# A novel automated approach for software effort estimation based on data augmentation

SONG, L.; MINKU, L.L.; YAO, X.

*DOI:*
10.1145/3236024.3236052

# A Novel Automated Approach for Software Effort Estimation Based on Data Augmentation

Liyan Song
Southern University of Science and Technology, China
University of Birmingham, UK
songly@sustc.edu.cn

Leandro L. Minku
School of Computer Science, University of Birmingham, UK
L.L.Minku@cs.bham.ac.uk

Xin Yao
Southern University of Science and Technology, China and
University of Birmingham, UK
xiny@sustc.edu.cn

## ABSTRACT

**Background:** software effort estimation (SEE) usually suffers from data scarcity problem due to the expensive or long process of data collection. As a result, companies usually have limited data projects for effort estimation, causing unsatisfactory prediction performance. Few studies have investigated strategies to generate additional SEE data to aid such learning. **Aim:** to propose a synthetic data generator to address the data scarcity problem of SEE. The proposed approach should be general to be used with any state-of-the-art SEE method. Ideally, it should be simple and hardly have negative effect on SEE performance. **Method:** our synthetic generator enlarges the SEE data set size by slightly displacing some randomly chosen training examples. It can be used with any SEE method as a data preprocessor. Its effectiveness is justified with 6 state-of-the-art SEE models across 14 SEE data sets. We also compare our data generator against the only existing approach in the SEE literature. **Results:** our synthetic projects can significantly improve the performance of some SEE methods especially when the training data is insufficient. When they cannot significantly improve the prediction performance, they are not detrimental either. Besides, our synthetic data generator is significantly superior or perform similarly to its competitor in the SEE literature. **Conclusion:** our data generator plays a non-harmful if not significantly beneficial effect on the SEE methods investigated in this paper. Therefore, it is helpful in addressing the data scarcity problem of SEE.

## CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by regression**; *Bayesian network models*; Ensemble methods; • **Software and its engineering** → **Software creation and management**;

## KEYWORDS

Software effort estimation, data scarcity, synthetic data, data augmentation, data generation


*ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA*

ACM ISBN 978-1-4503-5573-5/18/11…$15.00
https://doi.org/10.1145/3236024.3236052

**ACM Reference Format:**
Liyan Song, Leandro L. Minku, and Xin Yao. 2018. A Novel Automated Approach for Software Effort Estimation Based on Data Augmentation. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3236024.3236052

## 1 INTRODUCTION

Software effort estimation (SEE) is the process of predicting the effort (e.g. in person-month or person-hour) required to develop a software system. It often takes place in the very early stage of software development, and is an important task in software project management. Over and under estimation can cause either waste of resources or result in compromising the product quality [13, 71].

One of the core challenges of SEE is the high cost associated with data collection [13, 60]. The collection of software projects is very costly and may require considerable amount of time and workload [37, 38, 43]. Consequently, companies usually have small numbers of completed projects to estimate the effort of new projects. It would be hard to make accurate estimates with inadequate SEE data because the information contained in such small data probably cannot support training of SEE models [24, 37, 66]. Existing work has frequently attempted to tackle this issue by creating advanced predictors that are more suitable for this problem [40, 43, 49].

Rather than introducing sophisticated SEE models or collecting as many completed projects as possible, we can augment SEE data set by generating synthetic projects based on the existing data. However, little work has been done to investigate such strategies. This paper proposes an automated data augmentation approach that can be used as a preprocessor for any SEE method. Our data generator produces additional synthetic projects by slight displacing some randomly chosen completed projects, with each synthetic data associated with one existing project. Though the synthetic projects are not 'real', they can enrich the representativeness of the area they are generated and potentially enhance the effort prediction.

Our data generator provides a second and much cheaper way to tackle SEE data scarcity problem compared to proposing sophisticated models or strategies of real data collection. To evaluate its effectiveness, we investigated the following research questions:

- RQ1 Given an SEE predictor, can our synthetic data generator help improving prediction performance over the baseline that does not use synthetic data? When? Could it be detrimental?
- RQ2 Given an SEE predictor, if our synthetic projects are helpful to prediction performance, why are they helpful? If they are detrimental, why are they detrimental?

