all supervised learning tasks [9]. However, a single sophisticated algorithm may not be a consideration for building current SEE models, as the performance of any model mainly depends on the characteristics of the data set used, such as data set size, outliers, categorical features, and missing values.

Several machine learning techniques have been widely applied in the SEE context which have been considered as necessary steps, such as: Case-Based Reasoning (CBR), Artificial Neural Networks (ANN), Support Vector Regression (SVR) [10], while for other ML methods, such as Random Forest (RF), Classification And Regression Tree (CART) and K-Nearest Neighbor (kNN), they are still ignored. RF is a powerful, easy-to-train ensemble learner with big data [11]. RF is widely used in the data mining domain and has achieved good performance when dealing with regression and classification problems [12]; this method can overcome overfitting and is less affected by outliers [13]. On the other hand, RF improves prediction accuracy without significant computational improvement, and is not sensitive to multicollinearity [14]. Some researchers have also applied RF to solve regression problems in the context of SEE, for example [15][16][17]. Unfortunately, the RF model can only be extended horizontally because decision trees exist in parallel and these decision trees have equal weight in voting for the final prediction even though some of these trees may underperform.

The use of Ensemble learning combines several algorithms that process different hypotheses to make their predictions perform well [18]. According to Lessmann et al. (2015), the ensemble method is better than the single ML and other statistical method [19]. While, Kocaguneli et al. (2012), the proper use of the ensemble method can outperform the performance of single learners on the ML model [20], as well as being one of the best methods in increasing the accuracy and stability of the most influential estimation [21]. Averaging, voting, and bagging are three common broad ensemble approaches that have piqued the interest of machine learning researchers. Meanwhile, developing ensemble methods such as stacked generalization, AdaBoost algorithm, Gradient tree boosting have not been widely tried/ignored [22].

Ren et al. (2016) investigated the use of ensembles in classification and regression and the success of AdaBoost about regression behaviour [23]. AdaBoost, as a popular boosting algorithm, combines weak estimators and implements them on an improved version of the data with the help of weighted majority voting/hard voting [24]. However, AdaBoost may not offer high accuracy when the dataset is heavily contaminated with noise [25]. In contrast to, Martin-Diaz et al. (2017), AdaBoost is also known to achieve a

*Corresponding Author.

significant reduction in bias as well [26], and low error variance [24]. Also, it is not easy to overfit during training [27].

Unfortunately, the method will need to be fine-tuned to fit the scenario at hand. Automatic hyperparameter adjustment saves time and effort when experimenting with different machine learning model configurations, improves algorithm accuracy, and increases reproducibility. Hyperparameter tuning is a well-known approach for achieving optimal performance in machine learning models [28][29]. Several other studies have shown that the accuracy dimension in SEE is strongly influenced by choice of information estimation using parameter tuning techniques [30]. Because determining the best hyperparameter combination can improve the ML model's performance [28][31]. However, much of the work seems to implicitly assume that tuning the parameters will not significantly change the results [32]. In the worst case, improper parameter setting may lead to inferior performance results [33]. However, the default hyperparameter setting may not produce consistent results depending on the data set used [34]. Based on the work by [35], there is still limited work investigating the impact of parameter setting for SEE methods in improving prediction accuracy.

Manual search, grid search, and random search are some of the most used hyperparameter optimization strategies [36]. When the hyperparameter space is large, however, this method is time consuming and impractical [6]. Manual search necessitates a higher level of professional knowledge, is difficult to implement without prior experience, and takes time [37]. Meanwhile, grid search suffers from a dimensional curse, which means that as the number of hyperparameters set or the complexity of the search space and the range of values of the hyperparameters increases, the algorithm's efficiency declines dramatically [37]. Random search, on the other hand, is more effective in high-dimensional space [38], but this method is not reliable for training complex models [38]. Despite the fact that this method provides automatic tuning, it can acquire the optimization goal function's ideal global importance.

Among other classic hyperparameter optimization techniques, Bayesian optimization is a very successful optimization algorithm for solving computationally expensive functions [39]. Bayesian hyperparameter optimization technique to further promote generalization accuracy [40]. Bayesian optimization is effective for problems with fewer hyperparameters and that are difficult to parallelize [6]. Therefore, it promises to encourage the use of hyperparameter tuning for further applications in SEE.

Motivated by the benefits mentioned above, the AdaBoost RF-based method of ensemble learning was developed to capture the relationships between features in an SEE context. To reduce time dependence and impracticality, Bayesian optimization method is used to find suitable hyperparameter models. The datasets in the PROMISE and ISBSG repositories have built the model in this paper. Based on literature review, there is still a limited use of the AdaBoost and RF ensemble learning methods adopted to build the SEE model. The remainder of this paper begins with related work, followed by experimental design. Then, the results and discussion containing the comparison of models. At the end, discuss the conclusions and future work.

II. Related Work

Meanwhile, the literature on offline learning does not have a supervised procedure for automatic parameter setting. In the context of offline SEE, the use of Classification and Regression Tree (CART) with the addition of more innovative features can improve accuracy [28], the researchers used a grid search strategy to obtain optimal parameters for five machine learning techniques (KNN, Regression Trees (RT), Multilayer Perceptrons (MLP), Bagging+RT, and Bagging+MLP) without used generating ensembles. The results revealed that while RT, Bagging+RT, and KNN were unaffected by tuning settings, MLP and Bagging+MLP were. Minku (2019) Linear Regression in Logarithmic Scale (LogLR) results in a more stable prediction performance [41]. Meanwhile, Minku and Yao (2013) investigated the RT, Bagging+MLP, and Bagging+RT techniques shown to perform well across several data sets. This suggests that the best parameters to use with a machine learning approach may change over time [35]. In the context of SEE, parameter tuning in Support Vector Regression (SVR) is critical. A tabu search has been proposed in particular to find the best SVR parameter tuning [42]. Elish (2013) conducted an empirical study based on five machine learning approaches (KNN, SVR, MLP, Decision Tree (DT), and Radial Basis Function Network (RBFN)) that suggested a heterogeneous ensemble. Due to its irregular and unstable performance across the specified data set, the single approach is unreliable. Furthermore, across all data sets, five machine learning approaches were trained using the same configuration, but no explanation for parameter tuning was supplied [43].

Meanwhile, Villalobos-Arias and Quesada-López (2021) investigated CART, SVR, and ridge regression (RR) using random search and grid search with six bio-inspired algorithms. The results demonstrate that the Flash+Log+SVR model is the most accurate in the most data sets, while the Hyperband+Log+RR model is the most stable in the most datasets [44]. In particular, the stacked ensemble that offers the best overall accuracy in this study takes the average of the estimated effort values generated by Bagging, RF, ABE, AdaBoost, Gradient boosting machine, and Ordinary least squares regression which are optimized using grid search techniques [25]. Meanwhile, Zakrani et al. (2018) used the grid search (GS) method to improve SVR. The results show that this approach can help improve the SVR technique's performance [45]. Stacking ensemble learning uses two hyperparameter tuning (Particle Swarm Optimization and Genetic Algorithms) while base learners (linear regression, MLP, RF, and Adaboost regressor). Experimental results reveal that the estimation accuracy is higher when hyperparameters are set using PSO [6]. ROME (Rapid Optimizing Methods for Estimation) is a configuration technique that uses sequential model-based optimization (SMO) to identify configuration settings for KNN, SVR, RF, and CART techniques. For both traditional and current tasks, ROME outperforms sophisticated approaches [46]. The accuracy and stability of SVR in SEE were tested to see how a

random search hyperparameter tuning strategy affected them. According to the findings, the SVR set for random search performed similarly to the SVR set for grid search [47].

