Information and Software Technology 125 (2020) 106330

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

On an optimal analogy-based software effort estimation

Passakorn Phannachitta*

College of Arts, Media and Technology Chiang Mai University, 239 Suthep, Muang, Chiang Mai 50200, Thailand

ARTICLE INFO

Keywords:
Software effort estimation
Analogy
Effort adaptation
Hyperparameter optimization
Ensemble
Empirical experiments

ABSTRACT

Context: An analogy-based software effort estimation technique estimates the required effort for a new software project based on the total effort used in completing past similar projects. In practice, offering high accuracy can be difficult for the technique when the new software project is not similar to any completed projects. In this case, the accuracy will rely heavily on a process called effort adaptation, where the level of difference between the new project and its most similar past projects is quantified and transformed to the difference in the effort. In the past, attempts to adapt to the effort used machine learning algorithms; however, no algorithm was able to offer a significantly higher performance. On the contrary, only a simple heuristic such as scaling the effort by consulting the difference in software size was adopted.

Objective: More recently, million-dollar prize data-science competitions have fostered the rapid development of more powerful machine learning algorithms, such as the Gradient boosting machine and Deep learning algorithm. Therefore, this study revisits the comparison of software effort adaptors that are based on heuristics and machine learning algorithms.

Method: A systematic comparison of software effort estimators, which they all were fully optimized by Bayesian optimization technique, was carried out on 13 standard benchmark datasets. The comparison was supported by robust performance metrics and robust statistical test methods.

Conclusion: The results suggest a novel strategy to construct a more accurate analogy-based estimator by adopting a combined effort adaptor. In particular, the analogy-based model that adapts to the effort by integrating the Gradient boosting machine algorithm and a traditional adaptation technique based on productivity adjustment has performed the best in the study. Particularly, this model significantly outperformed various state-of-the-art effort estimation techniques, including a current standard benchmark algorithmic-based technique, analogy-based techniques, and machine learning-based techniques.

1. Introduction

Similar effort is required to develop similar software projects is the essential principle of the Analogy-based software effort estimation technique (*ABE*) [31,64], one of the most widely adopted techniques in both research communities and practice [27]. *ABE* offers not only outstanding performance, but also an intuitively straightforward principle, which is an important factor in facilitating its wide application. However, because *ABE* reuses the effort value of the most similar past completed software project to estimate the effort required for the new project, obtaining high accuracy when the new project is not closely identical to any past completed projects is difficult. In this case, the estimated effort is adjusted by an imperative process known as effort adaptation [6,35], where the difference in characteristics between the new project and its past similar projects are quantified and transformed into the difference between effort.

Supported by robust statistics, [54] concluded that heuristic-based effort adaptation techniques, such as Linear size adaptation (*lsa*) [69] and Regress towards the mean (*rtm*) [26,61] are the current state-of-the-art adaptation techniques for determining effort. These techniques significantly outperformed the other common techniques including an effort adaptor constructed with a machine learning algorithm called artificial neural networks (*ann*) [44]. However, it remains inconclusive whether machine learning algorithms are unsuitable for software effort adaptation since, elsewhere, it has been widely suggested that an estimation obtained from a machine learning algorithm is usually more accurate than its heuristic counterpart [34,66]. Moreover, *ann* is usually considered a moderately performing algorithm, which typically performs poorer than many more recent algorithms, such as Gradient boosting machines (*gbm*) and Deep learning (*deep*) [17].

More recently, the emergence of online data science competition communities, such as Kaggle [23,28], has accelerated the development

* Corresponding author.
E-mail address: passakorn.p@cmu.ac.th

https://doi.org/10.1016/j.infsof.2020.106330
Received 22 June 2019; Received in revised form 20 April 2020; Accepted 24 April 2020
Available online 6 May 2020

and enhancement of more advanced machine learning algorithms. Many recent algorithms, such as *gbm* and *deep*, are continually adopted for solutions addressing difficult real-world problems, such as algorithmic trading [1] and drug design [67]. Phannachitta and Matsumoto [55] investigated the performance of these recent algorithms as software effort estimators and reported many significant findings. The following two findings represent the essential motivations for the present study:

- The competition winning algorithms, *gbm*, which had not previously been explored in the area of software effort estimations, was a high performing algorithm for model-based software effort estimation.
- *ABE* offered almost the same outstanding accuracy as *gbm* even if effort adaptation was not considered in the experiment.

This study represents a continuation of the author's past two studies; namely, [54], and [55]. The main objective of the present study is to explore the potential performance of the *ABE* model, with its effort adapted by recent and outstanding machine learning algorithms, such as *gbm* and *deep*. The experiments in the present study are organized to answer these following three research questions:

RQ1 What is the overall performance of the *ABE* models with the effort adaptation process undertaken by machine learning algorithms?. A systematic comparison was carried out to compare the performance of ten software effort adaptors. Of these, seven are based on machine learning algorithms. Unexpectedly, the techniques based on a heuristic, i.e., the *lsa* and the *rtm* techniques, outperformed all other techniques.

RQ2 What is the potential performance of the effort adaptors utilizing both heuristic and machine learning algorithm together?. Further analysis was carried out over the datasets in which the *lsa* and the *rtm* techniques did not perform highly. The results suggest examining the potential accuracy achieved by combining the best heuristic-based technique and the best machine learning-based technique into one single estimator.

RQ3 How does the performance of the best *ABE* model discovered in the present study compare to other state-of-the-art software effort estimation techniques?. The *ABE* model that adapts to the effort by the best combined effort adaptor of the experiment in RQ2 was further compared with the current standard benchmark technique named Linear programming as a baseline for software effort estimation (*lp4ee*) [57], and eight other model-based estimators that performed outstandingly in [55]. The comparison results decisively suggest that the best *ABE* is a candidate of the new state-of-the-art model-based software effort estimator.

The details of the evaluation and analysis are organized in this paper as follows. Section 2 provides the essential background and related works. Section 3 explains all the effort adaptors along with their implementation in the study. Section 4 demonstrates the results for the three research questions. Section 5 further discusses the results and the validity of the threats of the study. Finally, the conclusion is provided in Section 6.

2. Background

2.1. Analogy-Based software effort estimation (ABE)

Accurately estimating the necessary effort in developing a software project is crucial for software development because an inaccurate estimation can potentially lead to a reduction in software quality or worse still, the potential of project cancellation [39]. *ABE* is widely adopted in both the research community and in industrial practice. The general principle of *ABE* is to reuse a past development experience by assuming that the amount of effort required for developing a new software project should be similar to that required for completing its most similar past projects [64]. In steps, *ABE* computes the level of similarity between the new project and all the existing past projects by using a distance function, such as Euclidian distance [53]. Next, *ABE* generates the effort using the mean effort value of the k most similar projects to be the estimated effort value of the new project. In the literature, this k value is usually set as one, three, or five [31,69]. This principle is straightforward and *ABE* is usually able to offer excellent estimation performance in practice [27,40,45,62].

2.1.1. Effort adaptation

Following the principles of *ABE*, the development team should be able to effortlessly estimate the effort for any new project if they already have solid experience in developing similar software projects, in the past. However, often in practice, it is common that the development team is asked to develop software whose requirements are not substantially similar to their past completed projects. In this case, estimating the effort using *ABE* requires a reliable approach to adjust the estimated effort based on the level of differences between the estimating project and its not-so-substantially similar past projects.

Let the term ABE_x denote the *ABE* model having the effort adjusted by an adaptation technique x. Among the common adaptation techniques, the technique named the Unweighted average of the effort (*uavg*) is the most straightforward technique used since its first proposal in [64]. Explained mathematically, *uavg* aggregates effort values of the selected k analogues by using an unweighted mean:

$$\hat{E}_{ABE_{uavg}}(P_{new}) = \frac{1}{k}\sum_{i=1}^{k} E(P_{analog(new,i)}), \tag{1}$$

where P_{new} and $P_{analog(new,\,i)}$ are the new project to be estimated and the i^{th} most similar project of P_{new}, respectively. E and $\hat{E}_{ABE_{uavg}}$ are the actual effort, and the estimated effort value generated by the ABE_{uavg} technique, respectively. In a past study of the author [54], eight common effort adaptation techniques including *uavg* were compared, based on the Stable ranking identification method [32], and the authors found that the heuristic-based techniques, such as *lsa* and *rtm* were the best among the eight techniques. *lsa*, in particular, is based on an exploitation of the linear extrapolation between a variable indicating the software size of the new project and its most similar past projects. Explained mathematically, *lsa* adapts to the effort by scaling the software size of past similar projects to that of the new project using the formula below:

$$\hat{E}_{ABE_{lsa}}(P_{new}) = \frac{SS(P_{new})}{\frac{1}{k}\sum_{i=1}^{k} SS(P_{analog(new,i)})} \times \frac{1}{k}\sum_{i=1}^{k} E(P_{analog(new,i)}) \tag{2}$$

where $SS(P_{new})$ is the software size of P_{new}. The adjusted function points (*afp*) are *SS* for the dataset that are available. Otherwise, *SS* is the number of lines of code.