RQ3 How well our data generator performs compared against other existing data generators in the SEE literature?

Experimental studies based on six state-of-the-art SEE models show supportive results of our data generator. Our synthetic projects always have positive effect on and are rarely detrimental to the baseline performance of the investigated SEE models, especially when the training data is insufficient. Besides, our data generator is similar or significantly superior to its only competitor in SEE [32].

The main contribution of this paper is to propose and validate a novel synthetic data generator, and provide the understanding of when and why the synthetic projects generated by this approach can help improving the baseline performance of the SEE model.

## 2 RELATED WORK

### 2.1 Data Augmentation for Classification

There are many studies in machine learning (ML) that augment the data set size for better performance. Imbalance classification is a typical example, where the difference between the numbers of data samples in different categories is huge [25]. One problem of learning from imbalanced data is that the classifiers would often predict a new sample with the majority label though its label should equal to the minority [70]. Data over-sampling is a popular and effective approach to tackle the imbalance classification problem, where synthetic data is generated in the minority class to form a more balanced data set for performance improvement [10, 19].

Software defect prediction (SDP) is a typical counterpart of imbalance classification in the context of software engineering (SE), since the defect modules are much less likely to happen than the non-defect ones. Data imbalance usually undermines the performance of SDP methods, where the defect predictors often rarely predict the faulty modules [54, 70]. To tackle the imbalance problem of SDP, a few methods that augment the data set size of the minority class (i.e. the faulty class) have been proposed [33, 54, 69, 70].

For instance, [33, 54, 69] employed several data augmentation methods in ML, such as random over-sampling that reproduces the data of minority class randomly and SMOTE (Synthetic Minority Over-sampling Technique) [10] that produces new samples based on $k$-nearest neighbours, to enlarge the data set size of the minority class. Their experimental results showed promising or better effect of the augmented data in performance improvement. Another genetic algorithm-based data augmentation method was proposed in [17], which outperformed the predictor without the augment data and the predictor with other augmentation methods.

### 2.2 Data Augmentation in SEE Literature

The augmentation methods designed for classification cannot be directly used for SEE since by nature there is no minority/majority class in regression. Section 2.1 is discussed for being among the most related to our work. Despite many studies on synthetic over-sampling for classification, there have been few for regression (e.g. SMOTER [9, 64, 65] and its adaptation for SEE [32]). This may be due to the difficulty in defining minority and majority values for regression. Some studies generate only synthetic inputs [23, 59]. However, they are either only applicable to images [59] or require large training sets, which are unavailable for SEE [23].

To our best knowledge, there has been only one work in the SEE community that tackles the data scarcity problem by generating synthetic data [32]. Their proposed approach extended SMOTE from classification to regression by attributing class imbalance from the most predictive numerical feature, which is usually a size-related feature such as *functional size*. After casting the entire data samples into three classes (small, medium, and large according to, e.g., *functional size*), conventional SMOTE [10] was used to generate synthetic projects to small and medium classes to balance the data distribution. The entire data set size was thus increased. Then, these synthetic projects together with the real SEE data were passed to $k$-nearest neighbours ($k$-NN) for the purpose of getting better performance. Their experiments showed promising results based on Desharnais data set from SEACRAFT [48] repository.

Despite that the data generator of [32] was designed for $k$-NN, it can be easily extended for other SEE models as a data preprocessor. We will compare the effect of this data generator with ours in term of improving the performance of the baseline SEE models in Sec. 5.3.

## 3 OUR SYNTHETIC DATA GENERATOR

Different from the synthetic data generator in the literature [32], where a synthetic project was generated by a combination of two existing projects, our approach produces a synthetic project by displacing one existing project that is randomly selected.

Consider a training set of $N$ software projects $\mathcal{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^{N}$, where an input vector $\mathbf{x}^n \in \mathbb{R}^d$ includes software features such as *software development type*, *team expertise* and *functional size*, and $y^n$ is the actual effort for developing this software. Our synthetic data generator will produce $\lceil \gamma N \rceil$ synthetic projects to enlarge the training set size and tackle the SEE data scarcity problem, where $\gamma$ is the *synthetic rate* and $\lceil \cdot \rceil$ denotes the upward rounding operator (e.g. $\lceil 1.4 \rceil = 2$). The *synthetic rate* $\gamma$ should not be too large in order to retain the synthetic projects in good quality. In this paper, $\gamma$ is chosen from $\{0.25, 0.5, 0.75, 1\}$ as shown in table 3.