Based on the findings of previous empirical investigations in the context of SEE. This paper is different from previous research. The RF-based AdaBoost ensemble learning method reinforced by the Bayesian optimization method was used to find the appropriate hyperparameter model. Meanwhile, four ML techniques, such as: classification and regression tree (CART), k-nearest neighbor (k-NN), multilayer perceptron (MLP), and support vector regression (SVR) were used to compare the performance results of the proposed SEE model.

III. Experimental Design

The data sets, ensemble learning, hyperparameter tuning, parameter setting ML, and evaluation measures utilized in this paper are described in depth in this section.

A. Datasets

The most widely used datasets related to the SEE context are the Repositories on PROMISE and ISBSG, among the most popular datasets. In 9 datasets from the public PROMISE repository (also known as SEACRAFT), as well as two sub-datasets from the ISBSG10 and ISBSG18-IFPUG repositories.

Table I lists the data set that was used in paper, including the number of projects, features, and categorical features. This paper uses eleven datasets (china, albrecht, maxwell, nasa93, cocomo81, kitchenham, kemerer, desharnais, and ISBSG10) and the preprocessing rules used in the study by [28][1], and the UCP dataset according to the regulations [48]. Meanwhile, ISBSG18-IFPUG refers to research [44].

B. Data Preprocessing

The Data Preprocessing approach in study was used to improve prediction accuracy in the end. The Data Preprocessing technique is an effective option for effort estimation [50], is a crucial step in improving machine learning performance [51]. According to Famili et al. (1997) the first step by removing features on the dataset that is not relevant. Machine learning algorithms will perform better if irrelevant features are removed [52]. Subsequent processing converts information on categorical data into numeric. Ordinal coding assigns a unique number code to each category [53], the advantage is that the dimensions of the problem space do not increase because each category is displayed as a single input [54]. Handling missing data by using kNNI (K-Nearest Neighbor Imputation). This method proved to be efficient for estimating missing attribute values in various software engineering datasets [55]. In this study, data normalization was used as a scale of values 0 and 1. For all datasets, Mensah et al. (2018) discovered that the normalized Z-score [0,1] generated the best prediction accuracy when compared to box-cox and log transformation [56]. This research will utilize two subsets at random: training (80%) and testing (20%) as a predicted evaluation of the training procedure.

C. Random Forest Algorithm

For regression purposes, the random forest is a set of tree predictors $h(x;\theta_k), k = 1,2,\dots,K$ where x represents the observed input vector (covariate) of length p with a random vector associated with X and θ_k is independent and identically distributed (iid) random vector. A regression setting where has a numeric result, Y, but makes multiple points of contact with the classification problem (categorical results). The observed (training) data are assumed to be taken independently of the combined distribution (X,Y) and consist of $n(p+1) - tuples\ (x_1,y_1),\dots..(x_n,y_n)$.

For regression, the random forest prediction is the unweighted mean of the collection $\bar{h}(x) = (\frac{1}{K})\sum_{k=1}^{K} h(x;\theta_k)$. For $k \to \infty$ the Law of Large Numbers ensures [57].

$$E_{X,Y}(Y - \bar{h}(X))^2 \to E_{X,Y}(Y - E_{\theta}h(X;\theta))^2 \quad (1)$$

The quantity on the right is the prediction (or generalization) error for a random forest, designated PE_f^*. Convergence in Eq. (1) implies that the random forest is not overfit [57]. Determine the mean prediction error for the individual tree $h(X;\theta)$ as:

$$PE_t^* = E_{\theta}E_{X,Y}(Y - h(X;\theta))^2 \quad (2)$$

Assume that for all θ the tree is unbiased, eg $EY = E_X h(X;\theta)$, then yields:

$$PE_f^* \leq \bar{\rho}PE_t^* \quad (3)$$

Where $\bar{\rho}$ is the weighted correlation between residuals $Y - h(X;\theta)$ and $Y - h(X;\theta')$ for independent θ, θ'.

D. Adaboost Ensemble Learning

AdaBoost is a popular variation of the original Boosting scheme [27]. AdaBoost is a robust ensemble approach for fitting a poor collection of learners to a enhanced data set. The predictions of the weak learner are merged using weighted summation, to reproduce the final prediction [58]. The Adaboost regressor is a high-accuracy ensemble learner that is used to tackle regression issues [59], Adaboost.R2, a modified version of Adaboost.R, is used for regression [27].

TABLE I. The Study's Data Set

Dataset	***Size*** [49]	***Proj***	***N***	***Cat.***
China	large	499	17	0
Albrecht	small	24	7	0
Cococmo81	small	63	17	0
Desharnais	medium	81	8	0
IFPUG	large	36	12	11
ISBSG10	large	952	11	6
Kemerer	small	16	6	0
Kitchenham	large	145	4	0
Maxwell	small	62	26	0
Nasa93	small	93	18	16
UCP	small	71	5	0

Given $w_1, w_2, \dots, w_N$, which is applied to the training set, seeks to improve the training data in each boosting iteration, using different weights. The first update iteration uses the same weights and training data as before. The learner's algorithm is then re-applied to the new weighted data, after each initial weight is updated. If the weights for the training data that were predicted incorrectly in the previous phase are increased, the weights for the training data that were successfully predicted will be reduced. Finally, each weak learner is compelled to concentrate on the sample that the preceding one missed in the sequence [27]. In the following, Adaboost.R2 steps according to the rules [60][61]:

- Set the initial weight (w_i) to the training set.
- As the training set, define the original data's input (x) and target (y) variables.
- Install the regression model (h_t) to the training set with the notation $h_t: x \rightarrow y$.
- Get the predicted training target value: $y_i^{(p)}(xi)$.
- Use the following equation to find the loss value for the i-th training sample ($\mathcal{L}_i$).

$$\mathcal{L}_i = \delta\left[\left|y_i^{(p)}(xi) - y_i\right|\right] \quad (4)$$

Any function that is $\mathcal{L}_i \epsilon [0,1]$, can be used as the loss function (δ). Calculate the average loss ($\bar{\mathcal{L}}$) for v_i using the following formula:

$$\bar{\mathcal{L}} = \sum_{i=1}^{N} \mathcal{L}_i \frac{w_i}{\sum w_i} \quad (5)$$

When $\bar{\mathcal{L}}$ is smaller than 0.5, an appropriate forecast is produced.

- If $\bar{\mathcal{L}}$ is more than 0.5, the weights should be updated using the equation below.

$$w_i \rightarrow w_i \beta^{[1-\mathcal{L}_i]} \quad (6)$$

$$\beta = \frac{\bar{\mathcal{L}}}{1-\bar{\mathcal{L}}} \quad (7)$$

- To get the required loss function range, repeat steps 4-7.

