Evaluating Hyper-parameter Tuning using Random Search in Support Vector Machines for Software Effort Estimation

Leonardo Villalobos-Arias
leonardo.villalobosarias@ucr.ac.cr
Universidad de Costa Rica
San Pedro, Costa Rica

Christian Quesada-López
cristian.quesadalopez@ucr.ac.cr
Universidad de Costa Rica
San Pedro, Costa Rica

Jose Guevara-Coto
jose.guevaracoto@ucr.ac.cr
Universidad de Costa Rica
San Pedro, Costa Rica

Alexandra Martínez
alexandra.martinez@ucr.ac.cr
Universidad de Costa Rica
San Pedro, Costa Rica

Marcelo Jenkins
marcelo.jenkins@ucr.ac.cr
Universidad de Costa Rica
San Pedro, Costa Rica

ABSTRACT

Studies in software effort estimation (SEE) have explored the use of hyper-parameter tuning for machine learning algorithms (MLA) to improve the accuracy of effort estimates. In other contexts random search (RS) has shown similar results to grid search, while being less computationally-expensive. In this paper, we investigate to what extent the random search hyper-parameter tuning approach affects the accuracy and stability of support vector regression (SVR) in SEE. Results were compared to those obtained from ridge regression models and grid search-tuned models. A case study with four data sets extracted from the ISBSG 2018 repository shows that random search exhibits similar performance to grid search, rendering it an attractive alternative technique for hyper-parameter tuning. RS-tuned SVR achieved an increase of 0.227 standardized accuracy (*SA*) with respect to default hyper-parameters. In addition, random search improved prediction stability of SVR models to a minimum ratio of 0.840. The analysis showed that RS-tuned SVR attained performance equivalent to GS-tuned SVR. Future work includes extending this research to cover other hyper-parameter tuning approaches and machine learning algorithms, as well as using additional data sets.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**; • **Computing methodologies → Machine learning**; *Supervised learning*.

KEYWORDS

Software effort estimation, empirical study, support vector machines, hyper-parameter tuning, random search, grid search

ACM Reference Format:
Leonardo Villalobos-Arias, Christian Quesada-López, Jose Guevara-Coto, Alexandra Martínez, and Marcelo Jenkins. 2020. Evaluating Hyper-parameter Tuning using Random Search in Support Vector Machines for Software Effort Estimation. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '20), November 8–9, 2020, Virtual, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3416508.3417121

1 INTRODUCTION

The process of estimating the effort required to develop a software product is known as software effort estimation (SEE). Among these is support vector regression (SVR), a machine learning algorithm which has been successfully used for effort estimation of cross-company (CC) data sets [7, 8]. The effectiveness of SVR can be attributed to its ability to adapt to different and heterogeneous chunks of data. Hyper-parameter settings allow SVR to better represent the characteristics of the data, but inappropriate selection of values can result in model over- or under-fitting. This could lead to potentially worse prediction accuracy than default values [19].

Hyper-parameters are sets of values defined before the SEE model's construction, and can affect model performance [21, 30]. Appropriate hyper-parameter values can increase the prediction accuracy of the model, when compared to the default values [35]. To avoid manually tuning hyper-parameters, research in SEE has studied automated hyper-parameter tuning approaches [21]. A common characteristic among these studies is the frequent use of grid search, a method that exhaustively explores the hyper-parameter value space to identify the best set of values [4]. Yet, grid search is computationally costly and suffers from the so-called curse of dimensionality [3]. Random search, on the other hand, is a hyper-parameter tuning approach that takes and evaluates samples from a search space. Existing evidence suggests that random search can provide model performance similar to that of grid search, while being less computationally costly [3]. Previous studies have evaluated hyper-parameter tuning approaches in other areas of software engineering [1, 13, 33, 34]. We attempt to extend this work in the area of software effort estimation, initially with the random search tuning approach.

In this study, we evaluate the effect of the random search hyper-parameter tuning approach applied to SVR for software effort estimation. To this end, we performed an experimental study that compares the prediction accuracy and stability of an SVR model tuned with random search to an SVR model built with default hyper-parameters. Four subsets obtained from the ISBSG 2018 Release 1 data set are used. Additionally, ridge regression models and grid search-tuned models are used as evaluation benchmarks.

2 RELATED WORK

Several studies have investigated the effect of hyper-parameter tuning for SEE models. Song *et al.* [30] performed a study to assess the impact of hyper-parameter tuning in different machine learning algorithms (MLA). The authors trained and evaluated four different MLA using all possible combinations of hyper-parameter values defined across a range, akin to grid search. The study compared the best, the worst, and the default settings for hyper-parameter values.

Dejaeger *et al.* [9] performed a benchmarking study on the performance of multiple effort estimation techniques. They employed nine different data sets and two feature selection methods to train 13 different machine learning models. Besides, they used grid search to select appropriate hyper-parameter values for each technique lacking a recommended value in the literature.

Minku [23] proposed a novel technique for online effort estimation using data clustering and hyper-parameter tuning techniques to provide better estimates and minimize the need for collecting within-company data. The study compared their technique to the untuned version of the model, and other baselines for online effort estimation. The results showed that their tuning approach increased the accuracy with respect to the untuned model.

Xia *et al.* [37] present a hyper-parameter tuning architecture called OIL for SEE. OIL is used to construct and evaluate the combinations of 3 optimizers and 2 learners against 4 'off-the-shelf' methods. The study concludes that off-the-shelf methods and default parameters should be deprecated, and instead recommends the usage of simple, automatic, effective and fast optimization methods in conjunction with learners for SEE. Corazza *et al.* [8] applied the Tabu Search technique as an automated hyper-parameter tuning approach for support vector regression in the context of SEE.

Oliviera *et al.* [24] used Genetic Algorithms (GA) for simultaneous feature selection and hyper-parameter tuning applied to machine learning algorithms for SEE.

Similar studies have been performed in other software engineering research areas. Tantithamthavorn *et al.* [33] applied a hyper-parameter tuning approach (Caret optimization) to improve the performance of defect prediction models. An extension study by Tantithamthavorn *et al.* [34] further assess the interpretability of models, transferability of parameters, and computational cost of hyper-parameter tuning. Fu *et al.* [13] evaluate three defect predictors using differential evolution tuning in 9 data sets. The results of this study show that tuning defect predictors can be simple and can improve their performance, to the point of changing which learners are the 'best'. Similarly, Agrawal et al. [1] propose a novel tuning approach called DODGE(ϵ) that avoids redundant hyper-parameter settings, runs orders of magnitude faster, and generates more accurate models. Our future work will encompass evaluation of some tuners proposed in these papers.

3 STUDY DESIGN

The main objective of this study is to investigate to what extent the random search hyper-parameter tuning approach affects the accuracy and stability of support vector regression for software effort estimation. We compared the results against those obtained from ridge regression models and grid search-tuned models. The following research questions were posed:

RQ1 What is the improvement in prediction accuracy of support vector regression when random search is used?
RQ2 How stable is the prediction accuracy of support vector regression when random search is used?
RQ3 Which of the evaluated models yields the best accuracy for SEE?