Overall, based on randomly selected training examples from the data set $\mathcal{D}$, the proposed data generator will produce $\lceil \gamma N \rceil$ synthetic projects one-by-one, each of which consists of two steps: synthetic feature generation and synthetic effort generation.

### 3.1 Synthetic Feature Generation

SEE features can be categorized into three classes according to the types of feature values: (1) categorical features with discrete nominal values such as *enhancement*, *re-development* and *new development* for *software development type*, (2) ordinal features with discrete ordinal values such as *very low*, *low*, *normal* and *high* for *team expertise*, and (3) numerical features with continuous values such as *functional size* and *line of codes*.

Given a randomly chosen training example $\mathbf{x} \in \mathcal{D}$, a synthetic project $\mathbf{x}^{(syn)}$ is generated feature-by-feature by displacing each training feature individually. The generation approach varies depending on the types of feature values as follows.

*3.1.1 Categorical Feature.* For a categorical feature $x_c \in \mathbf{x}$ with $k$ values $\{v_{c1}, \cdots, v_{ck}\}$, our proposed approach will generate its synthetic counterpart $x_c^{(syn)}$ by uniformly sampling a new categorical value from the set $\{v_{c1}, \cdots, v_{ck}\} \backslash \{v_{c,x_c}\}$, where $v_{c,x_c}$ denotes the categorical feature value of the chosen training project.

We assign a model parameter $0 \leq \tau < 1$ to the synthetic categorical feature generation, such that with probability $1 - \tau$ the synthetic feature retains the training value $v_{c,x_c}$, and with

probability $\tau$ the synthetic feature randomly takes a value from $\{v_{c1}, \cdots, v_{ck}\} \backslash \{v_{c,x_c}\}$ having the same probability for each value to be taken. The process can be formulated as

$$x_c^{(syn)} = \begin{cases} v_{c,x_c} & \text{if } \tau < \eta \leq 1 \\ \sim U(\{v_{c1}, \cdots, v_{ck}\} \backslash \{v_{c,x_c}\}) & \text{if } 0 \leq \eta \leq \tau \end{cases} \quad (1)$$

where $\eta$ is a random variable uniformly taken from [0,1], and $U(\{\cdots\})$ denotes a discrete uniform distribution function. To retain a moderate shift on the synthetic feature, we adopt small changing probability $\tau$ as listed in table 3.

Taking the categorical feature *development type* with values of *enhancement*, *re-development*, and *new development* as an example, if the training example is *re-developed*, the synthetic feature will stay the same with probability $1-\tau$, or be uniformly chosen from {*enhancement*, *new development*} with probability $\tau$.

*3.1.2 Ordinal Feature.* For an ordinal feature $x_o \in \mathbf{x}$ with $k$ values $\{v_{o1}, \cdots, v_{ok}\}$ where $v_{oi} \leq v_{oj}$ for $1 \leq i \leq j \leq k$, our approach will generate its synthetic counterpart $x_o^{(syn)}$ according to binomial distribution.

Binomial distribution $B(n,p)$ is frequently used to model the number of successes in a sequence of $n$ independent experiments, each of which succeeds with probability $p$ or fails with $(1-p)$ [8, 68]. For random variable $\xi \sim B(n,p)$, its *expectation* equals to $E[\xi] = np$. Binomial distribution is suitable to model ordinal features because it is a discrete distribution and can manifest the ordered relationship between feature values. Figure 1(a) illustrates the histogram of a binomial distribution $B(n = 10, p = 1/5)$.