E. Bayesian with Gaussian Process Optimization of Model Hyperparameters

Bayesian optimization is a useful strategy for locating the by extremes of computationally expensive functions [39]. In this paper, Bayesian optimization is used in this research to discover the maximum value at the sampling point for the unknown function f in model hyperparameter configuration problem [62][63]. The objective function is computationally $f: X \rightarrow \mathbb{R}^+$ over the compact hyperparameter domain X^1, which aims to minimize *f* without using gradient information.

Thus, the hyperparameter mapping in Bayesian depends on the objective function. The target value is predicted with the historical result X. A series of steps to find the hyperparameter as follows: 1) Define the historical model, 2) Find the optimal parameter, 3) Apply the detected hyperparameter to the objective function, 4) Update the model with new result, and 5) 2-4 steps are repeated until the threshold value is reached or the process exceeds the limited time.

Determines a previously small sample of n points $x_i \in X$ uniform at random, and computes the value of the function at those locations $f(x_1), \dots, f(x_n)$. Then, model f using a probabilistic model for the function. We'll take *f* as a Gaussian process. Because the Gaussian process is so flexible and simple to use, Bayesian optimization uses it to fit the data and update the posterior distribution [37]. For only a finite collection of points $x_1, \dots x_n$, posterior delivers a probability distribution over a particular function. The Gaussian process posits that the probabilities $p(f(x_1), \dots, f(x_n))$ form a multivariate Gaussian distribution that is specified by the mean function $\mu(x)$ and the covariance function $\kappa(x_i, x_j)$, where κ is a positive definite kernel function (such as: the squared exponential kernel, the rational quadratic kernel, and the Matern kernel). The posterior predictive mean function $\mu(x)$ and the posterior predictive marginal variance function $\sigma^2(x)$, both specified across the X, domain and calculated in closed form, are obtained by modeling f using the Gaussian process [62].

Then determine the sampling point, x_{n+1}, from X to find the location of the minimum function. Controlled by the optimization proxy of the acquisition function, $u: X \rightarrow \mathbb{R}^+$.

$$x_{n+1} = arg\max_x u(x) \quad (8)$$

This paper, will use the expected enhancement algorithm which is defined as follows [64]:

$$u_{EI}(x) = E[max(0, f(x^*) - f(x))] \quad (9)$$

The minimum observation value of f is $f(x^*)$, while $E[\cdot]$ which expresses the expectation of a random variable at $f(x)$. Thus, we receive the same reward as "fixing" $f(x^*) - f(x)$ there is no alternative reward when $f(x)$ is less than $f(x^*)$. The right-hand side of Eq. (9) can be written as given the Gaussian Process predicted mean and variance functions:

$$u_{EI}(x) = \big(f(x^*) - \mu(x)\big)\Phi(\mathrm{Z}) - \phi(\mathrm{Z}) \quad (10)$$

Where, ϕ is its derivative, and Φ is the standard normal cumulative distribution function, while $Z = \frac{f(x^*)-\mu(x)}{\sigma(x)}$.

Based on the above analysis, the basic framework of Random Forest and AdaBoost embedded in Bayesian Optimization is formulated in Fig. 1.

F. Parameter Settings ML

Classification And Regression Tree (CART), Multilayer Perceptron (MLP), Support Vector Regression (SVR), K-Nearest Neighbor (KNN), and Random Forest (RF) were employed in the experiment. Table II shows the hyperparameter search space for setting the parameter values of a single approach using Bayesian Optimization.

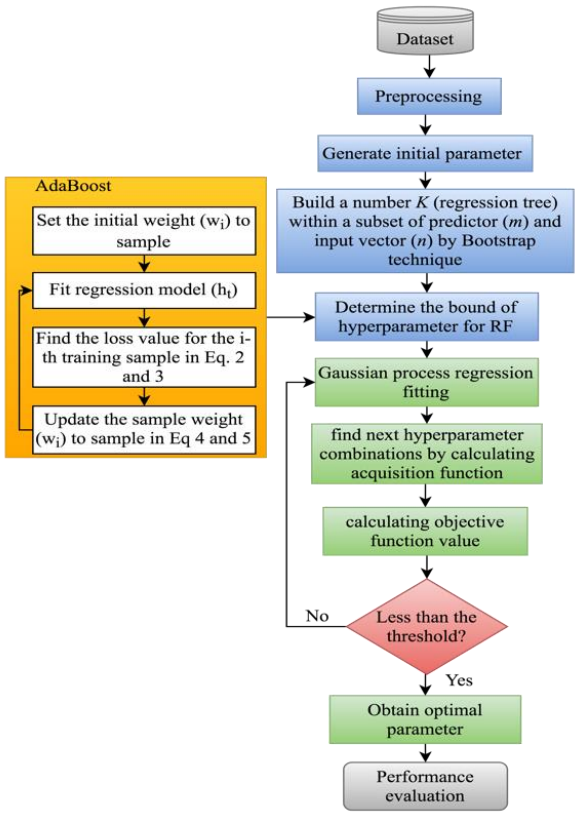


Fig. 1. The Flow Chart of the Proposed RF-Adaboost with Bayesian Optimization.

TABLE II. ML TECHNIQUE PARAMETER VALUES

ML	Parameter Values
CART	criterion: {mse, mae} max_depth: {2, 6, 7, 8} min_samples_split: {10, 20, 40} max_leaf_nodes: {5, 20, 100} min_samples_leaf: {20, 40, 100} max_features:{auto, log2, sqrt, None}
MLP	hidden_layer_sizes: {(50,50), (100,50)} solver: {adam, sgd} activation: {'relu', 'tanh'} learning_rate: {constant, adaptive} alpha: {0.0001, 0.05}
SVR	kernel: {sigmoid, poly, rbf} degree: {3, 4, 5, 6, 7, 8, 9} kernel parameter: {0.0001, 0.001, 0.01, 0.1} learning rate: {0.01, 0.02, 0.03, 0.04, 0.05} deviation tolerated: {0.001, 0.01, 0.1}
k-NN	K: {1, 2, 3, 4, 5, 6} weights: {uniform, distance}
RF	number of trees: 125 criterion: {mse, mae} n_estimators: {10, 20, 30, 50, 100} min sample leaf: {3, 4, 5, 6, 7} min samples split: {3, 5, 7, 9} max_features: {1,13} max_depth: {5, 15, 20, 30, 50} max depth of the tree: {100, 200, 300}

G. Cross-Validation

The cross-validation methodology is a widely known method for revealing the model's true performance, and it is highly recommended by researchers [58]. For the PROMISE and ISBSG R10/R18-IFPUG datasets, will apply ten times 3-fold cross-validation. This paper divided the data into three folds or groups at random. The test set is chosen, and the remaining two folds are joined to form the train set. For each schema, the model is based on a train set (a combination of machine learning algorithms, ensemble learning, and hyperparameter tuners). Within this framework, AdaBoost functions as a meta-regressor for ensemble and Bayesian optimization to provide automatic tuning that functions as an appropriate hyperparameter model optimization objective. Sub-partitioning is done via 3-fold cross-validation because the tuner does not have access to the test set. The model is retrained on the entire set with these parameters once the optimal hyperparameter values have been identified. Finally, an assessment matrix will be used to assess the model. The flowchart of the framework in the proposed regression scheme based on AdaBoost and bayesian hyperparameter tuning is summarized in Fig. 2.