3.1 Data set

The analyses presented in this paper are based on the International Software Benchmarking Standards Group (ISBSG) Repository (https://www.isbsg.org/). We used the Development & Enhancement 2018 Release 1 data set. The preprocessing and project selection procedures were based on known recommendations [9, 30]. We selected projects with the following characteristics [9, 30]: (a) Data points quality A or B. For IFPUG 4+ data sets, projects must also have function point quality A or B. (b) Recorded effort accounts only for development team. (c) Development type is new development. (d) Functional sizing with the selected approach (IFPUG 4+ or COSMIC).

With the selected projects, we split the data set in subsets based on the function point measure unit: IFPUG 4+ and COSMIC. We select the IFPUG method as it is the most used in the industry [12] and COSMIC as it has had high adoption in the latest years [10]. Moreover, we analyze the performance of the estimation models using base functional components (BFC) and unadjusted function points (UFP) as BFC are correlated with effort and can affect the performance of the estimation models, quesada2016cosmic. Thus, the study used four data sets obtained from ISBSG 2018 Release 1: IFPUG 4+ unadjusted function points (UFP), IFPUG 4+ basic functional components (BFC), COSMIC full function points (FFP), and COSMIC BFC. The preprocessing was applied to each data set independently. Outlier values were not removed.

The feature selection procedure was performed based on the protocol of Dejaeger *et al.* [9] and recommendations detailed in González-Ladrón-de-Guevara *et al.* [14]. The following feature selection guidelines were defined:

(1) Only retain features relevant for effort estimation
(2) Dimensionality reduction by removal of redundant features
(3) Remove features that are not available at the time of estimation, such as Project Elapsed Time.
(4) Remove features with more than 25% missing values.

The exception to the second guideline was the function point total. Instead, functional size features were selected depending on the counting approach. For IFPUG 4+ UFP and COSMIC FFP, the Functional Size feature was selected. For IFPUG 4+ BFC, the features Input count, Output count, Enquiry count, and File count were selected. For COSMIC BFC, the features COSMIC Entry, COSMIC Exit, COSMIC Read, COSMIC Write were selected. In all cases, the remaining software size features were removed. In addition to these, each set retained the following features: Application Group, Architecture, Case Tool Used, Development Methodologies, Development Platform, Industry Sector, Intended Market, Language Type, Max Team Size, Team Size Group, Used Methodology, and

Table 1: ISBSG 2018 Release 1 data sets

Name	Projects	Features
COSMIC BFC	168	15
COSMIC FFP	168	12
IFPUG 4+ BFC	821	18
IFPUG 4+ UFP	821	14

Year of Project. Summary Work Effort was the target feature. Table 1 contains the final descriptions of each data set.

3.2 Machine Learning Model Evaluation

In this study, machine learning models were trained and evaluated using the data sets presented in section 3.1. The data was partitioned using a hold-out group, where 90% of the data was used for constructing the models and collecting the performance metrics, and the remaining 10% was used as a prediction set or test set. The evaluation process was conducted using the 90% of the data, and was used to obtain the accuracy metrics presented in this study.

A second validation approach was used on the 90% of the data selected to construct the models. We validated the performance of the investigated SEE approaches through 10 times 10-fold Cross-Validation (CV), based on previous work by Song *et al.* [31]. The accuracy metrics were calculated for each validation fold.

After the metrics were obtained for the 90% set, the models were evaluated one more time using the 10% set. To accomplish this, all evaluated models were re-trained using the entirety of the 90% set and then were used to predict effort for the 10% set. The metrics obtained were compared to those reported in the study, with the objective of detecting problems such as overfitting and data mismatch. In this case, the results obtained for the test set did not greatly differ from those presented in the study.

3.3 Machine learning algorithms and hyper-parameter settings

We constructed the SEE models by combining the following methods: logarithmic (Log) data transformation (DT), 2 feature selection (FS) methods—variance threshold (VT) and correlation percentile (CP)—, and 2 machine learning algorithms(MLA)—support vector regression (SVR) and ridge regression (RR). For model tuning, we used 2 hyper-parameter tuning (PT) approaches:—random search (RS) and grid search (GS). A total of twenty-four models were compared. For example, we compared RS+Log+SVR with Log+SVR and GS+Log+SVR. We did not investigate VT+SVR and CP+SVR, as we determined from preliminary runs that feature selection had almost no effect on accuracy when using SVR. We used the implementation from the scikit-learn library for Python (https://scikit-learn.org/) for all studied models. The following sections explain each technique and the reasoning behind its use.

3.3.1 Hyper-parameter tuning.

Grid search. Grid search is a hyper-parameter tuning approach that evaluates each possible parameter combination in the search space [4]. The search space is formed by a hyper-parameter grid: a multi-dimensional space with one dimension per parameter. A point in the search space is defined by a value along each dimension. For example, a valid point in the search space comprised by hyper-parameters $C = \{100, 150, 200\}$ and $\gamma = \{0.001, 0.01, 0.1\}$ would be $(C = 150, \gamma = 0.01)$. Grid search explores all combinations of values for each parameter (i.e., the entire search space). For each such combination (point in the search space), a model is built and evaluated using those hyper-parameter values. The hyper-parameter combination with the highest accuracy is reported. The scikit-learn implementation of grid search search uses cross-validation for the search process. We selected a 10-fold cross-validation approach.

Random search. Random search is an hyper-parameter tuning approach that samples a subset of the search space, making it less computationally expensive than grid search. Additionally, random search provides a level of accuracy improvement comparable to grid search [3]. The theoretical soundness of random search is probability: assuming that at least 5% of all points in the hyper-parameter space are optimal (or close) solutions, by sampling 60 points there is a 95% chance at least one of them will be in the top-performing hyper-parameters [38]. Thus, we used a sample of 60 hyper-parameters in this study. Similar to grid search, we employed an inner 10-fold cross-validation approach.

Default hyper-parameters. To represent the scenario whithout hyper-parameter tuning, we used the default hyper-parameter values for each studied method, as defined by the scikit-learn library. This is our baseline for determining the accuracy improvement of hyper-parameter tuning. In scikit-learn, the default hyper-parameters for SVR are: a) kernel = rbf, b) $C = 1$, and $\epsilon = 0.1$ c) $\gamma = 1/(n * var)$ where n = number of features, var = variance of the data set. The default hyper-parameter for ridge regression is $\alpha = 1.0$. The default hyper-parameter for VT is $threshold = 0$. The default hyper-parameter for CP is $percentile = 10$. Regarding the kernel hyper-parameter for SVR, its default value is radial basis function (rbf). This kernel has been successfully used in multiple SEE studies [8, 17], and has shown to work well in both high and low dimensional feature spaces when adequately adjusted [18].