For the *rtm* technique, first, *rtm* quantifies the productivity of all the projects by using the triangular relationship: $\hat{E} = SS \times Productivity$ proposed by Walkerden and Jeffery [69]. Next, *rtm* calibrates the productivity of new project to be consistent with each of the other past completed projects [26], and then multiplies the calibrated productivity with *SS* of the new project to generate the $\hat{E}$. The formula describing *rtm* is shown in Eq. (3).

$$\hat{E}_{ABE_{rtm}}(P_{new}) = SS(P_{new}) \times \left(\frac{1}{k}\sum_{i=1}^{k} Productivity(P_{analog(new,i)}) + \left(M - \frac{1}{k}\sum_{i=1}^{k} Productivity(P_{analog(new,i)}) \right) \times (1 - r) \right) \tag{3}$$

where M is the mean productivity of all past projects, and r is the correlation coefficient between the actual productivity of all past completed projects and the productivity of their single most similar projects.

2.1.2. Effort adaptation based on machine learning algorithms

Alternative approaches to heuristic-based adaptation include utilizing machine learning algorithms, such as neural networks [44]. As an effort adaptor, a machine learning model is built to learn the relationship

between the difference in effort and the quantified magnitude of difference in project characteristics. Using a machine learning algorithm, when a new project arrives for an estimation, the estimated effort value will be the summation between the effort generated by the ABE_{uavg} technique and the effort difference estimated by the learning model. Explained in steps below, an effort adaptor utilizing a machine learning algorithm is constructed as follows [5,44]:

Step 1. A new software project is set as P_{new}. Then, set all the other existing software projects as the training cases of this P_{new}, i.e., $P_{training}$.

Step 2. Let P_i denotes a project case in $P_{training}$. Then, construct a machine learning model that transforms the difference in project characteristics into the difference in effort using all the existing projects in $P_{training}$. That is, for each P_i, the model learns the effort effort difference in the form of [44]:

$$E(P_i) - \sum_{j=1}^{k} E(P_{analog(i,j)}) = w \sum_{j=1}^{k} (P_i - P_{analog(i,j)}), \quad (4)$$

where w is the weights which are determined by the training a machine-learning model to minimize the differences in project characteristics and the differences in effort, and k is the number of analogies, which is normally set as 1, 2, or 3 [47].

Step 3. Estimate the effort of P_{new} using the ABE_{uavg} technique, i.e., $\hat{E}_{ABE_{uavg}}(P_{new})$.

Step 4. Estimate the effort differences between $\hat{E}_{ABE_{uavg}}(P_{new})$ and its k closest analogues, i.e., $E(P_{analog(new,1.k)})$ by using the machine learning model constructed in Step 2. This will generate k different adjustment suggestions. Then, calculate the mean value of $E(P_{analog(new,1.k)})$ to represent the overall magnitude of the effort difference. Let the term $\delta(\hat{E}(P_{new}), E(P_{analog(new,1.k)}))$ denote this magnitude of the effort difference.

Step 5. Adjust the estimated effort generated in Step 3 by consulting the magnitude of the effort difference estimated in Step 4. Precisely, the *ABE* model constructed with an effort adaptor based on a machine learning algorithm x generated the estimated effort value in the form of $\hat{E}_{ABE_x}(P_{new}) = \hat{E}_{ABE_{uavg}}(P_{new}) + \delta(\hat{E}(P_{new}), E(P_{analog(new,1.k)}))$.

Even if this methodology is sound and considered robust, it remains inconclusive whether high accuracy and high robustness effort adaptation can be assisted using machine learning algorithms as discussed in [54].

2.1.3. Machine learning algorithms evaluated in this study

More recently, online data science competition communities have become a place the enterprises to seek skilled data scientists. The most current popular competition is known as [28], where the competitions offering million-dollar prizes are hosted. In the author's view, this highly competitive environment has a strong incentive for facilitating the acceleration of advancements in machine learning performance. According to [17], since 2015 when *gbm* and *deep* were introduced to the community, these two algorithms have quickly become the algorithms of choice for the competition entrants up to this time of writing. Alongside these two algorithms, the algorithms made available in the Scikit-learn python library have also become the set of machine learning algorithms of choice for data scientists for their experiments [1]. In this study, seven machine learning algorithms are examined. They all are selected from the list of the algorithms performing the best in each group of similar algorithms in the study of [55]. A brief explanation of the seven algorithms investigated in this study are as follows:

Ordinary least squares regression (ols). [3] represents a generalized linear model for the present study. *ols* fit a regression line through data that minimizes the sum of squared differences between the estimated effort and the actual efforts.

Classification and regression tree (cart). [11] represents a decision tree. *cart* generates a binary decision tree based on the Gini index that best explains the effort.

Support vector regression (svr). [19] represents the support vector machine. *svr* attempts to find a certain function that best fits the dataset based on optimization. It utilizes a kernel function to map the variables into a higher dimension if it fits better to the dataset. The use of kernel functions enables *svr* to handle non-linear regression, which is generally more robust than simple linear models.

ann. [25] represents neural networks. *ann* forms a network to mimic the human brain mechanism. The use of *ann* in solving an effort estimation problem is to learn the optimal weight values assigned to each edge in the network in order to produce minimum differences in the estimated effort and the actual effort.

deep. [43] represents a deep neural network. *deep* can be considered an *ann* with more than one hidden layer, so that *deep* generally forms a more complex network[17]. noted that *deep* is more suitable for unstructured data. Therefore, *deep* may not be fully appropriate for the problem of this study.

rf. [10] represents ensemble learning based on an extension over the bagging algorithm [3] which builds multiple regression trees over multiple randomly generated instance subsets, and simply uses the mean value from all the estimated values as the final estimation. A significant extension from bagging was proposed by Breiman [10], who suggested to apply feature reduction along with a subsetting of instances. According to the essential theory supporting ensemble learning, where an estimation error is accounted by bias, variance, and covariance [12,13], applying both feature reduction and instance subsetting can potentially increase the accuracy due to the reduction of the correlation between all the regression trees generated. According to [17], *rf* had been one of the most popular algorithms before the introduction of *gbm*.

gbm. [16] represents ensemble learning based on boosting. *gbm* first builds a regression tree over the entire training dataset. Then, it iteratively builds a weak estimator focusing on the training cases whose fit accuracy were relatively poor. The strength of *gbm* is that it adds an error-correction component to each iteration by feeding the generated error back to the model built in the succeeding iteration. Furthermore, *gbm* applies the gradient descent technique [21] to optimize the error reduction process to produce the resultant model with minimal overall error.Chollet [17] noted that since 2017, *gbm* has been dominating the Kaggle leaderboards for the problems in which structured data are available.

3. Evaluation methodologies

All the experiments undertaken in this study are based on the author's improvement of the common evolution methods used in empirical software engineering studies, known as the Stable ranking indication method [32]. The details of the improvement are explained throughout this section. In steps, the evaluation procedure of this study comprises: (1) experimental datasets selection, (2) training and test samples generation, (3) construction and optimization of the estimation process of multiple effort estimators to be compared, (4) performance evaluation based on different performance metrics, and (5) summarization of the results by using Brunner's statistical test method [14].

Note that, all the implementations related to the *ABE* technique, the machine learning algorithms, the normalization techniques, and the feature selection methods were developed using Python version 3.7.2, the Scikit-learn library version 0.20.2, the Keras library version 2.2.4, and the HyperOpt library version 0.2.2. The experiments were undertaken on the Amazon web services platform [4], using a c5.9xlarge compute instance of the Elastic compute cloud service (*Ec2*) to accelerate the experimentations with 36 virtual processors and a 72 GiB main memory.

Table 1
Experimental datasets.

	Datasets	#Features	#Instances	The development effort			
				Unit	Min	Median	Max
1	Albrecht [2]	7	24	months	0.5	11.5	105.2
2	Cocomo-sdr [7]	23	12	months	1	3.5	22
3	Cocomo81-e [65]	17	28	months	9	354	11,400
4	Cocomo81-non-e [65]	17	35	months	5.9	50	6400
5	Desharnais-cobol [64]	8	67	hours	805	3913.5	23,940
6	Desharnais-4GL [64]	8	10	hours	546	1123.5	5880
7	Finnish [36]	4	38	hours	460	5430	26,670
8	Kemerer [29]	5	15	months	23.2	130.3	1107.3
9	Maxwell [46]	24	62	hours	583	5189.5	63,694
10	Miyazaki94 [52]	7	48	months	5.6	38.1	1586
11	Nasa93-c1 [50]	20	12	months	24	66	360
12	Nasa93-c2 [50]	20	37	months	8.4	82	1350
13	Nasa93-c5 [50]	20	39	months	72	571.4	8211

For the statistical test, an R package named *WRS* proposed by [71] was used.

3.1. Datasets

Table 1 lists the 13 datasets which were used for the performance comparison in the study. They all are standard benchmark datasets [48,49] and were also used in [55]. From the table, the datasets whose names contain a postfix, such as Cocomo81-e are composite datasets. That is, the Cocomo81-e and Cocomo81-non-e datasets were homogenized from the well-known Cocomo81 datasets [65], where the -e and the -non-e postfixes indicate the projects completed in embedded mode and the non-embedded modes of the Cocomo principle, respectively. Likewise, the -cobol and the -4GL postfixes indicate the Desharnais projects [64], which were developed in the Cobol and the 4GL languages, respectively. Finally, the -c1, the -c2 and the -c5 postfixes indicate the Nasa projects developed from different centers.