H. Evaluation Metrics

Mean absolute error (MAE), root mean square error (RMSE), and R-squared (R^2) are the metrics that are used in the evaluation. If the MAE and RMSE are low, and the R^2 is high, the model is better.

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|X_i - Y_i| \quad (11)$$

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(X_i - Y_i)^2} \quad (12)$$

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(X_i - Y_i)^2}{\sum_{i=1}^{m}(\bar{Y} - Y_i)^2} \quad (13)$$

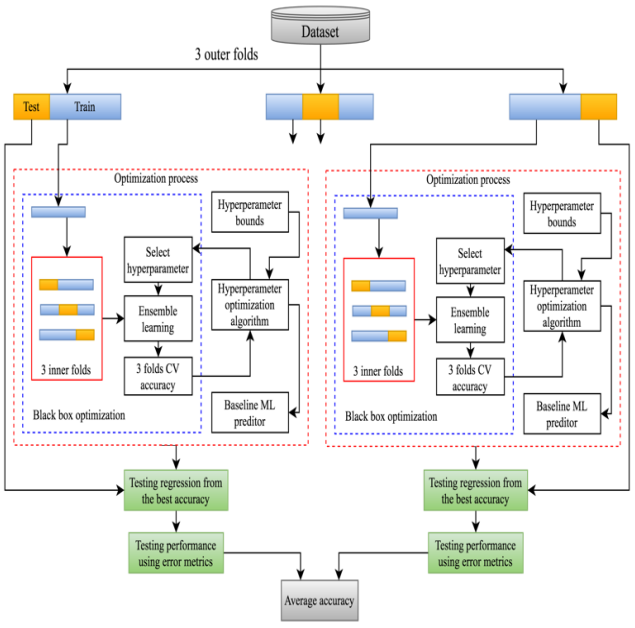


Fig. 2. Procedure Training and Testing Scheme.

IV. Result and Discussion

This section will address the issues raised in section 1 by conducting three sets of experiments to determine the accuracy of the SEE: 1) without hyperparameters tuning (default), 2) hyperparameters tuning, and 3) SEE model using Adaboost ensemble learning with Bayesian hyperparameters.

In this paper describe the experiments in depth and offer the findings of each experiment in this section. All of the tests were run in a Python environment. In this study, eleven software engineering repository data sets with various dimensions and attributes were employed. Table I lists the dataset's details, including the number of samples, characteristics, and target value. In the first step, carried out the data preprocessing stage, which was used to overcome missing data by Missing Data Treatment (MDT) using k-NNI and converting categorical data into numeric using the ordinal encoder. In the next step, will normalize the data with a scale of 0 and 1. The data has been converted to a format where powerful machine learning algorithms may be deployed to create accurate predictions in the SEE context using various data preprocessing approaches.

A. Model with Default Parameter Tuning

After passing through the data preprocessing stage, the next step will be to compare 5 ML methods (namely, CART, MLP, SVR, KNN, and RF) without setting the hyperparameters using the default parameters tuning. The purpose of the ML algorithm comparisons is to assess which algorithm is more likely to have the best performance without tuning in to different problems. The default settings in the training data are used to train the machine learning technique. More exact findings are obtained by using the ML approach with the lowest MAE and RMSE. When it comes to the R^2 value, the greater the number, the better the accuracy. For the performance of the ML method, which considers the values of MAE, RMSE, and R^2. Where the best value is marked in bold, otherwise the worst value is marked in italics.

Tables III to V list the best possible performance values for each model and dataset, as well as the tuner who achieved them (without tuning where the parameters are set randomly within the range). Each model's performance changes depending on the dataset. For the PROMISE and ISBSG datasets, the dataset used with small/medium/large effect sizes [49]; in the first experimental stage, the default settings for the base learners model will yield the most accurate predictions. Based on the experimental results in this paper, it can be seen that RF almost usually outperforms other methods. In particular, when measured in MAE (Table III), RF achieved the best average rating, CART performed second, followed by k-NN and SVR with a slightly worse average rating, and MLP performed the worst among all related methods. Nonetheless, no significant differences could be found concerning the three methods: k-NN, MLP, and SVR. RF has advantages over other methods in most datasets with medium/large effect sizes but performs worse than CART, k-NN, and SVR in many datasets with small effect sizes. In terms of RMSE (Table IV) and R^2 (Table V), the results are almost similar to the MAE values.

TABLE III. Comparison MAE using Default Parameter Tuning

Dataset	MAE				
	CART	KNN	MLP	SVR	RF
Albrecht	**0.0534**	0.1016	0.0888	*0.1694*	0.1217
China	0.0178	0.0244	0.0284	*0.0567*	**0.0115**
Cocomo81	0.0934	**0.0466**	*0.0949*	0.1230	0.0681
Desharnais	0.0228	0.0594	*0.0678*	0.0522	**0.0161**
IFPUG	0.2557	0.1591	0.1853	*0.1923*	**0.1698**
ISBSG10	0.0174	0.0268	0.0228	*0.0568*	**0.0132**
Kemerer	**0.1167**	0.2732	0.2442	*0.3319*	0.2326
Kitchenham	0.0065	0.0068	0.0074	*0.0823*	**0.0044**
Maxwell	0.0667	0.0718	0.0638	*0.0982*	**0.0544**
Nasa93	0.0433	0.0492	0.0451	*0.0945*	**0.0213**
UCP	0.1146	0.1367	*0.2307*	**0.0960**	0.1012

TABLE IV. Comparison RMSE using Default Parameter Tuning

Dataset	RMSE				
	CART	KNN	MLP	SVR	RF
Albrecht	**0.0776**	0.1538	0.1238	*0.2379*	0.1702
China	0.0479	0.0479	0.0567	*0.0730*	**0.0400**
Cocomo81	*0.1911*	**0.0745**	0.1139	0.1528	0.1396
Desharnais	0.0321	0.0656	*0.0861*	0.0654	**0.0248**
IFPUG	*0.3762*	0.2290	0.2379	0.2382	**0.2271**
ISBSG10	0.0387	0.0455	0.0309	*0.0624*	**0.0280**
Kemerer	**0.1964**	0.2934	0.2680	*0.3673*	0.2674
Kitchenham	0.0129	0.0156	0.0150	*0.0831*	**0.0098**
Maxwell	0.1049	0.0982	**0.0855**	*0.1120*	0.0959
Nasa93	0.1102	*0.1223*	0.0717	0.1179	**0.0457**
UCP	0.2655	0.1801	*0.2932*	**0.1206**	0.1961

TABLE V. Comparison R^2 using Default Parameter Tuning

Dataset	R^2				
	CART	KNN	MLP	SVR	RF
Albrecht	**0.9524**	0.8135	0.8791	*0.5536*	0.7714
China	0.7954	0.7956	0.7131	*0.5246*	**0.8570**
Cocomo81	*-0.6659*	**0.7467**	0.4075	-0.0650	0.1108
Desharnais	0.9284	0.7014	*0.4870*	0.7032	**0.9572**
IFPUG	*-2.7883*	-0.4036	-0.5145	-0.5186	**-0.3807**
ISBSG10	0.6840	0.5636	0.7984	*0.1795*	**0.8344**
Kemerer	**0.8173**	0.5926	0.6599	*0.3614*	0.6616
Kitchenham	0.7354	0.6119	0.6429	*-9.8874*	**0.8474**
Maxwell	0.5269	0.5848	**0.6853**	*0.4590*	0.6043
Nasa93	0.2071	*0.0227*	0.6643	0.0928	**0.8635**
UCP	0.3484	0.7003	0.2054	**0.8656**	0.6445

Based on the study results in the table, it shows that the algorithm with the best performance in almost all datasets is RF. RF obtained the highest accuracy in china, desharnais, IFPUG, ISBSG10, kitchenham, Maxwell, and Nasa93. With the best accuracy value in the kitchenham dataset with MAE (0.0044), RMSE (0.0098), and R^2 (0.8474), although Desharnais owns the best R^2 value. Furthermore, the second position is obtained by CART, which has the best accuracy value on albrecht and kemerer. Meanwhile, KNN, MLP, and SVR have almost similar performance.