3.3.2 Machine learning algorithms.

Support vector regression. Support vector machines (SVM) are a type of model useful in high dimensional feature spaces [28]. This technique searches for a boundary that splits the data based on the target feature. Support vector regression is an approach based on SVMs that is suitable for prediction problems [26]. SVR was chosen for this study as it is a technique that has been used in SEE literature [36], and whose accuracy depends on appropriate hyper-parameter settings [19].

Ridge regression. Ridge regression is an approach based on ordinary least squares (OLS) regression, which addresses the problem of highly correlated attributes [9, 15, 16] by estimating the coefficients using a ridge estimator. The ridge estimator is biased but has a lower variance than the OLS estimator, due to the penalty factor (λ), which penalizes high values of β (coefficient), resulting in coefficient shrinkage. This model is selected as a baseline for comparison against SVR, as previous studies have reported that regression-based models outperform more complex machine learning methods [9]. The scikit-learn implementation of this approach applies standardization to the data. Ridge regression was selected as a baseline technique that has been successfully used in previous SEE studies, such as Dejaeger *et al.* [9], Ertuğrul *et al.* [11], and Malgonde et al. [22].

3.3.3 *Feature selection.*

Variance threshold. Variance thresholding (VT) is a basic feature selection approach, based on the idea that low variance features could be less useful than high variance features [2]. The method calculates the variance of each feature, and then drops all those whose variance is under the threshold. The sci-kit learn library implementation has one hyper-parameter: the threshold value.

Correlation percentile. Pearson's correlation coefficient can be used as a filter approach for feature selection [28]. A feature selection approach based on this metric calculates the correlation between each feature and the target feature. Features that have high correlation with the target are likely to be very informative for model training [2]. Based on these correlations, a subset of the most correlated features is selected.

Correlation percentile is a method based on the *SelectPercentile* feature selector of the scikit-learn library and Pearson's correlation coefficient. The method calculates the correlation between each feature and the target feature, and then selects those features with the highest correlation in the percentile. This percentile can be adjusted as a hyper-parameter of the method.

3.3.4 *Data transformations.*

Log. The logarithmic transformation has been traditionally used in SEE studies [7, 9]. We used a modified version of this transformation, which is defined as $x = log(1 + x)$, where x is each numerical feature. The logarithmic transformation is used to address two problems in the data: 1) large differences in the feature ranges that can bias the model, and 2) non-linearity in the feature space that may affect the applicability of linear methods [7].

Whether or not the Log transformation is used, the following transformations were applied to the data:

- Missing values are treated using 1NN (k-Nearest neighbors with $k = 1$) imputation [5].
- Categorical features are transformed via one-hot-encoding. This representation converts a categorical feature with K unique values into K binary features. These features are exempted from the data transformation technique.

3.3.5 Hyper-parameter values. The hyper-parameter values researched in this study are shown in table 2. These values were selected from existing recommendations in the literature [22, 31], with modifications product of adjustment performed in preliminary iterations. The grid size (amount of possible parameter combinations) for each technique is as follows: 4128 for SVR, 29 for RR, 232 for VT+RR, and 261 for CP+RR. For the γ parameter of SVR, the value auto equals to $1/n$ and scale equals to $1/(n * var)$, where n = number of features.

3.4 Performance metrics

We measured the prediction accuracy of SEE models using several metrics based on the absolute residual: $AR_i = |\hat{y}_i - y_i|$, where y_i is the observed effort value for the i-th project on the test set, and $\hat{y}_i$ is the predicted effort for the same project. The absolute error is calculated for each predicted value. The average of these errors is the mean absolute residual (MAR):

$$MAR = \frac{1}{n}\sum_{i=1}^{n} AR_i. \tag{1}$$

The median of the absolute residuals $MdAR$ is another summary metric that is more resilient to outliers than MAR.

We can also calculate the standard deviation $SdAR$ of the AR values:

$$SdAR = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(\hat{y}_i - MAR)}. \tag{2}$$

The standardized accuracy, defined by Shepperd and MacDonell [29], is a ratio of how much better is the estimation model than the baseline model. A value of 0 indicates the same accuracy as a random guessing, while a negative value indicates lower accuracy than random guessing. The use of such metric is recommended for comparability with other studies. We used the modified version proposed by Minku [23], which employs $MdAR$ instead of MAR, as follows:

$$SA = 1 - \frac{MdAR}{MdAR_{p0}}, \tag{3}$$

where $MdAR_{p_0}$ is the median absolute residual of the baseline predictor p_0. We used the random estimation baseline model recommended by Langdon *et al.* [20].

To answer RQ1, the improvement in accuracy was determined as $imp = SA_1 - SA_2$, where SA_1 and SA_2 are the standardized accuracy of the tuned and untuned models, respectively. Positive values indicate an improvement in accuracy, whereas negative values indicate a diminishing in accuracy. A value of 0 indicates that tuning has no effect in performance. The magnitude of changes was quantified using the standardized metric described by Glass [25]:

$$\Delta = \frac{MdAR_1 - MdAR_2}{SdAR_2}, \tag{4}$$

where MAR_1 is the MAR obtained by the hyper-parameter tuning approach, and MAR_2 and $SdAR_2$ are the MAR and $SdAR$ obtained by default hyper-parameters, respectively. To interpret this metric, we use the categories proposed by Cohen [6]:

Table 2: Hyper-parameter values

Approach	Hyper-parameters and values
SVR	kernel = {rbf, sigmoid} $\gamma = (10^x, x = \{-3, -2.5, -2, \ldots, -0.5\})$; auto, scale $C = 1, 5, 15, 30, \{50, 100, \ldots, 450\}, \{500, 1000, \ldots, 15000\}$ $\epsilon = 10^x, x = \{-3, -2.5, -2, \ldots, -0.5\}$
RR	$\alpha = 1, \{5, 10, \ldots, 45\}, \{50, 75, \ldots, 500\}$
VT+RR	$threshold = 0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5$ $\alpha = 1, \{5, 10, \ldots, 45\}, \{50, 75, \ldots, 500\}$
CP+RR	$percentile = \{10, 20, 30, \ldots, 90\}$ $\alpha = 1, \{5, 10, \ldots, 45\}, \{50, 75, \ldots, 500\}$

$$\text{effect size} = \begin{cases} \text{negligible} & \text{if } \Delta \leq 0.2 \\ \text{small} & \text{if } 0.2 < \Delta \leq 0.5 \\ \text{medium} & \text{if } 0.5 < \Delta \leq 0.8 \\ \text{large} & \text{if } 0.8 < \Delta \end{cases} \tag{5}$$

To answer RQ2, the stability ratio of tuned against default hyper-parameters was calculated as [33]:

$$\text{stability ratio} = \frac{SdAR_1}{SdAR_2} \tag{6}$$

where $SdAR_1$ and $SdAR_2$ are the $SdAR$ obtained by tuned hyper-parameters and default hyper-parameters, respectively. The stability ratio metric functions as an inverse of SA. A stability ratio of 1 indicates that the tuning method is equally stable as the default hyper-parameters. Whereas values above 1 indicate that the tuning approach induces instability, and values smaller than 1 indicate an increase or improvement in stability.