In [55], by using the Kernel density plot technique as suggested by [37], all the datasets listed in Table 1 were proved to be independent. This test ensures the strong generalization ability of the performance results of the present study.

3.2. Training and test sample generation

For performance evaluation, all the experiments undertaken in the present study used the leave-one-out technique [41] to generate the training and test samples. The technique let one instance become the test instance of a model built from the remaining instances, and this process was repeated until all the instances were tested. The leave-one-out technique was selected even if its popular alternatives such as the 10-fold cross validation more frequently appear in other kinds of empirical software engineering research [8], mainly because the leave-one-out technique was already proven to be the most suitable techniques for evaluating multiple software effort estimators as suggested by Kocaguneli and Menzies [38].

Note that the author opted the 10-fold cross validation technique for the later hyperparameter optimization process because the search space of the optimization is exponentially larger than that of this training and test sample generation. To reduce the possible influence due to the randomness inherent in the 10-fold cross validation technique, all the tests were run ten times and used the average as the final results.

3.3. Machine learning model construction and optimization

To examine the performance of the *ABE* technique tailoring with a particular software effort adaptor, in all the experiments, a search-based technique [56,58] named Bayesian hyperparameter optimization [9] was adopted using a Python library named [24] to find the best data normalizer, feature subset selector, the k parameter of *ABE*, and the best model hyperparameter set for a particular machine learning algorithm. The choice of this search technique is different from other studies in software effort estimation where the grid search technique [68] is commonly used. The Bayesian hyperparameter optimization technique was selected based on a motivational finding from more recent prestigious studies on effort estimation, where search-based techniques have often significantly outperformed all the other techniques in the studies. For example, [18] demonstrated that an *svr* model configured using a search-based technique namely Tabu search has shown a higher accuracy compared to traditional *svr* models. This finding is in broad agreement with practices in the field of data science, where the Bayesian hyperparameter optimization technique is commonly used for model configuration [15,60,73].

3.3.1. Normalization techniques

Two candidate normalization techniques were examined including *Apply nothing* and *Interval 0–1*. *Apply nothing* means leaving all data values unadjusted. *Interval 0–1* normalizes each feature value x_i of a continuous feature x by subtracting the minimum value of x of x_i, and dividing the product by the difference between the maximum value of x and the minimum value of x. This technique is mostly used in the literature on software effort estimation, such as [40].

3.3.2. Feature selection methods

For the choices of the feature selector, three candidate techniques were examined including *Select all features, Recursive feature elimination*, and *Principal component analysis. Select all features* is the selection of all the explanatory variables of the dataset. *Recursive feature elimination* iteratively removes variables from the dataset until the termination criteria is met. For example, the default configuration of the Scikit-learn implementation eliminates half the variables from the initial set. For each iteration, the *Recursive feature elimination* fits a learning model, such as, *ols*, over all the possible variable subsets with the size of the current subset minus one. Then, the *Recursive feature elimination* ranks the accuracy score, such as, the mean squared error, of all the models, and eliminates the variable associated with the model with the least accuracy. *Principal component analysis* transforms all the explanatory variables into a new lower-dimensional space without dropping any important information. This transformation is done by decomposing the entire dataset into orthogonal components that explain the maximum variance.

3.3.3. Number of analogies (k)

The common approach used in the literature on *ABE* is to fix k as a static number or a range of numbers in the experiments. For example, [69] examined $k = 1$, [44] used $k = 5$, and [47] examined k in a range from 1 to 3. However, different from these past studies, the present study treats this k as a parameter that can be fully optimized by using the

Table 2
Learning algorithms and their set of their hyperparameters for optimization. C is the penalty parameter of the error term used in training an *svr* model. N is the number of instances in a training dataset.

Algorithm	Model hyperparameter set
ols	-
cart	minimum samples split $\in$ [2, $\lceil\sqrt{N}\rceil$]
svr	kernel $\in$ {linear,rbf}, C $\in$ [1, 1000]
ann	hidden layer sizes $\in$ [2, $\lceil log_2(N)\rceil$], activation = relu, solver = adam
deep	hidden layer sizes $\in$ [2, $\lceil log_2(N)\rceil$], number of hidden layers $\in$ {2, 3, 4}, batch size = N, epochs = 20, activation = relu, solver = adam
rf	number of estimators $\in$ [2, 256], minimum samples split $\in$ [2, $\lceil\sqrt{N}\rceil$], max depths $\in$ [2, $\lceil\sqrt{N}\rceil$]
gbm	number of estimators $\in$ [2, 256], minimum samples split = [2, $\lceil\sqrt{N}\rceil$], max depths $\in$ [2, $\lceil\sqrt{N}\rceil$], learning rate $\in$ [0.001, 0.1]

Bayesian optimization techinque. In other words, the author used the optimization technique to determine the optimal k for every experiment.

3.3.4. Model hyperparameters

Hyperparameter optimization is an important process used for machine learning models to achieve optimal performance [72]. It is a process of finding the most appropriate combination of hyperparameters toward maximizing the performance of a machine learning model, given a dataset. Table 2 presents the set of different hyperparameter sets that the author's attempted to optimize for each of the seven machine learning algorithms examined in the study.

3.4. Multiple performance metrics

In the literature on software effort estimation, performance is measured in terms of estimation errors [20,22,32]. To maximize the credibility of the performance conclusion, recent publications have generally used more than one metric and assessed the overall performance of an estimator by aggregating the rank generated from each metric using the mathematical mean. In the present study, the same six performance metrics used in [55] were used. Specifically, they are the Mean absolute error (*mae*), the Relative standard deviation (*rsd*), the Logarithmic standard deviation (*lsd*), the Mean of the balanced relative error (*mbre*), the Magnitude of error relative to the estimate (*mmer*), and the Standardized accuracy (*sa*). The equations explaining these metrics are as follows:

$$mae = \frac{1}{N}\sum_{i=1}^{N} |E(P_i) - \hat{E}(P_i)|, \tag{5}$$

where N is the total number of instances of a dataset, $E(P_i)$ and $\hat{E}(P_i)$ denote the actual effort and the estimated effort for a P_i, that is, for the i^{th} instance of the dataset, respectively.

$$rsd = \sqrt{\frac{\sum_{i=1}^{N}\left(\frac{E(P_i)-\hat{E}(P_i)}{SS_i}\right)^2}{N-1}}, \tag{6}$$

where SS_i is a variable indicating software size. For a dataset where the adjusted function points (*afp*) are available, SS_i will be *afp*. Otherwise, the line of codes will be used.

$$lsd = \sqrt{\frac{\sum_{i=1}^{N}\left(e_i - \left(-\frac{s^2}{2}\right)\right)^2}{N-1}}, \tag{7}$$

where e_i is given by $ln(E(P_i)) - \ln(\hat{E}(P_i))$, and s^2 is the variance of e_i. As noted in [20], the mean and the variance of errors of a model used by the logarithm are equal to $s^2/2$ and s^2, respectively, if they exhibit normal distribution.

$$mbre = \frac{1}{N}\sum_{i=1}^{N}\frac{|E(P_i) - \hat{E}(P_i)|}{min(E(P_i), \hat{E}(P_i))} \tag{8}$$

$$mmer = \frac{1}{N}\sum_{i=1}^{N}\frac{|E(P_i) - \hat{E}(P_i)|}{\hat{E}(P_i)} \tag{9}$$

$$sa = 1 - \frac{mae_{estimator_j}}{mae_{random}}, \tag{10}$$

where $\text{mae}_{estimator_j}$ is the *mae* of the j^{th} effort estimator being evaluated, and mae_{random} is the *mae* of a large number of, for example 1000, randomized trials [42,63]. *sa* shows how much better $estimator_j$ is than to guess the effort by simply picking an effort value from the other cases in the dataset.

3.5. Pairwise comparison

Following the Stable ranking indication method [32], the overall performance of an estimator is determined by the total number of times it was statistically significantly outperformed by the other estimators, where, 0 time indicates the maximum achievable performance result. This counting method is known as *win-tie-loss* statistics, mainly used in recent literature on empirical software engineering, such as in [8,39,40,58] and [57]. For the testing of statistical significance, [37] recommended Brunner's test [14] as the most suitable test method for small datasets or the datasets that may not underlie normal distribution, where the common software effort estimation datasets are of this size and distribution.

In the experiment, a counter named *losses* was used to indicate the overall preference of an estimator. When the Brunner's test reported the p-value less than 0.05, which indicates that the performance between any pair of estimators being evaluated is significantly different, the *losses* associated with the estimator with the higher error will be incremental. After all the pairs of estimators were tested over all the six performance metrics, the estimators whose *losses* value were the lowest, were ultimately the relative highest performers. Suggested by [32], this evaluation method can offer a stable performance if the estimators based on similar algorithms are ranked closely to each other.

Table 3 shows a summary of all the *ABE* variants compared in the study. In this table, to the best of the author's knowledge, the entries, which do not have any citations associated with their full names, are the *ABE* variants that have not been examined in any other study.

4. Results

4.1. RQ1 What is the overall performance of the ABE models with the effort adaptation process undertaken by machine learning algorithms?