The lack of parameter adjustment in this situation can result in worst performance of CART, KNN, MLP, and SVR. This is due to SVR's ability to perform effectively with limited data sets. However, it is unsuccessful in dealing with outliers in training data, which is a common occurrence in real-world applications. As a result, some outliers cause regression to be poor. Meanwhile, MLP, with a small data set without any appropriate parameter tuning, can reduce the number of hidden nodes which causes a decrease in its approximation ability [65]. KNN is extremely sensitive to characteristics that are irrelevant or redundant. Because it is unclear which sort of distance and which attribute are employed in distance-based learning KNN to give the best results, and because must calculate the distance of each query instance to all training samples, the computational cost is relatively large [66]. CART performs badly on smaller data sets compared to bigger data sets. As a result, using this CART approach without considering the data's magnitude is not recommended [67].

B. Comparison Model with Hyperparameter Tuning and Ensemble Learning

Next, this model will use hyperparameter tuning based on the Bayesian-based Gaussian process. The ML method is trained with hyperparameter tuning on the training data in the training process. All tuners are used as an optimization method combined with a cross-validation procedure. Configure using cross-validation (i.e. cv: 3), verbose: 3, scoring: 'mean_squared_error', and 10 iterations. Because the datasets in this scenario are small, will narrow the search space to the most promising values based on previous research.

Next, the same experiment was repeated using Adaboost ensemble learning. With repeat the experiment to find the best convergence with 10 iterations. Configure the Adaboost ensemble learning using the maximum number of estimators at which the algorithm is terminated (n_estimator: 200), learning_rate: 1, and random_state: 0. After that, will compare the algorithms have done, aiming to assess which algorithm is more likely to be efficient and how this efficiency varies by hyperparameters tuning and reinforcement using ensemble learning on different problems. Fig. 3 shows the performance of Bayesian hyperparameter and Adaboost ensemble learning on the ML model concerning the error function based on MAE, RMSE, and R^2.

The performance of each model varies depending on the dataset. Base learners model with parameter tuning using Bayesian which produces the most accurate predictions. In Fig. 3, it shows that the algorithm that has the best performance in almost all datasets is RF. RF achieved the highest accuracy in cocomo81, ISBSG10, kitchenham, maxwell, nasa93, and UCP. Meanwhile, CART, KNN, MLP, and SVR have almost similar performance. These results show that CART, KNN, MLP, and SVR are not very sensitive to parameters tuning, while RF is very sensitive to parameters tuning which results in stable prediction performance. This suggests that the best parameters to use with a machine learning approach may change over time.

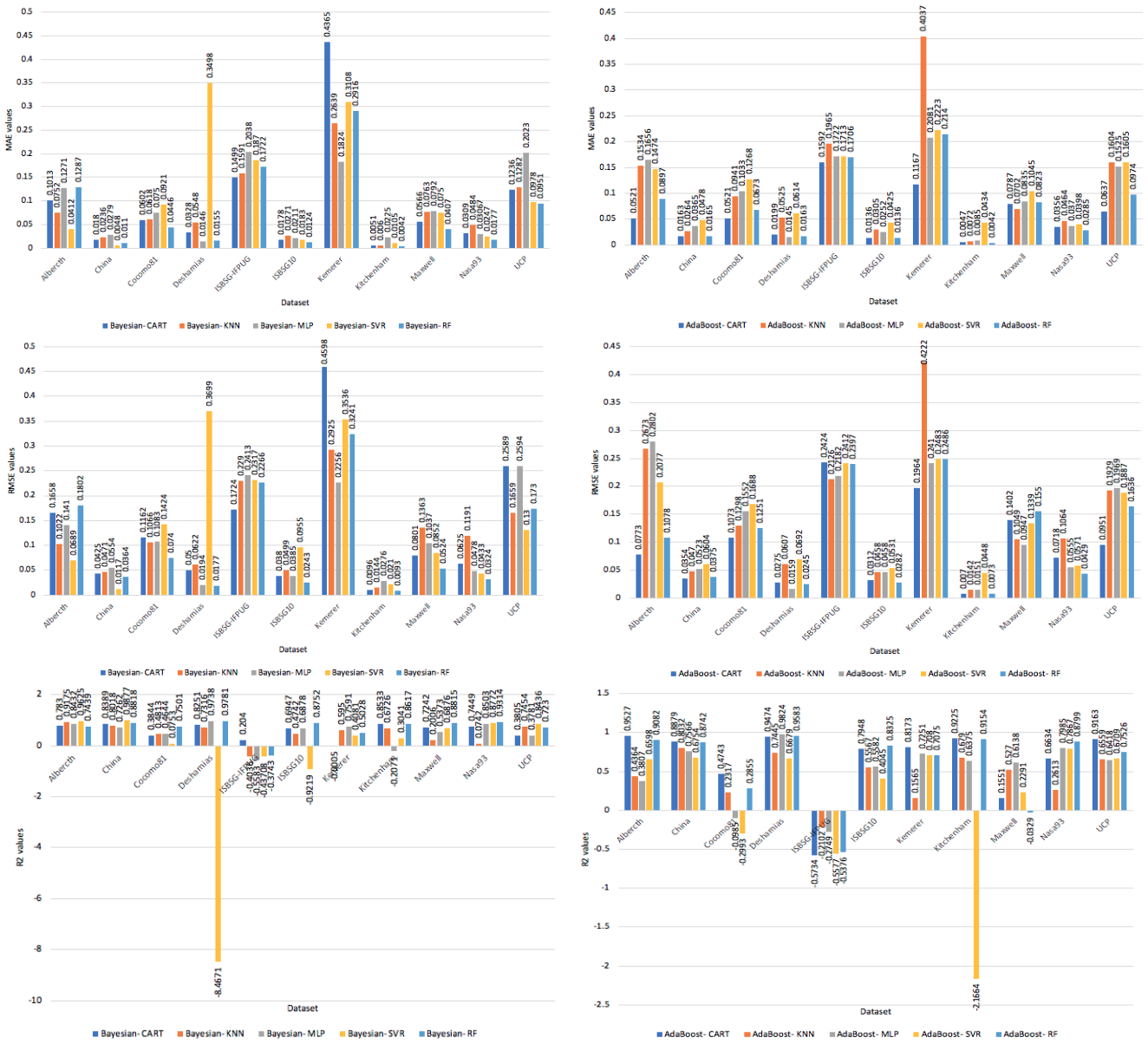
As for the base learner model with Adaboost ensemble learning, it shows different results. Where the algorithm that has the best performance is CART, followed by RF as the second algorithm that has the best performance. While KNN, MLP, and SVR have almost similar performance. For CART, obtain the highest accuracy in albrecht, china, cocomo81, ISBSG10, IFPUG, kemerer, and UCP. This analysis of different optimization approaches reveals that the Adaboost ensemble learning optimization is the clear victor, as it can create a model with the highest test accuracy for eleven data sets. To summarize, the meta-parameter analysis for Adaboost, which was used to strengthen the basic CART model, significantly outperforms other models (on this dataset).