We use an example to demonstrate our procedures in deciding the parameters of binomial distribution $B(n,p)$ of a training project. Given an ordinal feature *team expertise* with values of *1=very low*, *2=low*, *3=normal* and *4=high*, if the *team expertise* of the training example is *3=normal*, the synthetic feature should have the highest chance for taking *3=normal*, the second highest and the same chance for *4=high* and *2=low*, and the lowest chance for *1=very low*. To guarantee the expectation to be *3=normal*, the binomial parameters should satisfy $n \cdot p = 3$. To guarantee the same chance of taking *2=low* and *4=high*, $p$ should be 1/2. Combining the two equations, the binomial distribution should be $B(n = 6, p = 1/2)$. Figure 1(b) shows a solution of the binomial distribution for *team expertise*. It is noteworthy that to retain feature value *3=normal* situating at the distribution centre, three *dummy* values are added.

A synthetic ordinal feature is sampled from $B(n = 6, p = 1/2)$. If we get a *dummy* value, resume the sampling process until acquiring a valid feature value. The process can be formulated as

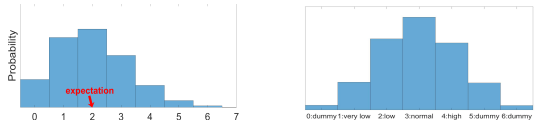
$$x_o^{(syn)} \sim B(n = 2 \cdot v_{o,x_o}, p = 1/2), \quad (2)$$

where $v_{o,x_o}$ is the ordinal feature value of the training example.

*3.1.3 Numerical Feature.* For a numerical feature $x_f \in \mathbf{x}$ with continuous values $x_f \in \mathbb{R}^1$, our proposed approach will generate its synthetic counterpart $x_f^{(syn)}$ by adding a zero-mean Gaussian variable $\epsilon \in \mathcal{N}(0, \sigma^2)$ to its baseline value $x_f$ as

$$x_f^{(syn)} = x_f + \epsilon_f, \quad \epsilon_f \sim \mathcal{N}(0, \sigma^2). \quad (3)$$

Usually the numerical features are size-related. Here, we normalize each numerical feature to have zero-mean and unit-variance, and assign Gaussian $\sigma^2$ with small values {0.1, 0.2, 0.3} as shown in table 3 to restrict the impact of Gaussian displacement.




(a) **Binomial $B(n=10, p=1/5)$.** (b) **Synthetic *team expertise*.**

**Figure 1: Binomial distribution and its ordinal feature modelling**

Overall, all numeric/ordinal features change with large probability based on Gaussian/binomial distribution, and each categorical feature has some chance to change based on the probability $\tau$.

## 3.2 Synthetic Effort Generation

Denote $y$ as the actual effort of training example $\mathbf{x}$, the aim of synthetic effort generation is to assign a proper value $y^{(syn)}$ to the synthetic feature $\mathbf{x}^{(syn)}$.

Similar to the numerical feature generation, our approach assigns the synthetic effort by adding a zero-mean Gaussian variable $\epsilon \sim \mathcal{N}(0, \sigma'^2)$ to its baseline effort value as

$$y^{(syn)} = y + sign(\epsilon_f) \cdot |\epsilon|, \quad \epsilon \sim \mathcal{N}(0, \sigma'^2), \quad (4)$$

where $sign(\epsilon_f)$ is the positive/negative sign of the injected Gaussian variable of the numerical feature in Eq. (3). When there are more than one numerical features, $\epsilon_f$ is their summation.

By doing so, $(y^{(syn)} - y)$ and $(x_f^{(syn)} - x_f)$ can have the same increasing/decreasing direction, catering the well-known fact that numerical size-related features are positively correlated with effort values [11, 44]. In this work, we confine $\sigma' = \sigma$ for simplicity. Exploration of a separate parameter $\sigma'$ can be conducted in future.

## 3.3 Further Discussions and Summary

There are several research lines that may further enhance the effectiveness of our proposed synthetic generator as:

• Our ordinal/categorical feature modelling may not fit reality perfectly. For instance, a *newly developed* software project would be more likely to be *enhanced* rather than *re-developed*; employees with *normal* expertise would be more likely to evolve to *high* rather than *low* expertise. Thus, it is interesting to study whether other non-symmetric distributive modellings of ordinal/categorical features would improve performance further. This would depend on expert knowledge of the data distribution.

• Our approach assumes that synthetic efforts are only affected by the change in numerical features. Assigning synthetic effort from the changes of ordinal/categorical features is very challenging, as it requires expert knowledge or data analyses with large training sets. This is potentially a harder problem than SEE itself. Since our strategy has achieved good results, we did not investigate effects of changes in categorical/ordinal features on synthetic effort. Nevertheless, it is an interesting research direction.