Table 4 shows the performance results of the ten effort estimators of Table 3 evaluated over the 13 experimental datasets. For each dataset, the maximum *losses* an effort estimator can achieve is equal to the multiplication between the number of estimators it has been compared to,

Table 3
Summary of ten effort estimation techniques compared in this study.

	Abbr.	Effort adaptors	Type	Implementation or library used in the study [30,59]
1	ABE_{uavg}	Unweighted mean of the k analogues [62]	Simple aggregation	Eq. 1
2	ABE_{lsa}	Linear size adaptation [69]	Heuristic	Eq. 2
3	ABE_{rtm}	Regression towards the mean [26]	Heuristic	Eq. 3
4	ABE_{ols}	Ordinary least squares regression	Machine learning	Sklearn.linear_model.LinearRegression
5	ABE_{cart}	Classification and regression tree	Machine learning	Sklearn.tree.DecisionTreeRegressor
6	ABE_{svr}	Support vector regression	Machine learning	Sklearn.svm.SVR
7	ABE_{ann}	Artificial neural networks [44]	Machine learning	Sklearn.neural_network.MLPRegressor
8	ABE_{deep}	Deep neural networks	Machine learning	Keras.wrappers.scikit_learn.KerasRegressor
9	ABE_{rf}	Random forest	Machine learning	Sklearn.ensemble.RandomForestRegressor
10	ABE_{gbm}	Gradient boosting machine	Machine learning	Sklearn.ensemble.GradientBoostingRegressor

Table 4
Comparison results showing the total *losses* in ascending order, and the number of datasets the estimators achieved high, moderate, and low performance, indicated by proportional to the maximum achievable *losses*. Note that lower *losses*, the higher number of datasets achieving *losses* $\in$ [0%, 5%), and the lower number of datasets achieving *losses* $\in$ [15%, 100%] indicate better performance.

Rank	Algorithm	Total losses	#Datasets having *losses* $\in$		
			[0%, 5%)	[5%,15%)	[15%,100%]
1	ABE_{rtm}	9	11/13	2/13	0/13
2	ABE_{lsa}	33	8/13	4/13	1/13
3	ABE_{gbm}	42	6/13	6/13	1/13
4	ABE_{ann}	52	7/13	4/13	2/13
5	ABE_{svr}	57	5/13	6/13	2/13
6	ABE_{uavg}	59	8/13	1/13	4/13
7	ABE_{deep}	85	6/13	2/13	5/13
8	ABE_{rf}	99	2/13	6/13	6/13
9	ABE_{ols}	170	3/13	2/13	8/13
10	ABE_{cart}	228	2/13	1/13	10/13

and the number of performance metrics being used, that is $9 \times 6 = 54$. Correspondingly, the maximum *losses* an estimator can achieve across 13 datasets is $54 \times 13 = 702$. In the past, many studies which utilized the *win-tie-loss* statistics simply aggregated the ranks for each estimator across all the experimental datasets, by using the mathematical mean. However, Phannachitta and Matsumoto [55] observed that tie ranks usually appeared in a way where the mean rank may not be fully appropriate in illustrating the performance of the compared estimators. In turn, the *losses* values achieved by each estimator were divided into bands based on their proportion to the maximum achievable *losses*. Specifically, for each dataset, the total *losses* were divided into three bands: *losses* $\in$ [0%, 5%), *losses* $\in$ [5%, 15%), and *losses* $\in$ [15%, 100%] of the maximum achievable *losses*, and the estimators were classified into the best performers, the moderate performers, and the poor performers, respectively, based on the number of datasets presenting in each band. In this study, the number of bands and thresholds are set the same as that of [55]. That is, 5% and 15% of the total *losses* per dataset of 54 are 2.7 and 8.1, respectively. In sum, the algorithms that obtained the minimum total losses at the same time with the highest number of datasets having *losses* less than 5% are the overall best estimators, since these estimators should be less likely to be outperformed by the other estimators on an unseen dataset.

Table 5 illustrates more detailed performance results of Table 4 by demonstrating the *losses* values per dataset. The results of these two tables clearly show that many estimators performed differently over different datasets; however, it is obvious that, contrary to expectations, the *ABE* models that had their effort adapted by the heuristics-based techniques, i.e., ABE_{rtm} and ABE_{lsa}, were the best overall. ABE_{rtm} and ABE_{lsa} obtained the *losses* values of 9 and 33, respectively. These *losses* are lower than 5% of the maximum achievable *losses* summed over all the datasets, i.e., 5% of 702 is 35. Furthermore, the ABE_{rtm} and the ABE_{lsa} techniques also performed outstandingly for the highest number of datasets, i.e., their *losses* are less than 5% for 11 of 13 datasets and 8 of 13 datasets, respectively. In particular, the ABE_{rtm} technique has not been outperformed by any other technique for more than 15% of the maximum achievable *losses* in any dataset. Clearly, the effort adaptation techniques based on heuristics are significantly better than the techniques based on machine learning algorithms.

From the results of the machine learning-based adaptation techniques, it can be observed that different algorithms show different performance results. That is, the effort estimators ABE_{gbm}, ABE_{ann}, and the ABE_{svr} could offer a better performance than to simply calculate the mean values taken from the analogues of a new project, i.e., better than ABE_{uavg}. However, the estimators such as ABE_{deep}, ABE_{rf}, ABE_{ols} and ABE_{cart} were the otherwise.

Scrutinizing the results obtained from the datasets where the ABE_{rtm} or ABE_{lsa} techniques did not perform outstandingly, such as the *Albrecht* and the *Maxwell* datasets, it was found that, the ABE_{gbm} technique or the ABE_{ann} technique could achieve 0 loss on these datasets. In the literature [39,51], when this performance result happened, it was highly recommended that the potential performance of the stacked ensemble learner generated by combining these algorithms together be explored.

4.2. RQ2 What is the potential performance of the effort adaptors utilizing both heuristic and machine learning algorithm together?

In the literature on software effort estimation related to stacked ensemble, such as [39], when the performance of a combined estimator is explored, typically, the combination scheme used in the experiments was usually limited to simply taking the average of the estimated effort values from many solo estimators to generate the estimated effort [39]. Brown et al. [13], however, based on the results of their mathematical works, suggested that a combined estimator can achieve the theoretical optimal performance if all the solo estimators are combined into one single entity such that the combined estimator can be trained and optimized as one single algorithm in one hyperparameter optimization process. Following this suggestion, in this work, the effort adaptors based on a heuristic and those based on machine learning algorithms are combined by building an ensemble learner that initially generates the estimation effort using a heuristic-based technique, and then the ensemble learner fine tunes this effort value using a machine learning algorithm. Specifically, let the term ABE_{a+b} denote this particular *ABE* model having its effort adapted by a combined adaptor, where a and b indicate a heuristic, such as *rtm*, and machine learning algorithm, such as *gbm*, respectively. For example, an estimator $ABE_{rtm+gbm}$ is constructed in steps as follows:

Step 1. A new software project is set as P_{new}. Then, set all the other existing software projects as the training cases of this P_{new}, i.e., $P_{training}$.

Step 2. Calculate $E(P_{training})$ and $\hat{E}_{ABE_{rtm}}(P_{training})$, which the two terms denote the actual effort and the effort adjusted by the *lsa* technnique of all the training cases, respectively. Note that, $\hat{E}_{ABE_{rtm}}(P_{training})$ is calculated by using Eq. (3).

Table 5
Comparison results classified into 3 groups based on the range of the achieved *losses*: *losses* ∈ [0%, 5%), *losses* ∈ [5%, 15%), and *losses* ∈ [15%, 100%] of the maximum achievable *losses*, where 5% and 15% are 2.7 and 8.1, respectively. Lower losses indicate better performance.