C. The Best Model using Adaboost with Bayesian Hyperparameter Optimization

The same experiment used the ML algorithm to set the Adaboost Ensemble learning parameters using Bayesian hyperparameter optimization. In this paper, will repeated the experiment to find the best convergence with iterations from 10 to 200. The effect of the ML model on setting the Bayesian hyperparameter values of the Adaboost ensemble model is presented in Table VI to VIII.

In particular, when measured in MAE, RMSE, and R^2 (Table VI to VIII), RF and SVR achieve the best average ratings, followed by MLP, and CART, while k-NN with slightly worse average ratings among all related methods. In this respect, RF, SVR, and MLP have advantages over other methods in most datasets with medium/large effect sizes but perform worse than CART and k-NN in many datasets with small effect sizes. CART and k-NN perform best on data sets with small effect sizes. No significant differences could be found among the three methods RF, SVR, and MLP had similar overall performance and were superior to CART and k-NN with medium/large effect sizes depending on the data set. Nonetheless, the RF method is more consistent among the best methods regardless of the metric.

This experiment shows that overall, the five machine learning models that are strengthened by the Bayesian gaussian process and Adaboost ensemble learning have almost the same performance in all datasets used. However, it can be determined that RF, SVR, and MLP have the best results in this area.



(a) Baseline ML with Bayesian gaussian Process (MAE, RMSE, and R^2). (b) Baseline ML with Adaboost ensemble learning (MAE, RMSE, and R^2).

Fig. 3. Comparison Algorithm: (a) Baseline ML with Bayesian Gaussian Process; (b) Baseline ML with Adaboost Ensemle Learning.

TABLE VI. COMPARISON MAE USING BAYESIAN HYPERPARAMETER TUNING WITH ADABOOST ENSEMBLE LEARNING

Dataset	MAE (Bayesian optimization-Adaboost ensemble learning)				
	CART	KNN	MLP	SVR	RF
Albrecht	0.1463	*0.1888*	0.1694	**0.0490**	0.1070
China	0.0396	0.0436	*0.0528*	**0.0243**	0.0359
Cocomo81	**0.0557**	0.1462	0.1563	*0.1603*	0.1600
Desharnais	0.0166	*0.0482*	0.0277	**0.0081**	0.0210
IFPUG	0.2431	0.1849	**0.1771**	*0.2713*	0.1842
ISBSG10	0.0326	0.0452	*0.0476*	0.0436	**0.0241**
Kemerer	*0.4681*	0.2340	**0.2317**	0.3388	0.3521
Kitchenham	0.0079	0.0130	*0.0663*	0.0195	**0.0068**
Maxwell	0.1114	**0.0753**	0.0909	0.0924	*0.1431*
Nasa93	0.0653	0.1101	0.0729	*0.8203*	**0.0456**
UCP	**0.1131**	0.1612	*0.1969*	0.1460	0.1574

TABLE VII. COMPARISON RMSE USING BAYESIAN HYPERPARAMETER TUNING WITH ADABOOST ENSEMBLE LEARNING

Dataset	RMSE (Bayesian optimization-Adaboost ensemble learning)				
	CART	KNN	MLP	SVR	RF
Albrecht	0.1463	*0.1888*	0.1694	**0.0490**	0.1070
China	0.0396	0.0436	*0.0528*	**0.0243**	0.0359
Cocomo81	**0.0557**	0.1462	0.1563	*0.1603*	0.1600
Desharnais	0.0166	*0.0482*	0.0277	**0.0081**	0.0210
IFPUG	0.2431	0.1849	**0.1771**	*0.2713*	0.1842
ISBSG10	0.0326	0.0452	*0.0476*	0.0436	**0.0241**
Kemerer	*0.4681*	0.2340	**0.2317**	0.3388	0.3521
Kitchenham	0.0079	0.0130	*0.0663*	0.0195	**0.0068**
Maxwell	0.1114	**0.0753**	0.0909	0.0924	*0.1431*
Nasa93	0.0653	0.1101	0.0729	*0.8203*	**0.0456**
UCP	**0.1131**	0.1612	*0.1969*	0.1460	0.1574

TABLE VIII. COMPARISON R^2 USING BAYESIAN HYPERPARAMETER TUNING WITH ADABOOST ENSEMBLE LEARNING

Dataset	R^2 (Bayesian optimization-Adaboost ensemble learning)				
	CART	KNN	MLP	SVR	RF
Albrecht	0.8311	*0.7190*	0.7737	**0.9810**	0.9096
China	0.8602	0.8301	*0.7517*	**0.9473**	0.8847
Cocomo81	**0.8582**	0.0247	-0.1146	*-0.1723*	-0.1677
Desharnais	0.9807	*0.8386*	0.9468	**0.9954**	0.9693
IFPUG	-0.5819	0.0844	**0.1600**	*-0.9702*	0.0913
ISBSG10	0.7758	0.5687	*0.5220*	0.5987	**0.8772**
Kemerer	*-0.0370*	0.7408	**0.7458**	0.4566	0.4133
Kitchenham	0.9006	0.7320	*-5.9438*	0.3952	**0.9264**
Maxwell	0.4667	**0.7561**	0.6448	0.6329	*0.1194*
Nasa93	0.7216	0.2077	0.6527	*-42.901*	**0.8639**
UCP	**0.8818**	0.7597	*0.6418*	0.8030	0.7710

V. CONCLUSION

An enhanced hyperparameter tuning approach on an ensemble learning algorithm will be evaluated for its impact on model accuracy and stability in this study. The parameters of five machine learning models trained on eight datasets from the PROMISE repository and two subsets of data from the ISBSG R10/R18-IFPUG dataset are adjusted using this tuner. This study applies a state-of-the-art method by combining Bayesian-based gaussian processes with Adaboost ensemble learning to improve ML performance in a SEE context. Tuning, training, evaluation, and cross-validation will all be used in this project. The findings of this study show that optimizing machine learning models can considerably improve their performance. The implementation of AdaBoost ensemble learning and Bayesian hyperparameter optimization can improve the performance of the RF method. RF outperformed other methods in almost all datasets. As such, AdaBoost ensemble learning is the optimization that impacts machine learning model performance across all data sets in this scenario. On the other hand, the Bayesian optimization approach based on the Gaussian process to improve the performance of machine learning prediction models can achieve high accuracy in some cases.

More empirical research could be conducted in the future to support the conclusions of this study and to acquire knowledge utilizing different data sets. Additionally, compared or investigated various different optimization strategies, particularly for classification issues. It's also crucial to test the efficacy of various feature selection approaches, as well as increase with optimization tuning, when estimating software effort.