3.5 Threats to validity

Internal validity. The accuracy of the constructed models is measured in standardized accuracy, which is calculated using a baseline random model. In essence, the metric shows the ratio of improvement over random guessing; and thus shows if the accuracy of a model is due to random factors or due to the selected techniques. In addition, we report other accuracy measures commonly used in the literature to increase our comparability with other studies. Another factor that may affect the causality of the metrics is the split of training and validation data. To mitigate this effect, we have employed 10 times 10-fold cross-validation.

Construct validity. One threat to construct validity is the variability of the ISBSG data set. As data are collected from multiple sources, some variations on the measurements could affect the integrity of the data. To mitigate this effect, we selected only those projects with high data quality rating. Another threat is the search space for the hyper-parameter tuning approaches, as there are constraints on the execution time of this study. To reduce this threat, the search space for these techniques was selected and validated both through existing studies in tuning and empirical runs before the experiment. Missing value imputation introduces some threats to construct validity.

External validity. This study covers only the ISBSG 2018 Release 1 data set, thus limiting the reach of generalization of results to other data sets or projects. Future work includes the extension of this study into different data sets to provide more general results.

Conclusion validity. We use a large number of cross-validation iterations to provide enough experimental runs and measurements. Besides, we study the properties of the collected data (i.e. normality) and validate the accuracy metrics with those from the test set. We also validated the obtained results with previous SEE literature. One threat to conclusion validity is the obtained effect size for the hyper-parameter tuning approaches. According to the categories by Cohen, tuning was negligible in all but two cases. This could indicate that the results obtained in this study could be due to randomness instead of a significant difference between tuned vs. default parameters. However, results of the Scott-Knott analysis show that tuned and untuned SVR models have significant SA differences.

4 RESULTS

4.1 Prediction accuracy improvement when using random search

This section answers RQ1, which aimed at finding the improvement in SVR prediction accuracy when using random search. For this, we contrast hyper-parameter tuned models with untuned models (using default hyper-parameters).

The accuracy improvements and effect size achieved by random search for the SEE models constructed using each data set are shown in Figure 1. The overall accuracy improvement, represented as the median improvement (and standard deviation) for the models are summarized in Table 3. The use of random search allowed for the identification of a median increase in SA of up to 0.227. When analyzing the ranges, we observed that the maximum value for SA improvement was around 1, outliers excluded. The improvement provided by random search is not negligible for the ridge regression model and the VT-RR, for the COSMIC FFP data set.

The performance improvement provided by random search has an intrinsic relationship to the data set. Random search provided accuracy improvements on models trained with the COSMIC data sets than those trained with IFPUG 4+. It is also important to note that the effect of tuning was far more noticeable when function point total was used instead of BFC. Interestingly, also random search decreased the performance of several models: 1 in COSMIC BFC, 1 in IFPUG 4+ BFC, and 2 in IFPUG 4+ UFP, as can be seen in Table 3. In all cases, this decrease in performance was less than 0.05 SA.

The model that benefited the most of hyper-parameter tuning was Log+SVR, having the largest performance increase in three out of four data sets, and median increases in SA ranging from 0.129 to 0.227, as shown in table 3. Random search also had a positive effect on the SVR model, improving accuracy in all data sets. The ridge regression models were mostly improved when using function point totals, but had their performance mostly unaffected in BFC sets.

Random search provided similar increases in accuracy with respect to grid search. For both SVR models—SVR and Log+SVR—the difference between grid search and random search does not exceed 0.21 SA, with the contrast that random search used a smaller search space, obtaining results in less time. For the RR models, grid search offers, at best, an increase of 0.045 SA over random search. One explanation for random search having less performance than grid may be due to the 5% top-performing parameters assumption of the algorithm not applying to the searched space.

(RQ1) Random search increases the accuracy of SVR models to a maximum of 0.227 SA. SVR models benefited from random search in all of the data sets. The accuracy improvement obtained by random search and grid search was very similar, with grid search surpassing random search by at most 0.045 SA.

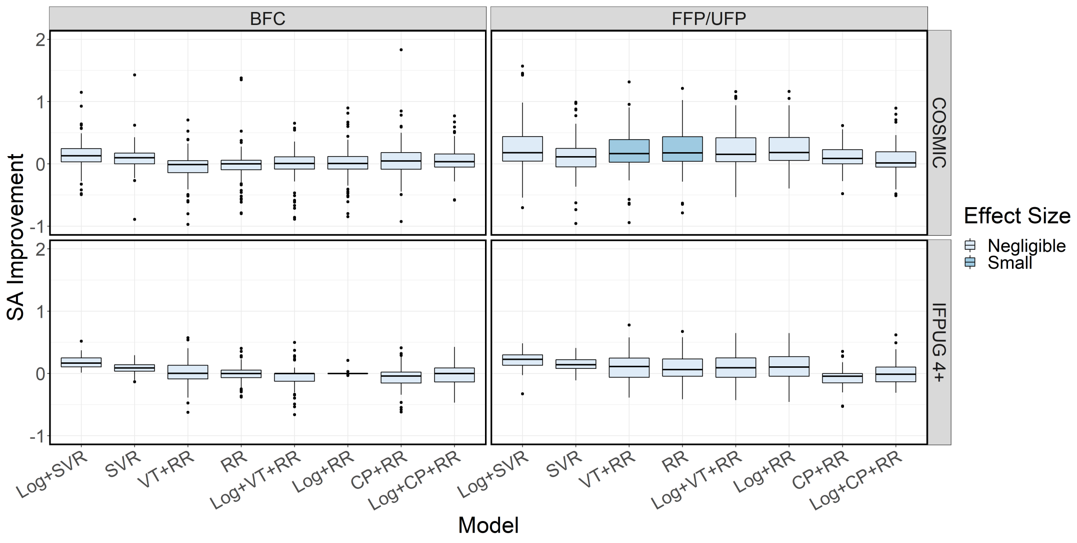


Figure 1: Accuracy improvement and effect size of random search.

Table 3: Median and deviation accuracy improvement of parameter tuning.