Dataset	*losses* ∈ [0%, 5%)			*losses* ∈ [5%, 15%)			*losses* ∈ [15%, 100%]		
Albrecht	ABE_{gbm}:0	ABE_{svr}:0	ABE_{uavg}:0	ABE_{ann}:3	ABE_{deep}:3	ABE_{lsa}:7			
	ABE_{ols}:0	ABE_{rtm}:1	ABE_{cart}:1	ABE_{rf}:8					
Cocomo81-e	ABE_{ann}:2			ABE_{rtm}:3	ABE_{gbm}:3	ABE_{lsa}:5	ABE_{uavg}:10	ABE_{cart}:19	ABE_{rf}:22
				ABE_{svr}:5	ABE_{deep}:7		ABE_{ols}:29		
Cocomo81-so	ABE_{rtm}:0	ABE_{lsa}:0	ABE_{ann}:0				ABE_{deep}:12	ABE_{rf}:14	ABE_{ols}:31
	ABE_{svr}:0	ABE_{uavg}:0	ABE_{gbm}:2				ABE_{cart}:44		
Cocomo-sdr	ABE_{rtm}:0	ABE_{lsa}:0	ABE_{gbm}:0	ABE_{rf}:7					
	ABE_{ann}:0	ABE_{svr}:0	ABE_{deep}:0						
	ABE_{ols}:0	ABE_{cart}:0	ABE_{uavg}:2						
Desharnais-L12	ABE_{rtm}:0	ABE_{ann}:0	ABE_{deep}:0	ABE_{lsa}:5	ABE_{svr}:5	ABE_{ols}:6	ABE_{cart}:49		
	ABE_{uavg}:1			ABE_{gbm}:6	ABE_{rf}:7				
Desharnais-L3	ABE_{rtm}:0	ABE_{lsa}:0	ABE_{gbm}:0	ABE_{svr}:6			ABE_{uavg}:9	ABE_{cart}:16	ABE_{ols}:27
	ABE_{ann}:0	ABE_{deep}:0	ABE_{rf}:0						
Finnish	ABE_{rtm}:0	ABE_{lsa}:0	ABE_{svr}:1	ABE_{gbm}:3	ABE_{ols}:3		ABE_{rf}:10	ABE_{cart}:15	ABE_{ann}:20
							ABE_{uavg}:20	ABE_{deep}:23	
Kemerer	ABE_{rtm}:0	ABE_{lsa}:0	ABE_{gbm}:0	ABE_{rf}:4			ABE_{cart}:11	ABE_{svr}:13	ABE_{ols}:15
	ABE_{uavg}:0	ABE_{deep}:0	ABE_{ann}:1						
Maxwell	ABE_{rtm}:0	ABE_{ann}:0	ABE_{svr}:0	ABE_{rf}:3			ABE_{gbm}:8	ABE_{lsa}:10	
	ABE_{uavg}:0	ABE_{deep}:0	ABE_{ols}:0						
	ABE_{cart}:0								
Miyazaki94	ABE_{deep}:0	ABE_{rtm}:1	ABE_{lsa}:1	ABE_{svr}:3	ABE_{uavg}:4	ABE_{gbm}:4	ABE_{rf}:9	ABE_{ols}:13	ABE_{cart}:19
				ABE_{ann}:5					
Nasa93-c1	ABE_{rtm}:0	ABE_{lsa}:0	ABE_{gbm}:0	ABE_{ann}:6	ABE_{svr}:7		ABE_{cart}:10	ABE_{ols}:11	ABE_{deep}:15
	ABE_{uavg}:0	ABE_{rf}:0							
Nasa93-c2	ABE_{uavg}:1			ABE_{gbm}:3	ABE_{ann}:3	ABE_{rtm}:4	ABE_{deep}:11	ABE_{ols}:21	ABE_{cart}:31
				ABE_{svr}:4	ABE_{lsa}:5	ABE_{rf}:5			
Nasa93-c5	ABE_{rtm}:0	ABE_{lsa}:0					ABE_{rf}:10	ABE_{ann}:12	ABE_{uavg}:12
							ABE_{gbm}:13	ABE_{cart}:13	ABE_{svr}:13
							ABE_{deep}:12	ABE_{ols}:14	

Step 3. Construct a *gbm* model that transforms the differences in project characteristics to the differences in effort over $P_{training}$. That is, for each project case P_i in $P_{training}$, the model attempts to learn the difference in effort in the form of [44]:

$$\hat{E}(P_i) - \sum_{j=1}^{k} E_{ABE_{rtm}}(P_{analog(i,j)}) = w \sum_{j=1}^{k}(P_i - P_{analog(i,j)}), \qquad (11)$$

where w is the weights which are determined by the training the *gbm* model to minimize the differences in project characteristics and the differences in effort, and k is the number of analogies.

Step 4. Generate $\hat{E}_{ABE_{rtm}}(P_{new})$ by using the ABE_{rtm} technique.

Step 5. Estimate the $\delta(\hat{E}(P_{new}), E(P_{analog(new,1.k)}))$, i.e., the effort difference between $\hat{E}(P_{new})$ and $\hat{E}_{ABE_{rtm}}(P_{new})$ using the *gbm* model constructed in *Step 2*.

Step 6. Adjust $\hat{E}_{ABE_{rtm}}(P_{new})$ by adding $\delta(\hat{E}(P_{new}), E(P_{analog(new,1.k)}))$ to $\hat{E}_{ABE_{rtm}}(P_{new})$. This is the final estimated effort for P_{new}, i.e., $\hat{E}_{abe_{rtm+gbm}}(P_{new})$.

Table 6 demonstrates the performance results of 12 effort estimators based on the same experimental setup as that of RQ1. Of these, six effort adaptors are based on ensemble learners, i.e., ABE_{a+b}, which are generated from a heuristic $a \in \{lsa, rtm\}$, and an algorithm $b \in B \mid B$, which is the set of algorithms performing better than the baseline ABE_{uavg} technique in the experiment of the RQ1. The other six effort estimators are the ABE_{rtm}, the ABE_{lsa}, the ABE_{uavg}, and the other three techniques that perform better than the ABE_{uavg} in Table 4.

In this experiment, the maximum *losses* per dataset and over all the datasets are 66 and 838, respectively. Subsequently, the 5% and 15% of the maximum achievable *losses* per dataset are 3.3 and 9.9, respectively. As shown in Table 6 the most noteworthy result is that the $ABE_{rtm+gbm}$ technique generated the best performance overall both in terms of the total *losses* it achieved, and in the total number of datasets it performed outstandingly. Precisely, $ABE_{rtm+gbm}$ has achieved the total *losses* of three, and it performed highly, i.e., *losses* were less than 5% for 13 of 13 experimental datasets. The $ABE_{rtm+ann}$ technique ranked second,

Table 6
Comparison results of the combined effort adaptors and the adaptors that performed at least as highly as the ABE_{uavg} technique in the experiment of RQ1.

Rank	Algorithm	Total losses	#Datasets having *losses* ∈		
			[0%,5%)	[5%,15%)	[15%, 100%]
1	$ABE_{rtm+gbm}$	3	13/13	0/13	0/13
2	$ABE_{rtm+ann}$	20	12/13	1/13	0/13
3	ABE_{rtm}	30	10/13	2/13	1/13
4	$ABE_{rtm+svr}$	37	11/13	1/13	1/13
5	$ABE_{lsa+gbm}$	38	10/13	2/13	1/13
6	ABE_{lsa}	42	9/13	2/13	2/13
7	$ABE_{lsa+ann}$	42	9/13	3/13	1/13
8	$ABE_{lsa+svr}$	45	8/13	3/13	2/13
9	ABE_{gbm}	100	5/13	6/13	2/13
10	ABE_{ann}	131	5/13	5/13	3/13
11	ABE_{svr}	142	5/13	3/13	5/13
12	ABE_{uavg}	147	8/13	0/13	5/13

followed by the ABE_{rtm} and the $ABE_{rtm+svr}$ techniques. In other words, the effort adaptation techniques which are the extension of theABE_{rtm} technique and the ABE_{rtm} technique itself, are the highest performers of this experiment. Further, the ABE_{rtm} technique, the ABE_{lsa} technique, and all the combined techniques that extended the two techniques performed better the other techniques based on solo machine learning algorithms, as well as the baseline ABE_{uavg} technique. Therefore, the $ABE_{rtm+gbm}$ technique is the overall best *ABE* variant, and it is one of the best variants of the *ABE* model to the best of the author's knowledge.

4.3. RQ3 How does the performance of the best ABE model discovered in the present study compare to other state-of-the-art software effort estimation techniques?

Since the extraordinary performance of the $ABE_{rtm+gbm}$ technique was discovered, the author has been interested in knowing how the performance of this estimator compares to other estimators in directly es-

Table 7
Comparison results of the $ABE_{rtm+gbm}$ technique over *lp4ee*, i.e., the current standard benchmark software effort estimator, and other eight model-based software effort estimators studied in [55]. The performance is indicated by the total possible *losses*, where lower *losses* and the higher number of datasets achieving a low number of *losses* indicate higher performance.

Rank	Algorithm	Total losses	#Datasets having *losses* ∈		
			[0%,5%)	[5%,15%)	[15%, 100%]
1	$ABE_{rtm+gbm}$	4	13/13	0/13	0/13
2	*lp4ee*	42	9/13	2/13	2/13
3	*rf*	48	8/13	3/13	2/13
4	*gbm*	57	8/13	1/13	3/13
5	*svr*	61	7/13	1/13	5/13
6	ABE_{uavg}	93	6/13	3/13	4/13
7	*ols*	136	4/13	1/13	8/13
8	*cart*	159	2/13	1/13	10/13
9	*ann*	195	3/13	0/13	10/13
10	*deep*	196	2/13	0/13	11/13

timating the software effort by using machine learning algorithms, and to the current standard benchmark software effort estimator, which is a method named *lp4ee* proposed by [57]. In [55], *rf* was the best solo machine learning algorithm in the study, and in [72], *lp4ee* outperformed various machine learning-based effort estimators, including *rf, svr*, and *cart*. Table 7 shows the performance results of ten effort estimators using the same approach that generated the results presented in Tables 4 and 6. These ten estimators are $ABE_{rtm+gbm}$, *lp4ee*, and the other eight machine learning algorithms examined in the experiment by Phannachitta and Matsumoto [55]. Note that there were six additional algorithms; namely, *bagging, adaBoost, lasso, ridge*, and *lars*, which were examined in [55] but they are not all included in the present study. The main reason for excluding these six algorithms is because [55] were able to conclude that the algorithms based on similar theoretical perspectives will perform similarly, such that *rf* was the highest performed algorithm in the same group as *bagging*. Thus, its performance already represents the performance of *bagging* without the need to examine both algorithms in every experiment.