REFERENCES

[1] L. Song, L. L. Minku, and X. Yao, "A novel automated approach for software effort estimation based on data augmentation," ACM on Eur. Soft. Eng. Conf. and Symp. on the Found. of Soft. Eng., pp. 468–479 , November 2018.

[2] S. S. Gautam and V. Singh, "The state-of-the-art in software development effort estimation," J. of Software: Evolution and Process , Vol. 30, No. 12 , May 2018.

[3] M. Usman, K. Petersen, J. Börstler, and P. Santos Neto, "Developing and using checklists to improve software effort estimation: A multi-case study," J. of Systems and Software , Vol. 146 , pp. 286–309 , September 2018.

[4] S. Ezghari and A. Zahi, "Uncertainty management in Software effort estimation using a consistent fuzzy analogy-based method," J. of Appl. Soft. Comp., Vol. 67 , pp. 540–557 , 2018.

[5] M. Azzeh, "Software Effort Estimation Based on Optimized Model Tree," PROMISE, 2011.

[6] S. K. Palaniswamy and R. Venkatesan, "Hyperparameters tuning of ensemble model for software effort estimation," J. of Amb. Intell. and Human. Comp., No. 0123456789 , 2020.

[7] P. Pospieszny, B. Czarnacka-Chrobot, and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms," J. of Systems and Software , Vol. 137, No. January , pp. 184–196 , 2018.

[8] R. Alizadehsani et al., "Handling of uncertainty in medical data using machine learning and probability theory techniques: a review of 30 years (1991–2020), " Springer US, 2021.

[9] J. Novaković, P. Strbac, and D. Bulatović, "Toward optimal feature selection using ranking methods and classification algorithms," J. of Op. Res., Vol. 21, No. 1 , pp. 119–135 , 2011.

[10] Y. Mahmood, N. Kama, A. Azmi, A. S. Khan, and M. Ali, "Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation," J. of Soft. Pract. and Exp., 2021.

[11] S. K. Palaniswamy and R. Venkatesan, "Hyperparameters tuning of ensemble model for software effort estimation," J. of Amb. Intell. and Human. Comp., Vol. 12, No. 6 , pp. 6579–6589 , 2021.

[12] H. Zhao, X. Chen, T. Nguyen, J. Z. Huang, G. Williams, and H. Chen, "Stratified over-sampling bagging method for random forests on imbalanced data," Springer Int. Pub. Switzerland , Vol. 9650 , pp. 63–72 , 2016.

[13] G. W. Cha, H. J. Moon, and Y. C. Kim, "Comparison of random forest and gradient boosting machine models for predicting demolition waste based on small datasets and categorical variables," Int. J. of Env. Res. and Pub. Heal. MDPI, Vol. 18, No. 16 , 2021.

[14] X. Ren et al., "A Dynamic Boosted Ensemble Learning Method Based on Random Forest," arXiv preprint, 2018.

[15] Z. Abdelali, H. Mustapha, and N. Abdelwahed, "Investigating the use of random forest in software effort estimation," Procedia Computer Science , Vol. 148 , pp. 343–352 , 2019.

[16] A. Zakrani, M. Hain, and A. Namir, "Software development effort estimation using random forests: An empirical study and evaluation," Int. J. of Int. Eng. and Sys., Vol. 11, No. 6 , pp. 300–311 , 2018.

[17] S. M. Satapathy, B. P. Acharya, and S. K. Rath, "Early stage software effort estimation using random forest technique based on use case points," IET Software, Vol. 10, No. 1 , pp. 10–17 , 2016.

[18] W. Zhang, C. Wu, H. Zhong, Y. Li, and L. Wang, "Prediction of undrained shear strength using extreme gradient boosting and random forest based on Bayesian optimization," Geo. Front., Vol. 12, No. 1 , pp. 469–477 , 2021.

[19] S. Lessmann, B. Baesens, H. V. Seow, and L. C. Thomas, "Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research," Eur. J. of Oper. Res., Vol. 247, No. 1 , pp. 124–136 , 2015.

[20] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," IEEE Trans. on Soft. Eng., Vol. 38, No. 6 , pp. 1403–1416 , 2012.

[21] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," J. of Mach. Learn. Res., Vol. 15 , pp. 3133–3181, 2014.

[22] S. Shukla, S. Kumar, and P. R. Bal, "Analyzing effect of ensemble models on multi-layer perceptron network for software effort estimation," IEEE World Cong. on Serv., Vol. 2642–939X, pp. 386–387 , 2019.

[23] Y. Ren, L. Zhang, and P. N. Suganthan, "Ensemble Classification and Regression – Recent Developments, Applications and Future Direction," IEEE Comput. Intell. Mag , Vol. 11, No. 1 , pp. 41–53 , 2016.

[24] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," J. of Mach. Learn. Res., Vol. 12 , pp. 2825–2830 , 2011.

[25] P. Phannachitta and K. Matsumoto, "Model-based software effort estimation - A robust comparison of 14 algorithms widely used in the data science community," Int. J. of Inn. Comp., Vol. 15, No. 2 , pp. 569–589 , 2019.

[26] I. Martin-Diaz, D. Morinigo-Sotelo, O. Duque-Perez, and R. J. De Romero-Troncoso, "Early Fault Detection in Induction Motors Using AdaBoost with Imbalanced Small Data and Optimized Sampling," IEEE Trans. on Ind. App., Vol. 53, No. 3 , pp. 3066–3075 , 2017.

[27] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," Annals of Statistics, Vol. 26, No. 5 , pp. 1651–1686 , 1998.

[28] T. Xia, R. Krishna, J. Chen, G. Mathew, X. Shen, and T. Menzies, "Hyperparameter Optimization for Effort Estimation," arXiv preprint, Vol. 4 , 2018.

[29] M. Hosni, A. Idri, A. Abran, and A. B. Nassif, "On the value of parameter tuning in heterogeneous ensembles effort estimation," J. of Soft Comp., pp. 1–34 , 2017.

[30] A. Kumar, B. D. . Patro, and B. K. Singh, "Parameter Tuning for Software Effort Estimation Using Particle Swarm Optimization Algorithm," Int. J. of App. Eng. Res., Vol. 14, No. 2 , pp. 139–144 , 2019.

[31] P. Phannachitta, "On an Optimal Analogy-based Software Effort Estimation," Inf. and Soft. Tech., Vol. 125, No. June 2019 , p. 106330 , 2020.

[32] L. Song, L. L. Minku, and X. Yao, "The impact of parameter tuning on software effort estimation using learning machines," ACM Int. Conf. Proc. Ser., Vol. Part F1288 , 2013.

[33] A. Arcuri and G. Fraser, "Parameter tuning or default values? An empirical investigation in search-based software engineering," Int. Sym. on Sear. Bas. Soft. Eng., pp. 33–47 , 2011.

[34] M. M. Ozturk, "The impact of parameter optimization of ensemble learning on defect prediction," Comp. Sci. J. of Mol., Vol. 27, No. 1 , pp. 85–128 , 2019.

[35] L. L. Minku and X. Yao, "An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation," ACM Int. Conf. Proc. Ser., Vol. Part F1288 , 2013.