	COSMIC				IFPUG 4+			
	BFC		FFP		BFC		UFP	
	Md	Sd	Md	Sd	Md	Sd	Md	Sd
RS+SVR	0.099	0.220	0.113	0.317	0.088	0.082	0.140	0.106
RS+RR	0.000	0.297	0.177	0.338	0.000	0.144	0.064	0.223
RS+VT+RR	-0.012	0.291	0.165	0.335	0.002	0.211	0.113	0.235
RS+CP+RR	0.047	0.321	0.081	0.230	-0.041	0.182	-0.045	0.140
RS+Log+SVR	**0.129**	**0.261**	0.180	0.406	**0.166**	**0.094**	**0.227**	**0.129**
RS+Log+RR	0.005	0.279	**0.183**	**0.316**	0.000	0.022	0.104	0.237
RS+Log+VT+RR	0.005	0.277	0.152	0.349	0.000	0.165	0.091	0.241
RS+Log+CP+RR	0.036	0.214	0.014	0.244	0.000	0.191	-0.012	0.159
GS+SVR	0.102	0.232	0.092	0.323	0.105	0.076	0.156	0.098
GS+RR	0.000	0.297	0.177	0.338	0.000	0.144	0.064	0.223
GS+VT+RR	0.000	0.313	0.160	0.345	0.000	0.169	0.108	0.237
GS+CP+RR	0.018	0.249	0.083	0.233	0.000	0.167	0.000	0.067
GS+Log+SVR	**0.128**	**0.262**	0.171	0.433	**0.177**	**0.094**	**0.223**	**0.117**
GS+Log+RR	0.005	0.279	**0.183**	**0.316**	0.000	0.022	0.104	0.237
GS+Log+VT+RR	0.007	0.299	0.158	0.353	0.000	0.057	0.093	0.233
GS+Log+CP+RR	0.040	0.275	0.001	0.249	-0.014	0.175	-0.002	0.154

4.2 Prediction accuracy stability when using random search

This section answers RQ2, which sought to determine the stability of SVR prediction accuracy when using random search. For this, we compare the variability in accuracy produced across each cross-validation iteration and data set.

Table 4 shows the median and standard deviation of the stability ratio for each data set and SEE model. In the majority of cases, models tuned by random search were as stable as those with default hyper-parameters. The median stability ratio of all techniques across all data sets is below 1.040 and often below 1. At its best, random search was able to reduce the median stability ratio of a SEE model to 0.840, and up to a maximum reduction of around 0.5 stability ratio.

The accuracy of the Log+SVR and SVR models became more stable due hyper-parameter tuning. These techniques boast a median stability ratio of 0.842 to 0.986 for Log+SVR, and 0.840 to 1.004 for SVR, as shown on table 4. For ridge regression models, the stability ratio varies from 0.880 to 1.040. Most cases in which regression models had a median increase in variability were on the IFPUG 4+ data sets.

When comparing the stability ratios of the models constructed using random search and grid search, both techniques produced highly similar results. For both SVR and Log-SVR, the differences between grid search and random search did not exceed 0.045. In most cases, grid search showed a better stability than random search. However, these differences are not large. At best, grid search offered an increase in stability ratio of 0.008, when compared to random search applied to regression techniques. Thus, the accuracy stability achieved by random search and grid search can be deemed as similar.

Table 4: Median and deviation stability ratio of parameter tuning.

	COSMIC				IFPUG 4+			
	BFC		FFP		BFC		UFP	
	Md	Sd	Md	Sd	Md	Sd	Md	Sd
RS+SVR	1.004	0.074	**0.840**	**0.215**	0.958	0.043	**0.877**	**0.113**
RS+RR	1.000	0.119	0.971	0.204	1.000	0.248	1.006	0.239
RS+VT+RR	0.999	0.146	0.971	0.203	1.040	0.288	1.010	0.262
RS+CP+RR	0.981	0.192	0.996	0.112	1.000	0.188	1.000	0.122
RS+Log+SVR	0.986	0.085	0.842	0.210	**0.911**	**0.059**	0.915	0.085
RS+Log+RR	0.997	0.124	0.880	0.177	1.000	0.006	1.018	0.188
RS+Log+VT+RR	0.988	0.134	0.892	0.178	1.000	0.136	1.024	0.184
RS+Log+CP+RR	**0.980**	**0.095**	0.998	0.112	0.999	0.081	1.001	0.135
GS+SVR	1.002	0.084	0.885	0.254	0.958	0.043	**0.864**	**0.076**
GS+RR	1.000	0.119	0.971	0.204	1.000	0.248	1.006	0.239
GS+VT+RR	1.000	0.127	0.968	0.200	1.005	0.266	1.009	0.261
GS+CP+RR	**0.980**	**0.190**	0.996	0.117	1.000	0.075	1.000	0.033
GS+Log+SVR	0.984	0.082	**0.845**	**0.191**	**0.922**	**0.054**	0.909	0.078
GS+Log+RR	0.997	0.124	0.880	0.177	1.000	0.006	1.018	0.188
GS+Log+VT+RR	0.985	0.130	0.899	0.170	1.000	0.014	1.023	0.184
GS+Log+CP+RR	0.988	0.092	1.000	0.104	0.999	0.074	1.000	0.125

(RQ2) Random search maintained median prediction stability for all constructed models, with a maximum stability ratio of 1.040. SVR models gained the most stability when tuned with random search, to a minimum of 0.840. Accuracy stability obtained by random search and grid search was very similar, with at most grid search surpassing random search by a ratio difference of 0.045.

4.3 Ranking of SEE models based on standardized accuracy

This section answers RQ3, which strove for the best performing models. To achieve this, we ranked all the constructed SEE models. This was done using the Scott-Knott algorithm [27]. This method uses a hierarchical clustering algorithm to partition the treatments into equal groups. Starting from a group comprised by all treatments, the algorithm splits it into two non-overlapping groups. The procedure orders the groups by the accuracy metric and splits it into two groups by determining the largest difference. The process is repeated, for each group, if the treatments are not equal.

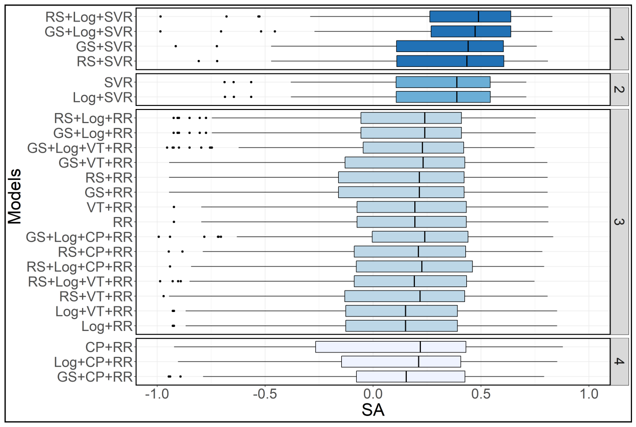
Figure 2 shows the ranked clusters and SA of each SEE model researched in this study, per data set. The median *SA*, *MdAR*, and *SdAR* of all researched SEE models across the different analyzed data sets is available in Table 5. The models that belong in the top group are highlighted. It can be appreciated that, the highest observations are comprised of RS+Log+SVR and GS+Log+SVR. In the case of COSMIC BFC, the highest group is comprised by RS+Log+SVR, GS+Log+SVR RS+SVR, and GS+SVR. Based on the results of the Scott-Knott method, RS+Log+SVR and GS+Log+SVR always outperformed their default counterpart, Log+SVR. Similarly, RS+SVR and GS+SVR always ranked above both SVR and Log+SVR. It is also noteworthy, that our results indicated that tuned Log+SVR outperformed other models such as RR or VT+RR. RS+Log+SVR has a median *SA* from 0.398 to 0.488 across all data sets. GS+Log+SVR has a median *SA* from 0.420 to 0.486 across all data sets. This shows a moderate improvement of prediction accuracy over the baseline. Moreover, this indicates that, for all data sets, random search has the same accuracy than grid search when applied to Log+SVR.