In this experiment, the maximum achievable *losses* per dataset and that of all the datasets overall are 48 and 624, respectively. Subsequently, 5% and 15% of maximum *losses* per dataset are 2.7 and 8.1, respectively. From the results of Table 7, it is clear that $ABE_{rtm+gbm}$ is the best effort estimator. Its performance result was significantly better than that of the *lp4ee* technique and all the other estimators benchmarked in [55], where none of the estimators in the study had their *losses* ∈ [0%, 5%) as outstanding as the $ABE_{rtm+gbm}$ technique, i.e., 13 of 13 datasets. To better see the performance spectrum as to conclude how outstanding the $ABE_{rtm+gbm}$ technique is, Fig. 1 shows the raw performance spectrums of the top four estimators of Table 7 based on all six performance metrics. The six plots in this figure sort all the metrics over 13 datasets in ascending order. For *mae, rsd, lsd, mbre*, and *mmer*, the lower values indicate better performance, whereas higher *sa* values indicate better performance. The six estimators ranked on the bottom of Table 7 were excluded from this plot because they had extremely poor performance for at least one metric, making it difficult to read corresponding plots if they are presented. In Fig. 1, the performance spectrums of the $ABE_{rtm+gbm}$ technique were shown to be better than that of the other three techniques for the *mbre*, the *mmer*, and the *sa* metrics. For the *lsd* and the *rsd* metrics, $ABE_{rtm+gbm}$ performed as highly as the second best techniques, which are *gbm* and *lp4ee*, respectively. For the *mae* metric, there appeared to be no clear difference in performance between the four techniques. Corroborating with the results of Table 6, this $ABE_{rtm+gbm}$ technique has achieved a new state-of-the-art result, such that it achieved the lowest number of times it was outperformed by the current standard benchmark technique, the other variants of the *ABE* models, and the other model-based software effort estimation techniques. Also, its *losses* have never exceeded 5% of the maximum *losses* in any comparison.

5. Discussion

5.1. The significance of an effort estimator like $ABE_{rtm+gbm}$

Until the present study, *ABE* models with effort adapted by heuristics have been known as the best variants of the models. In the author's view, it can be difficult for practitioners to have full confidence in an estimator if it relies heavily on a heuristic, since it is difficult to determine how likely the heuristic will perform in an unseen situation. In 2008, there was a strong debate on how likely any variant of the *ABE* models will perform on an unseen dataset. Subsequently, a method named *Analogy-X* [33] was invented and soon become the standard method to test whether or not *ABE* was suitable for a given dataset. *Analogy-X* is based on Mantel's correlation re-sampling test, which is able to quantify an index representing whether software projects with similar characteristics in a given dataset were completed with a similar effort or not. Tested on a dataset, if the index is lower than a particular threshold value, it is likely that *ABE* may not be a suitable estimator because it is likely not to perform at all. In the author's opinion, the discovery of high-performance effort adaptors that use both a machine-learning algorithm and heuristic together, such as the $ABE_{rtm+gbm}$ technique, can be an important step in developing an effort estimator that will perform in most, if not all, circumstances.

As shown by the outstanding results, these effort adaptors can adapt to the effort for the projects whose characteristics are not similar to any other projects. In other words, this improves the theoretical basis of *abe* by allowing it to offer an accurate estimation even for the datasets which may fail the test by *Analogy-X*.

5.2. Why $ABE_{rtm+gbm}$ performs best

5.2.1. Solo estimators are combined with effective approaches

To build a successful ensemble learner from multiple solo estimators, [12] have suggested that it is important to consider both the maximum potential performance and the level of diversity of all the estimators to be combined. Specifically, given a set of estimators, the maximum potential performance can be achieved with the set of estimators that performed with the highest overall accuracy. At the same time, all the estimators must also perform highly on the largest number of datasets. For the level of diversity, Brown and Kuncheva [12] suggested selecting the estimators that can complement each other under specific datasets. By applying these suggestions to the present study, when the performance result of Table 4 was generated, the author could see the potential performance gained by combining the ABE_{rtm} and the ABE_{gbm} techniques into one single estimator, as they are not only the high performers in the experiment, but the maximum level of diversity between the two techniques, which can be also be implied since they are based on different theoretical perspectives, i.e., heuristics and a machine learning algorithm, respectively.

To the best of the author's knowledge, the $ABE_{rtm+gbm}$ technique has offered the most outstanding performance of all the model-based techniques the author has investigated. As demonstrated by the results in Tables 6 and 7, and Fig. 1, the technique has outperformed all the estimators including the current standard benchmark technique, i.e., *lp4ee*, and current best model-based effort estimator, i.e., the estimator based on the *rf* algorithm [55]. The performance results of the $ABE_{rtm+gbm}$ techniques are so exceptional that the total *losses* achieved in Tables 6 and 7 are both less than 1% of the maximum possible *losses* which could be obtained in the two corresponding experiments. Furthermore, demonstrated by the results per dataset in all the result tables, the $ABE_{rtm+gbm}$ technique can be classified as the high performer for all the experimental datasets of the present study.

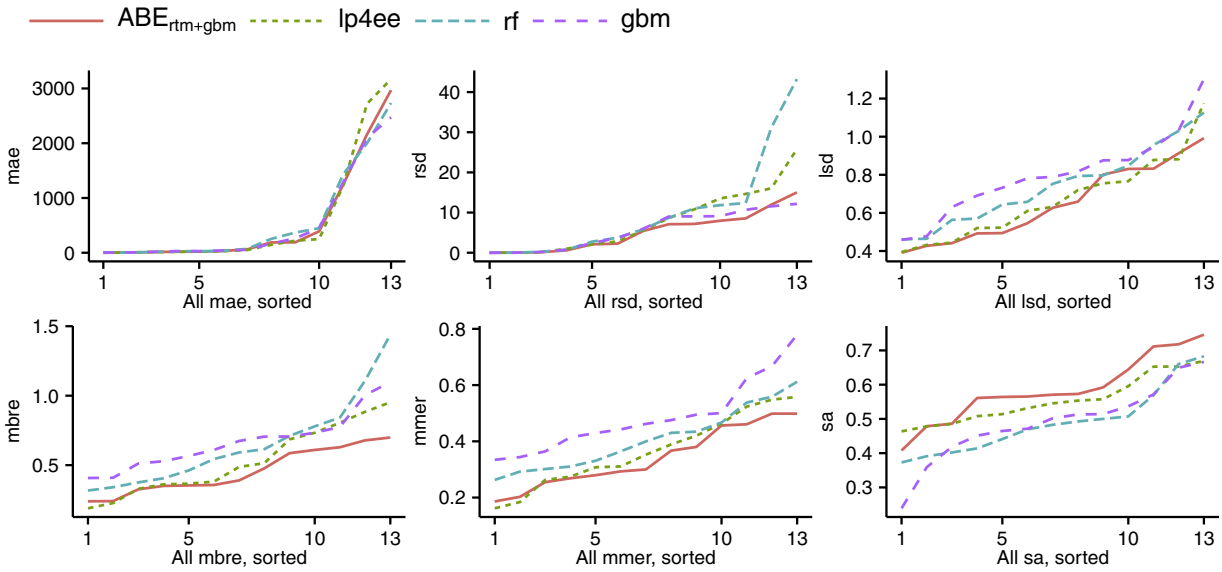

Fig. 1. The spectrums of the six performance metrics comparing $ABE_{rtm+gbm}$, *lp4ee, rf*, and *gbm* techniques. Lower *mae, rsd, lsd, mbre*, and *mmer* indicate better performance. Higher *sa* indicates better performance.

5.2.2. High performance is achieved at the same time as high explicability

The performance of the $ABE_{rtm+gbm}$ technique can be attributed to the three essential components of the technique, which are: *ABE, rtm*, and *gbm*. *ABE*'s performance has been widely recognized since its first proposal over two decades ago. Even if *ABE* was commonly implemented in the form of ABE_{uavg}, whose performance is not even close to the optimal performance of $ABE_{rtm+gbm}$, the ABE_{uavg} technique was widely seen on the top ranks in many studies. Most interesting about *ABE* is that, when explaining the *ABE* model in an algorithmic way to a data scientist, she or he will likely be surprised that *ABE* can be viewed as a simple k-nearest neighbors algorithm, which does not often offer a desirable performance [70]. Thus, the author believes that what makes *ABE* standout as a software effort estimator is that, software project managers believe and follow *ABE* to lead the software project to be completed within the similar effort, which was used in the past similar projects. For example, based on interviews with multiple local software companies in the northern-Thailand software industry hub, it was found that most of these companies estimate the software effort by dividing the task and letting each developer in charge estimate the required effort for the particular components in terms of man-hours. Typically, the developer will consult past records on how long it took to complete a similar task to generate the arbitrary estimated effort values. After the estimation for the subtasks are completed, the project manager will aggregate the effort to generate the final estimation for the entire project.