[36] R. Shu, T. Xia, J. Chen, L. Williams, and T. Menzies, "Improved Recognition of Security Bugs via Dual Hyperparameter Optimization," arXiv , 2019.

[37] J. Wu, X. Y. Chen, H. Zhang, L. D. Xiong, H. Lei, and S. H. Deng, "Hyperparameter optimization for machine learning models based on Bayesian optimization," J. of Elect. Sci. and Tech., Vol. 17, No. 1 , pp. 26–40 , 2019.

[38] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," Ann. Conf. on Neu. Inf. Proc. Sys., pp. 1–9 , 2011.

[39] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," arXiv preprint , 2010.

[40] J. C. Lévesque, C. Gagné, and R. Sabourin, "Bayesian hyperparameter optimization for ensemble learning," Conf. on Uncer. in Art. Intell., UAI 2016 , pp. 437–446 , 2016.

[41] L. L. Minku and X. Yao, "A Principled Evaluation of Ensembles of Learning Machines for Software Effort Estimation Categories and Subject Descriptors," Proc. of Int. Conf. on Pred. Mod. in Soft. Eng., Banff Alberta, Canada , 2011.

[42] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "Using tabu search to configure support vector regression for effort estimation," Emp. Soft. Eng., Vol. 18, No. 3 , pp. 506–546 , 2013.

[43] M. O. Elish, "Assessment of voting ensemble for estimating software development effort," Proc. of the IEEE Sym. on Comp. Intell. and Dat. Min., SSCI 2013, pp. 316–321 , 2013.

[44] L. Villalobos-Arias and C. Quesada-López, "Comparative study of random search hyper-parameter tuning for software effort estimation, " Ass. for Comp. Mach., Vol. 1, No. 1, 2021.

[45] A. Zakrani, A. Najm, and A. Marzak, "Support Vector Regression Based on Grid-Search Method for Agile Software Effort Prediction," Coll. in Inf. Sci. and Tech., Vol. 2018-Octob , pp. 492–497 , 2018.

[46] T. Xia, R. Shu, X. Shen, and T. Menzies, "Sequential Model Optimization for Software Effort Estimation," IEEE Trans. on Soft. Eng., Vol. 5589, No. c, pp. 1–16, 2020.

[47] L. Villalobos-Arias, C. Quesada-López, J. Guevara-Coto, A. Martínez, and M. Jenkins, "Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation," ACM Int. Conf. on Pred. Mod. and Dat. Analy. in Soft. Eng., pp. 31–40 , 2020.

[48] M. Azzeh, A. B. Nassif, and S. Banitaan, "Comparative analysis of soft computing techniques for predicting software effort based use case points," IET Software, Vol. 12, No. 1 , pp. 19–29 , 2018.

[49] L. Song, L. L. Minku, and Y. A. O. Xin, "Software effort interval prediction via Bayesian inference and synthetic bootstrap resampling," ACM Trans. on Soft. Eng. and Meth., Vol. 28, No. 1 , 2019.

[50] J. Huang, Y. F. Li, J. W. Keung, Y. T. Yu, and W. K. Chan, "An empirical analysis of three-stage data-preprocessing for analogy-based software effort estimation on the ISBSG data," Proc. of Int. Conf. on Soft. Qual. Relia. and Sec., Prague, Czech Republic , pp. 442–449 , 2017.

[51] J. Huang, Y. Li, and M. Xie, "An empirical analysis of data preprocessing for machine learning-based software cost estimation," J. of Inf. and Soft. Tech., Vol. 67 , pp. 108–127 , 2015.

[52] A. Famili, W. M. Shen, R. Weber, and E. Simoudis, "Data preprocessing and intelligent data analysis," Intell. Dat. Analy., Vol. 1, No. 1 , pp. 3–23 , 1997.

[53] S. Viaene, G. Dedene, and R. A. Derrig, "Auto claim fraud detection using Bayesian learning neural networks," J. of Exp. Sys. with App., Vol. 29, No. 3 , pp. 653–666 , 2005.

[54] E. Fitkov-Norris, S. Vahid, and C. Hand, "Evaluating the Impact of Categorical Data Encoding and Scaling on Neural Network Classification Performance: The Case of Repeat Consumption of Identical Cultural Goods," J. of Comm. in Comp. and Inf. Sci., Vol. 311 , pp. 343–0352 , 2012.

[55] I. Abnane, M. Hosni, A. Idri, and A. Abran, "Analogy Software Effort Estimation Using Ensemble KNN Imputation," Proc. of Int. Conf. On Soft. Eng. and Adv. App., Kallithea, Greece , pp. 228–235 , 2019.

[56] S. Mensah, J. Keung, S. G. MacDonell, M. F. Bosu, and K. E. Bennin, "Investigating the Significance of the Bellwether Effect to Improve Software Effort Prediction: Further Empirical Study," IEEE Trans. on Relia. Vol. 67, No. 3 , pp. 1176–1198 , 2018.

[57] M. R. Segal, "Machine Learning Benchmarks and Random Forest Regression," Biostatistics, pp. 1–14 , 2004.

[58] Z. Zhang, J. T. Kwok, and D. Y. Yeung, "Parametric distance metric learning with label information," IJCAI Inte. Joint Conf. on Art. Intell., pp. 1450–1452 , 2003.

[59] R. Polikar, "Ensemble based systems in decision making," IEEE Cir. and Sys. Mag., Vol. 6, No. 3 , pp. 21–44 , 2006.

[60] A. Sharafati, S. B. H. S. Asadollah, and M. Hosseinzadeh, "The potential of new ensemble machine learning models for effluent quality parameters prediction and related uncertainty," Proc. Saf. and Env. Prot., Vol. 140 , pp. 68–78 , 2020.

[61] D. L. Shrestha and D. P. Solomatine, "Experiments with AdaBoost.RT, an improved boosting scheme for regression," Neu. Comput., Vol. 18, No. 7 , pp. 1678–1710 , 2006.

[62] M. T. Young, J. Hinkle, A. Ramanathan, and R. Kannan, "HyperSpace: Distributed Bayesian Hyperparameter Optimization," Proc. Int. Sym. on Comp. Arch. and Hig. Perfor. Comput., SBAC-PAD 2018 , No. 1 , pp. 339–347 , 2019.

[63] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," Adv. in Neu. Inf. Proc. Sys., 2012.

[64] M. Feurer and F. Hutter, "Hyperparameter Optimization," Auto. Mach. Lear., The Springer Series, pp. 3–34 , 2019.

[65] R. Marco, S. S. S. Ahmad, and S. Ahmad, "Empirical Analysis of Software Effort Preprocessing Techniques Based on Machine Learning," Int. J. of Intell. Eng. and Sys., Vol. 14, No. 6 , pp. 554–567 , 2021.

[66] S. B. Imandoust and M. Bolandraftar, "Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events : Theoretical Background," Int. J. of Eng. Res. and App., Vol. 3, No. 5 , pp. 605–610 , 2013.

[67] E. Kocaguneli, T. Menzies, J. Hihn, and B. H. Kang, "Size doesn't matter? On the value of software size features for effort estimation," Proc. of Int. Conf. On Pred. Mod. in Soft. Eng., Lund, Sweden , pp. 89–98 , 2012.