The SVR models presented similar or better prediction accuracy than the RR models. Second to the tuned Log+SVR models, the SVR, Log+SVR, RS+SVR, and GS+SVR models formed the highest ranked groups for the COSMIC BFC, IFPUG 4+ BFC, and IFPUG 4+ UFP. Interestingly, The COSMIC FFP data set is the exception; the third group being comprised of both untuned SVR and tuned RR models. The median *SA* of the SVR models always was above 0.2; ranging from 0.325 to 0.434 for the RS+SVR model, from 0.301 to 0.440 for the GS+SVR model, from 0.245 to 0.378 for the Log+SVR model, and from 0.245 to 0.378 for the SVR model. Moreover, the RS+SVR and GS+SVR models belonged to the same group in all data sets. This indicates that, random search has the same accuracy than grid search when applied to SVR.

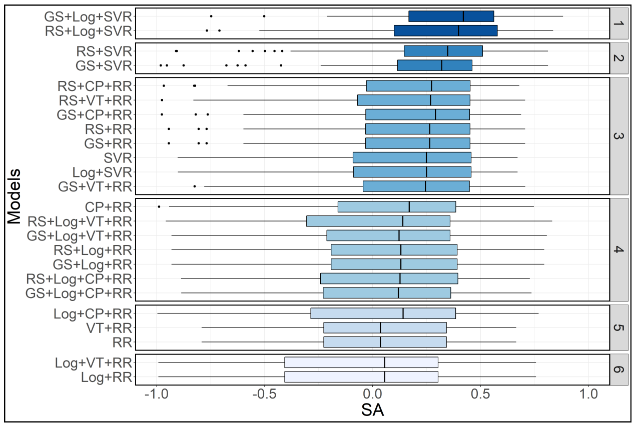
The accuracy of ridge regression models depends on the data set. For COSMIC data sets, the median *SA* of ridge models ranged from −0.02 to 0.266, and in the majority of cases (41 out of 72) the median *SA* was between −0.1 and 0.3. On COSMIC BFC, the hyper-parameter tuned RR models have a larger *SA* than their unoptimized counterparts (excepting one case). For IFPUG 4+ data sets, the performance of ridge regression models was always below the baseline. The exception to this tendency were the three CP+RR models in the IFPUG 4+ UFP data set.

SVR models generally had better performance in BFC data sets than in FFP data sets, which would indicate that there is value in using the basic functional components as features for SEE. Future work could further study the use of BFC against or along with the functional size.

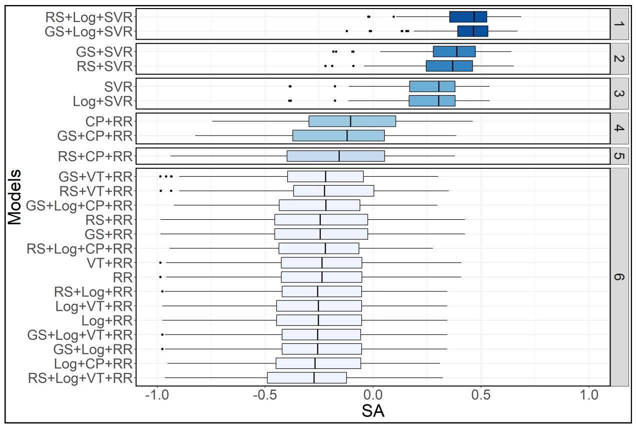
(RQ3) Tuned SVR models outperformed all other studied SEE models. Particularly, RS+Log+SVR and GS+Log+SVR placed in the top group in all datasets. These models achieved a maximum median *SA* of 0.488. Other SVR-based models often achieved high places in the ranking. In addition, SVR models tuned with random search were equivalent in accuracy to SVR models tuned with grid search.



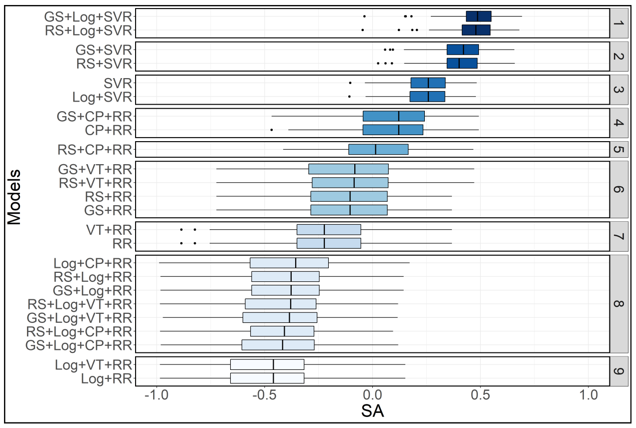
(a) COSMIC BFC



(b) COSMIC FFP



(c) IFPUG 4+ BFC



(d) IFPUG 4+ UFP

Figure 2: Scott-Knott clusters and SA of the studied SEE models.

5 DISCUSSION

Our results confirmed previous findings in the SEE literature. The hyper-parameter tuned SVR model had a significantly better performance than other regression methods. Besides, tuned SVR models had better prediction accuracy than untuned SVR. However, the effect of random search on the accuracy was negligible, according to the metric based on Glass's Delta. Further research with a larger hyper-parameter search space is necessary to corroborate this result.

Results obtained as part of RQ2 showed that random search was able to maintain or increase prediction stability, compared to default settings. This confirms previously reported results by Tantithamthavorn *et al.* [33]. The increase in stability was larger for SVR models, showing the importance of appropriate hyper-parameter tuning.

For COSMIC function points, the prediction accuracy for SVR models was higher when models were trained and tested with data sets that used basic functional components over function point total. This goes against previous studies that use the ISBSG repository [9, 11, 17, 23, 32], as they choose to select the function point total over the individual components. Our results show that SEE models could benefit from the use of BFC as features. For instance, SEE models could determine if BFC affected effort differently, depending on other features. For example, input points could involve more effort in projects developed with programming language A over programming language B. Future work could research use BFC alongside total functional size for SEE models.

Results obtained across all research questions show that the performance of random search is similar to that of grid search, when applied to support vector regression. RQ1 showed that the increase in accuracy with respect to default parameters was very similar, with a difference of at most 0.045 *SA*. Similarly, RQ2 shows that the stability of random search and grid search is very similar, with a difference of at most 0.045. Lastly, RQ3 shows that SVR models tuned with random search have performance equivalent to those tuned with grid search. These results suggest that random search could be used as a baseline technique for research on hyper-parameter tuning for SVR in the ISBSG data set, with almost no effect on prediction accuracy. Further comparisons among the techniques are necessary before being able to generalize this statement to other data sets and MLAs.