For *rtm* and *gbm*, not only is the principle of ensemble attributed to the performance gained from combining the two techniques, but the combination of the two techniques are also intuitively explicable. That is, from Eq. (3), effort adaptation by development productivity can be viewed as an extension of effort adaptation by software size, which is what has been widely used in generating an effort value by an individual developer before attempting to use any other criteria. However, software projects of the same size may sometimes not acquire similar amount of effort for their completion. Differences in productivity is one of the attributes which can intuitively explain the phenomenon. When information related to the changes in development productivity is available, a project manager will usually calibrate her or his final effort estimation using it. Moreover, there are the cases when the developers may feel that there is a gap between the adjusted productivity and what should actually be required as the difference in other aspects. In such cases, human intuition will be extensively used by making many assumptions to translate the differences in characteristics into the difference in the effort in the most explicable way possible. Based on these assumptions, the process of the $ABE_{rtm+gbm}$ is able to fully mimic how the effort is generated in the most similar way an experienced software developer or project manager would generate it. In the author's view, the ability to offer intuitive explanation is as critical as that of accuracy, because, in the past, the reasons that the developers opted not to undertake a model-based estimation, despite knowing its expectation performance, were mainly because the model-based estimators were usually excessively difficult to understand how a particular effort value is generated based on them.

In sum, the author believes that the variants of an *ABE* model constructed with the combined effort adaptors that exercises both development productivity and a powerful machine learning algorithm, such as $ABE_{rtm+gbm}$, are the most accurate and explicable variants of the *ABE* model up to the present.

5.3. Threats to validity

Internal validity questions whether different conclusion can be drawn with regards to different parameter setups in the study. The choice of feature selection methods, data normalization techniques, and the value assignment for all the experimental parameters are the common threats to internal validity in the literature on model-based software effort estimation. For example, in the study related to *ABE*, the choice of the *k* parameter value is one of the most encountered sources of the validity threat discussed in these studies. For the present study, it is however the author's belief that all of these validity threats were already handled by the use of hyperparameter optimization via the Bayesian optimization search technique. That is, the hyperparameter optimization process has attempted to search for the optimum set of feature selection methods, data normalization techniques, the *k* parameter, and all the other hyperparameters associated with different machine learning algorithms used in a particular experiment. The optimum parameter set is believed to be the most suitable setup for every test project case in the experiments. However, recent work that also utilizes the optimization has noted that the use of the optimization process itself can be considered as the other internal validity threat [72]. That is, due to the large search space involved, the optimization is usually undertaken in a smaller search space where randomization has to be accounted for. To minimize the threats to this internal validity, the author run ten ran-

domized trials on every experiment where the Bayesian hyperparameter optimization was undertaken.

For external validity, which is concerned with the generalization ability of the performance results, the main threat is that the present study has only evaluated the performance results of a search-based approach when applied for hyperparameter optimization, but not for making predictions. An effort estimator that directly makes predictions using a search-based technique was suggested by Sarro et al. [58] as a more sophisticated approach to estimate the software effort than those which are based on traditional machine learning algorithms. Since an improvement to the traditional *ABE* technique proposed in this study could significantly outperform many state-of-the-art approaches based on machine-learning algorithms, as future work, the author plans to systematically compare the proposed technique with other state-of-the-art search-based effort estimation techniques [18,56,58]. Other validity issues to mention are the selection of independent datasets and the selection of the independent performance metrics. For the datasets, the kernel density estimations were used to ensure that there is no overlap distribution for the 13 selected datasets. Similarly, for the performance metrics, the redundant test was carried out to ensure that there was no redundant metrics that may bias the performance conclusion.

For construct validity, which is concerned with whether or not what was measured throughout this study is really what should be measured. Brunner's test and the refinement of the Stable ranking identification method in its component that summarizes the performance results are the strategies that the author believes can minimize the possible threats to the construct validity. Specifically, the ability to reduce the effect of the tie ranks in both a non-parametric statistical test as well as in the performance comparison is why the author believes the results and the findings of this study are of the highest reliability.

6. Conclusion

This study discovered an effective solution to construct an exceptionally accurate, reliable, and explicable model-based software effort estimator. The proposed approach suggests constructing a combined estimator that initially generates the widely-accepted *ABE* model as the base estimator. Next, the estimation product was scaled by consulting the productivity differences between the new project and past completed projects. In the final step, the estimated effort was further fine-tuned using an advanced machine learning algorithm, *gbm*, which is an increasingly popular algorithm used in many winning solutions of recent data science competitions. In these three steps, hyperparameter optimization was required to ensure that all the experimental configurations have been made optimum in every attempt to make an estimation.

An outstanding variant of *ABE* model coined $ABE_{rtm+gbm}$ was discovered. This variant is a combination of ABE_{rtm} and ABE_{gbm} techniques at the algorithmic level. Not only was its outstanding performance demonstrated, but its intuitive ability also appeared to be outstanding. Based on interviews with software developers and project managers in the northern-Thailand software industry hub, the process that generates the effort based on this $ABE_{rtm+gbm}$ technique was found to fully explained in a similar way which an experienced software developer or project manager would generate. Therefore, this highly accurate and intuitive model, with its automatable effort estimation, does not need any expert interactions.

For future research, the author intends to evaluate the performance of the $ABE_{rtm+gbm}$ technique against other state-of-the-art effort estimation approaches, such as search-based techniques. A better understanding of various sophisticated approaches may foster the discovery of a more accurate effort estimator in a similar way to that in which the $ABE_{rtm+gbm}$ technique was discovered in the present study by combining *ABE* and the current state-of-the-art machine-learning approach. In another direction, a better understanding of how to quantify the actual explicable performance of an ensemble learner is the author's interest. This can potentially further improve the effort estimation model generated on the proposed solution. An effective ensemble learner requires multiple solo estimators with maximum achievable accuracy for each estimator as well as a level of diversity for these estimators. However, what was carried out in this study assumes the level of two properties, such that *rtm* and *gbm* were the highest performers, while at the same time their principle is based on different theoretical principles. The author believes that if the two properties can be numerically quantified, it will allow a precise determination of whether or not a particular combined technique will perform without the need to implement it. This will be highly useful in reducing the operation cost in finding an accurate and reliable model in practice.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Passakorn Phannachitta: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing - original draft, Writing - review & editing.

Acknowledgements

The author sincerely thanks the Amazon Web Services (AWS) Cloud Credits for Research Program, which provided the author with the opportunity to access high-performance computing resources through Amazon's EC2 cloud computing service.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.infsof.2020.106330.

References

[1] I.M. Abril, M. Sugiyama, Winning the kaggle algorithmic trading challenge with the composition of many models and feature engineering, IEICE T. Inf. Syst. 96 (3) (2013) 742–745.
[2] A.J. Albrecht, J.E. Gaffney, Software function, source lines of code, and development effort prediction: a software science validation, IEEE Trans. Softw. Eng. 9 (6) (1983) 639–648.
[3] E. Alpaydin, Introduction to Machine Learning, MIT press, 2014.
[4] Amazon, Amazon web services, 2019,. Available in: http://aws.amazon.com/.
[5] M. Azzeh, Model tree based adaption strategy for software effort estimation by analogy, in: Proceedings of the 11th International Conference on Computer and Information Technology, IEEE, 2011, pp. 328–335.
[6] M. Azzeh, A replicated assessment and comparison of adaptation techniques for analogy-based effort estimation, Empiric. Softw. Eng. 17 (1–2) (2012) 90–127.
[7] A. Bakır, B. Turhan, A.B. Bener, A new perspective on data homogeneity in software cost estimation: a study in the embedded systems domain, Softw. Qual. J. 18 (1) (2010) 57–80.
[8] K.E. Bennin, J. Keung, P. Phannachitta, A. Monden, S. Mensah, Mahakil: diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction, IEEE Trans. Softw. Eng. (2017).
[9] J. Bergstra, D. Yamins, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: Proceedings of the 30th International Conference on Machine Learning, 2013, pp. 115–123.
[10] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[11] L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, Classification and regression trees, CRC press, 1984.
[12] G. Brown, L.I. Kuncheva, Good and bad diversity in majority vote ensembles, in: Proceedings of the International Workshop on Multiple Classifier Systems, Springer, 2010, pp. 124–133.
[13] G. Brown, J. Wyatt, R. Harris, X. Yao, Diversity creation methods: a survey and categorisation, Inf. Fusion 6 (1) (2005) 5–20.
[14] E. Brunner, U. Munzel, M.L. Puri, The multivariate nonparametric behrens–fisher problem, J. Stat. Plan. Inf. 108 (1) (2002) 37–53.
[15] U. Çayır, I. Yenidoğan, H. Dağ, Use case study: data science application for microsoft malware prediction competition on kaggle, Proc. Int. Conf. Data Sci. Mach. Learn. Stat. (2019) 98–100.

[16] T. Chen, C. Guestrin, Xgboost: a scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 785–794.
[17] F. Chollet, Deep Learning with Python, Manning Publications Co., 2017.
[18] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, E. Mendes, Using tabu search to configure support vector regression for effort estimation, Empiric. Softw. Eng. 18 (3) (2013) 506–546.
[19] H. Drucker, C.J. Burges, L. Kaufman, A.J. Smola, V. Vapnik, Support vector regression machines, in: Proceedings of Neural Information Processing Systems Conference, 1997, pp. 155–161.
[20] T. Foss, E. Stensrud, B. Kitchenham, I. Myrtveit, A simulation study of the model evaluation criterion mmre, IEEE Trans. Softw. Eng. 29 (11) (2003) 985–995.
[21] J.H. Friedman, Greedy function approximation: a gradient boosting machine, Ann. Stat. (2001) 1189–1232.
[22] M.K.S. Ganesh, K. Thanushkodi, An efficient software cost estimation technique using fuzzy logic with the aid of optimization algorithm, Int. J. Innov. Comput. I. 11 (2) (2015) 587–597.
[23] A. Goldbloom, Data prediction competitions – far more than just a bit of fun, in: Proceedings of the IEEE International Conference on Data Mining Workshops, IEEE, 2010, pp. 1385–1386.
[24] HyperOpt, Hyperopt: Distributed hyperparameter optimization, 2019,. Available in: https://github.com/hyperopt/hyperopt.
[25] A.K. Jain, J. Mao, K.M. Mohiuddin, Artificial neural networks: a tutorial, Computer 29 (3) (1996) 31–44.
[26] M. Jørgensen, U. Indahl, D. Sjøberg, Software effort estimation by analogy and "regression toward the mean", J. Syst. Softw. 68 (3) (2003) 253–262.
[27] M. Jorgensen, M. Shepperd, A systematic review of software development cost estimation studies, IEEE Trans. Softw. Eng. 33 (1) (2007) 33–53.
[28] Kaggle, Kaggle: your home for data science, 2019,. Available in: https://www.kaggle.com,.
[29] C.F. Kemerer, An empirical validation of software cost estimation models, Commun. ACM 30 (5) (1987) 416–429.
[30] Keras, Keras: The python deep learning library, 2019,. Available in: https://keras.io/.
[31] J. Keung, Software development cost estimation using analogy: a review, in: Proceedings of the 2009 Australian Software Engineering Conference, 2009, pp. 327–336.
[32] J. Keung, E. Kocaguneli, T. Menzies, Finding conclusion stability for selecting the best effort predictor in software effort estimation, Automated. Softw. Eng. 20 (4) (2013) 543–567.
[33] J.W. Keung, B.A. Kitchenham, D.R. Jeffery, Analogy-x: providing statistical inference to analogy-based software cost estimation, IEEE Trans. Softw. Eng. 34 (4) (2008) 471–484.
[34] Y. Kim, J. Garvin, M.K. Goldstein, S.M. Meystre, Classification of contextual use of left ventricular ejection fraction assessments, Stud. Health Technol. Inform. 216 (2015) 599.
[35] C. Kirsopp, E. Mendes, R. Premraj, M. Shepperd, An empirical analysis of linear adaptation techniques for case-based prediction, in: Proceedings of the 5th International Conference on Case-Based Reasoning: Research and Development, 2003, pp. 231–245.
[36] B. Kitchenham, K. Känsälä, Inter-item correlations among function points, in: Proceedings of the 15th International Conference on Software Engineering, IEEE, 1993, pp. 477–480.
[37] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, A. Pohthong, Robust statistical methods for empirical software engineering, Empirical Softw. Eng. 22 (2) (2017) 579–630.
[38] E. Kocaguneli, T. Menzies, Software effort models should be assessed via leave-one-out validation, J. Syst. Softw. 86 (7) (2013) 1879–1890.
[39] E. Kocaguneli, T. Menzies, J. Keung, On the value of ensemble effort estimation, IEEE Trans. Softw. Eng. 38 (6) (2012) 1403–1416.
[40] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, R. Madachy, Active learning and effort estimation: finding the essential content of software effort estimation data, IEEE Trans. Softw. Eng. 39 (8) (2013) 1040–1053.
[41] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of 14th International Joint Conference on Artificial Intelligence, 1995, pp. 1137–1143.
[42] W.B. Langdon, J. Dolado, F. Sarro, M. Harman, Exact mean absolute error of baseline predictor, marp0, Inf. Softw. Technol. 73 (2016) 16–18.
[43] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[44] Y.F. Li, M. Xie, T.N. Goh, A study of the non-linear adjustment for analogy based software cost estimation, Empiric. Softw. Eng. 14 (6) (2009) 603–643.
[45] C. Mair, M. Shepperd, The consistency of empirical comparisons of regression and analogy-based software project cost prediction, in: Proceedings of the 2005 International Symposium on Empirical Software Engineering, 2005, pp. 509–518.
[46] K. Maxwell, Applied statistics for software managers, Englewood Cliffs, NJ. Prentice-Hall, 2002.
[47] E. Mendes, N. Mosley, S. Counsell, A replicated assessment of the use of adaptation rules to improve web cost estimation, in: Proceedings of the 2003 International Symposium on Empirical Software Engineering, 2003, pp. 100–109.
[48] T. Menzies, R. Krishna, D. Pryor, The promise repository of empirical software engineering data, 2015,.
[49] T. Menzies, R. Krishna, D. Pryor, The seacraft repository of empirical software engineering data, 2017, (https://zenodo.org/communities/seacraft).
[50] T. Menzies, D. Port, Z. Chen, J. Hihn, S. Stukes, Validation methods for calibrating software effort models, in: Proceedings of the 27th International Conference on Software Engineering, ACM, 2005, pp. 587–595.
[51] L.L. Minku, X. Yao, Ensembles and locality: insight on improving software effort estimation, Inf. Softw. Technol. 55 (8) (2013) 1512–1528.
[52] Y. Miyazaki, M. Terakado, K. Ozaki, H. Nozaki, Robust regression for developing software estimation models, J. Syst. Softw. 27 (1) (1994) 3–16.
[53] P. Phannachitta, Robust comparison of similarity measures in analogy based software effort estimation, in: Proceedings of the 11th International Conference on Software, Knowledge, Information Management and Applications, 2017, pp. 1–7.
[54] P. Phannachitta, J. Keung, A. Monden, K. Matsumoto, A stability assessment of solution adaptation techniques for analogy-based software effort estimation, Empiric. Softw. Eng. 22 (1) (2017) 474–504.
[55] P. Phannachitta, K. Matsumoto, Model-based software effort estimation - a robust comparison of 14 algorithms widely used in the data science community, Int. J. Innov. Comput. I. 15 (2) (2019) 569–589.
[56] F. Sarro, Search-based predictive modelling for software engineering: How far have we gone? in: Proceedings of the International Symposium on Search Based Software Engineering, 2019, pp. 3–7.
[57] F. Sarro, A. Petrozziello, Linear programming as a baseline for software effort estimation, ACM Trans. Softw. Eng. Methodol. 27 (3) (2018) 1–28.
[58] F. Sarro, A. Petrozziello, M. Harman, Multi-objective software effort estimation, in: Proceedings of 38th International Conference on Software Engineering, ACM, 2016, pp. 619–630.
[59] Scikit-learn, Scikit-learn: machine learning in python, 2019,. Available in: http://scikit-learn.org/.
[60] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N. De Freitas, Taking the human out of the loop: a review of bayesian optimization, Proc. IEEE 104 (1) (2015) 148–175.
[61] M. Shepperd, M. Cartwright, A replication of the use of regression towards the mean (r2m) as an adjustment to effort estimation models, in: Proceedings of the 11th IEEE International Software Metrics Symposium, 2005, pp. 38–47.
[62] M. Shepperd, G. Kadoda, Comparing software prediction techniques using simulation, IEEE Trans. Softw. Eng. 27 (11) (2001) 1014–1022.
[63] M. Shepperd, S. MacDonell, Evaluating prediction systems in software project estimation, Inf. Softw. Technol. 54 (8) (2012) 820–827.
[64] M. Shepperd, C. Schofield, Estimating software project effort using analogies, IEEE Trans. Softw. Eng. 23 (11) (1997) 736–743.
[65] K. Srinivasan, D. Fisher, Machine learning approaches to estimating software development effort, IEEE Trans. Softw. Eng. 21 (2) (1995) 126–137.
[66] D. Tkaczyk, A. Collins, P. Sheridan, J. Beel, Machine learning vs. rules and out-of-the-box vs. retrained: an evaluation of open-source bibliographic reference and citation parsers, in: Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries, ACM, 2018, pp. 99–108.
[67] T. Unterthiner, A. Mayr, G. Klambauer, M. Steijaert, J.K. Wegner, H. Ceulemans, S. Hochreiter, Multi-task deep networks for drug target prediction, in: Proceedings of Neural Information Processing Systems Conference, volume 2014, 2014, pp. 1–4.
[68] V. Vo, J. Lua, B. Vo, A turning points method for stream time series prediction, Int. J. Innov. Comput. I. 9 (10) (2013) 3965–3980.
[69] F. Walkerden, R. Jeffery, An empirical study of analogy-based software effort estimation, Empiric. Softw. Eng. 4 (2) (1999) 135–158.
[70] J. Wei, X. Qi, M. Wang, Collaborative representation classifier based on k nearest neighbors for classification, J. Softw. Eng. 9 (1) (2015) 96–104.
[71] R. Wilcox, Modern Statistics for the Social and Behavioral Sciences: A Practical Introduction, CRC press, 2011.
[72] T. Xia, R. Krishna, J. Chen, G. Mathew, X. Shen, T. Menzies, Hyperparameter optimization for effort estimation, in: Preprint to be appeared in Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering, 2018.
[73] Y. Xia, C. Liu, Y. Li, N. Liu, A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring, Expert Syst. Appl. 78 (2017) 225–241.