Table 5: Median SA, MdAR, and SdAR of all models.

	COSMIC						IFPUG 4+					
	BFC			FFP			BFC			UFP		
	SA	MdAR	SdAR	SA	MdAR	SdAR	SA	MdAR	SdAR	SA	MdAR	SdAR
SVR	0.378	2186	3664	0.245	2162	4111	0.305	1918	9297	0.259	1786	9968
RR	0.157	2667	3498	-0.020	2659	2909	-0.255	3285	6636	-0.224	2890	6450
VT+RR	0.157	2667	3498	-0.020	2659	2909	-0.255	3285	6636	-0.224	2890	6450
CP+RR	0.175	2946	2959	0.149	2386	2629	-0.118	2805	6679	0.121	2071	7035
Log+SVR	0.378	2184	3664	0.245	2160	4112	0.304	1916	9294	0.258	1798	9975
Log+RR	0.124	2790	3195	-0.010	2918	2796	-0.270	3238	6490	-0.493	3645	6754
Log+VT+RR	0.124	2790	3195	-0.010	2918	2796	-0.270	3238	6490	-0.493	3645	6754
Log+CP+RR	0.137	2921	2926	0.129	2446	2510	-0.273	3270	6675	-0.383	3295	6872
RS+SVR	**0.434**	**1882**	**3709**	0.325	1928	3128	0.369	1638	8686	0.401	1421	8538
RS+RR	0.173	2775	3214	0.248	2035	2730	-0.246	3294	7151	-0.104	2632	6609
RS+VT+RR	0.133	2800	3137	0.262	2071	2732	-0.237	3190	7468	-0.085	2633	6644
RS+CP+RR	0.137	2753	2864	0.266	2109	2704	-0.169	3007	6841	0.014	2291	6905
RS+Log+SVR	**0.488**	**1662**	**3444**	**0.398**	**1515**	**3058**	**0.468**	**1434**	**8375**	**0.478**	**1212**	**8987**
RS+Log+RR	0.215	2779	2972	0.099	2391	2373	-0.270	3228	6490	-0.385	3388	6765
RS+Log+VT+RR	0.134	2804	2915	0.126	2395	2337	-0.298	3453	6564	-0.389	3408	6761
RS+Log+CP+RR	0.134	2794	2778	0.096	2369	2342	-0.251	3233	6400	-0.414	3418	6708
GS+SVR	**0.440**	**1810**	**3602**	0.301	1998	3196	0.388	1604	8652	0.421	1394	8496
GS+RR	0.173	2775	3214	0.248	2035	2730	-0.246	3294	7151	-0.104	2632	6609
GS+VT+RR	0.203	2789	3199	0.223	2129	2758	-0.226	3220	7251	-0.082	2615	6644
GS+CP+RR	0.107	2858	2891	0.253	2077	2739	-0.129	2850	6592	0.121	2082	7035
GS+Log+SVR	**0.473**	**1685**	**3453**	**0.420**	**1561**	**2945**	**0.465**	**1424**	**8470**	**0.486**	**1196**	**9099**
GS+Log+RR	0.215	2779	2972	0.099	2391	2373	-0.270	3228	6490	-0.385	3388	6765
GS+Log+VT+RR	0.206	2789	3002	0.118	2433	2388	-0.270	3228	6490	-0.401	3408	6765
GS+Log+CP+RR	0.139	2633	2853	0.094	2394	2379	-0.242	3261	6488	-0.419	3468	6736

We compared the results obtained from this study with those from recent (2018–2019) SEE studies that use hyper-parameter tuning, SVRs, and the ISBSG data set.

Ertuğrul *et al.* [11] performed an experiment to compare 9 different machine learning algorithms, including grid search-tuned SVR. They partition the ISBSG Release 11 data set into five sub-sets depending on their effort size, and using IFPUG 4+ total functional size. In their second case study, using 10-fold CV, they report MAR values for the SVR model of 1242, 825, 1098, 838, and 5847. In comparison, our RS+Log+SVR and GS+Log+SVR models achieved a median *MdAR* of 1196 and 1212, respectively. In three cases, our study resulted in a model with a larger absolute residual, even when using a larger amount of data. Grouping of ISBSG projects according to project size is a worthwhile data transformation to explore in future studies.

Song *et al.* [32] proposes a prediction interval estimator called Synthetic Bootstrap ensemble of Relevance Vector Machines (SynB-RVM). They perform an evaluation using seven partitions of the ISBSG Repository Release 10, using IFPUG FPA. This evaluation compares their proposed technique to multiple hyper-parameter tuned point estimators, including SVR. For the 7 studied data sets, SVR resulted in median accuracy scores of *SA* of 0.465, 0.363, 0.461, 0.364, 0.325, 0.327, 0.247. In terms of *MdAR*, their SVR scored 546, 828, 517, 2118, 3955, 2032, and 3807. Our RS+Log+SVR achieved a median *SA* of 0.478 and a *MdAR* score of 1212, and our GS+Log+SVR a median *SA* of 0.486 and a *MdAR* score of 1196. We thus verify that the results of this study are in line with those previously reported in SEE literature.

Hosni *et al.* [17] study the effect of hyper-parameter values on heterogeneous ensemble effort estimation. We focus on the results obtained in the first experimental study presented by the paper, in which four base techniques, including SVR, are trained on the ISBSG Repository Release 8 data set, using IFPUG FPA. The study applies three hyper-parameter tuning approaches—grid search (GS), particle swarm optimization (PSO), and uniform configuration (UC, default hyper-parameters). For the GS-SVR, PSO-SVR, and UC-SVR models, the study reported *SA* values of 0.558, 0.605, and 0.463. The increase in accuracy achieved by grid search is 0.095. While the RS+Log+SVR model scored lower *SA* values (0.478 in IFPUG 4+ UFP), it was able to achieve a higher increase in *SA* using random search 0.227. The GS+Log+SVR model also resulted in a higher increase in *SA* with respect to default parameters (0.223). This increase in accuracy could potentially be attributed to the larger amount of projects in the ISBSG 2018 Release 1 and a larger search space. The results are similar to those previously reported in the SEE literature. The difference in results can be attributed to the version of the ISBSG data set, as well as the preprocessing applied to the data.

6 CONCLUSION

In this paper we evaluated the the impact of hyper-parameter tuning using random search (RS), and compared it with the accuracy of models tuned using the more exhaustive grid search in support vector regression algorithms. Our RS tuned SVR models were compared to multiple estimators, which included those subjected to tuning by RS and GS, as well as non-tuned models. The study used 4 sub sets of the ISBSG 2018 Release 1 data set to train and evaluate these models. Our findings indicated that performance of RS-tuned models was highly similar to those of grid search-tuned SEE models.

Furthermore, this study demonstrated that use of RS for model tuning could provide an improvement in model stability. This observation was considerable for SVR models, which had the best median metrics for accuracy and stability. Because RS searches a limited parameter space or number of iterations, compared to grid search, this could have accounted for some instances were our results were lower than expected. Finally, we identified that model tuning and data processing, in this case using logarithmic transformation, improved model performance.

The results in our work confirm: 1) the use of hyper-parameter tuning and data processing was crucial in constructing capable predictive SVR models, 2) model stability and accuracy are improved by the use adequate hyper-parameter tuning strategies, such as grid search or random search, and 3) the performance of the less exhaustive random search was comparable to the costly grid search, making random search a viable alternative for hyper-parameter tuning.

Future work encompasses various directions. One possibility would be extending this study to further compare random search and grid search using the data transformations, feature selectors, and machine learning algorithms that are most used in SEE literature. The evaluation performed in this study could also be replicated in more data sets to further generalize the obtained results. Future work could also explore comparing the performance of other hyper-parameter tuning approaches, such as hill climbing, genetic, and other search algorithms. Another line of research would be to investigate further the properties of the ISBSG 2018 Release 1 data set, and determine which preprocessing techniques would help increase prediction accuracy.

ACKNOWLEDGMENTS

This work was supported by project No. 834-B8-A27 at the University of Costa Rica (ECCI-CITIC).

REFERENCES

[1] Amritanshu Agrawal, Wei Fu, Di Chen, Xipeng Shen, and Tim Menzies. 2019. How to" DODGE" Complex Software Analytics. *IEEE Transactions on Software Engineering* (2019).
[2] Chris Albon. 2018. *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning.* " O'Reilly Media, Inc.".
[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, Feb (2012), 281–305.
[4] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.
[5] Michelle H Cartwright, Martin J Shepperd, and Qinbao Song. 2004. Dealing with missing software project data. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717)*. IEEE, 154–165.
[6] Jacob Cohen. 1992. A power primer. *Psychological bulletin* 112, 1 (1992), 155.
[7] Anna Corazza, Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, Federica Sarro, and Emilia Mendes. 2010. How effective is tabu search to configure support vector regression for effort estimation?. In *Proceedings of the 6th international conference on predictive models in software engineering*. 1–10.
[8] Anna Corazza, Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, Federica Sarro, and Emilia Mendes. 2013. Using tabu search to configure support vector regression for effort estimation. *Empirical Software Engineering* 18, 3 (2013), 506–546.
[9] Karel Dejaeger, Wouter Verbeke, David Martens, and Bart Baesens. 2011. Data mining techniques for software effort estimation: a comparative study. *IEEE transactions on software engineering* 38, 2 (2011), 375–397.
[10] Reiner Dumke and Alain Abran. 2016. *COSMIC Function Points: Theory and Advanced Practices*. CRC Press.
[11] Egemen Ertuğrul, Zakir Baytar, Çağatay Çatal, and Ömer Can Muratli. 2019. Performance tuning for machine learning-based software development effort prediction models. *Turkish Journal of Electrical Engineering & Computer Sciences* 27, 2 (2019), 1308–1324.
[12] S Fingerman. 2011. Practical software project estimation; a toolkit for estimating software development effort & duration. *Sci-Tech News* 65, 1 (2011), 28.
[13] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135–146.
[14] Fernando González-Ladrón-de Guevara, Marta Fernández-Diego, and Chris Lokan. 2016. The usage of ISBSG data fields in software effort estimation: A systematic mapping study. *Journal of Systems and Software* 113 (2016), 188–215.
[15] Arthur E Hoerl and Robert W Kennard. 1970. Ridge regression: applications to nonorthogonal problems. *Technometrics* 12, 1 (1970), 69–82.
[16] Arthur E Hoerl and Robert W Kennard. 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* 12, 1 (1970), 55–67.
[17] Mohamed Hosni, Ali Idri, Alain Abran, and Ali Bou Nassif. 2018. On the value of parameter tuning in heterogeneous ensembles effort estimation. *Soft Computing* 22, 18 (2018), 5977–6010.
[18] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2003. A practical guide to support vector classification. https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf. Accessed: 2020-07-07.
[19] S Sathiya Keerthi. 2002. Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms. *IEEE Transactions on Neural Networks* 13, 5 (2002), 1225–1229.
[20] William B Langdon, Javier Dolado, Federica Sarro, and Mark Harman. 2016. Exact mean absolute error of baseline predictor, MARP0. *Information and Software Technology* 73 (2016), 16–18.
[21] Gang Luo. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics* 5, 1 (2016), 18.
[22] Onkar Malgonde and Kaushal Chari. 2019. An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering* 24, 2 (2019), 1017–1055.
[23] Leandro L Minku. 2019. A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Software Engineering* (2019), 1–52.
[24] Adriano LI Oliveira, Petronio L Braga, Ricardo MF Lima, and Márcio L Cornélio. 2010. GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *information and Software Technology* 52, 11 (2010), 1155–1166.
[25] Robert Rosenthal, Harris Cooper, and L Hedges. 1994. Parametric measures of effect size. *The handbook of research synthesis* 621, 2 (1994), 231–244.
[26] Bernhard Schlkopf, Alexander J Smola, and Francis Bach. 2018. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* the MIT Press.
[27] Andrew Jhon Scott and M Knott. 1974. A cluster analysis method for grouping means in the analysis of variance. *Biometrics* (1974), 507–512.
[28] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms.* Cambridge university press.
[29] Martin Shepperd and Steve MacDonell. 2012. Evaluating prediction systems in software project estimation. *Information and Software Technology* 54, 8 (2012), 820–827.
[30] Liyan Song, Leandro L Minku, and Xin Yao. 2013. The impact of parameter tuning on software effort estimation using learning machines. In *Proceedings of the 9th international conference on predictive models in software engineering*. 1–10.
[31] Liyan Song, Leandro L Minku, and Xin Yao. 2014. The potential benefit of relevance vector machine to software effort estimation. In *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*. 52–61.
[32] Liyan Song, Leandro L Minku, and Xin Yao. 2019. Software effort interval prediction via Bayesian inference and synthetic bootstrap resampling. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 1 (2019), 1–46.
[33] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering*. 321–332.
[34] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2018. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering* 45, 7 (2018), 683–711.
[35] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 847–855.
[36] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. 2012. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology* 54, 1 (2012), 41–59.
[37] Tianpei Xia, Rahul Krishna, Jianfeng Chen, George Mathew, Xipeng Shen, and Tim Menzies. 2018. Hyperparameter optimization for effort estimation. *arXiv preprint arXiv:1805.00336* (2018).
[38] Alice Zheng. 2015. Evaluating machine learning models: a beginner's guide to key concepts and pitfalls. (2015).