In summary, our data augmentation approach generates synthetic projects individually, each of which is based on *slight* displacement of a training example that is chosen randomly. Thus, the produced synthetic projects can only impact the local areas they are generated. Besides, our synthetic data generator is data-driven and does not depend on any effort estimator. Thus, it can be used as a preprocessor with any SEE model.

# 4 EXPERIMENTAL DESIGN

## 4.1 Data Sets

The experiments are based on 14 data sets from the Software Engineering Artifacts Can Really Assist Future Tasks (SEACRAFT) [48] (former PROMISE [47]) and the International Software Benchmarking Standards Group (ISBSG) Release 10 [26]. To investigate the effect of the training set size, the data sets are grouped into small, medium, and large according to the ratio of the number of data over the number of features. Table 1 contains the basic description of the investigated data sets.

**Maxwell** [16] contains 62 projects from one of the biggest commercial banks in Finland, covering the years from 1985 to 1993 and both in-house and outsourced development. We removed the input features *start year* (*syear*) and *duration* (= *syear*-1985+1). *Syear* was removed because it was found to have no significant effect on the dependent effort according to one-way ANOVA [56]. *Duration* was removed since it was unknown in reality during effort prediction process. After the removal of 2 features, 23 input features were left.

**Cocomo81** and **Nasa93** were collected in the COCOMO [7] data format, which has 17 features consisting of 15 *cost drivers*, *lines of codes* and *development type*. We used the COCOMO numeric values for the cost drivers. Cocomo81 has 63 projects. Nasa93 contains 93 projects developed between 1970's and 1980'.

**Albrecht** contains 24 projects developed in IBM using the third generation languages in the 1970s [1]. Eighteen out of 24 projects were written in COBOL, four were written in PL1, and two were writen in DMS languages. Seven input features were used. The dependent effort is recorded in 1,000 hours.

**Kemerer** contains 16 projects donated by Dr. Jacky W. Keung in 2010. We use 6 input features and remove the feature *project ID* since it is irrelevant to the effort prediction.

**Desharnais** contains 81 projects with nine features from a Canadian software company. Four projects contained missing values, so they were excluded from our investigation. The 8 input features in use are *TeamExp*, *ManagerExp*, *Transactons*, *Entities*, *PointsNonAdjust*, *Adjustment*, *PointsAjust*, and *Language*. The depended feature *effort* is recorded in 1,000 hours.

**Kitchenham** contains 145 projects undertaken between 1994 and 1998 by a single software development company [15, 35]. We removed the input features *project ID*, *actual start date*, *actual duration*, *estimate completion data*, *first estimate* and *first estimate method*. *Project ID* was removed because it was irrelevant with SEE prediction. *Actual start date* was removed following the same preprocessing as [35]. *Completion date* together with *start date* would give the duration of the project, and *duration* was removed because it was considered as a dependent variable of SEE process. The other features were removed because they were themselves estimations of completion date or effort, or represent the method used for such estimations. This feature preprocessing led to 3 remaining features: *adjusted function points*, *project type* and *client code*.

**ISBSG release 10** [26] contains a large body of software projects (5,052 projects), covering many different companies, several countries, organisation types, application types, etc. We preprocessed ISBSG repository with the same procedures as [49]. We maintained 621 projects by only keeping projects with relatively high quality.

**Table 1: SEE data sets that are cast into 3 groups representing *small*, *medium* and *large* data set sizes according to the ratio of the number of data over the number of features. Three sets of *holdout* values are assigned to three groups of data sets respectively.**

| Size | Data set | #Fea | #Data | #Fea/#Data | Small | Medium | Large |
|---|---|---|---|---|---|---|---|
| Small | Maxwell | 23 | 62 | 2.70 | 0.3 | 0.7 | LOO |
| | Cocomo81 | 17 | 63 | 3.71 | | | |
| | Nasa93 | 17 | 93 | 5.47 | | | |
| | Albrecht | 7 | 24 | 3.43 | | | |
| | Kemerer | 6 | 16 | 2.67 | | | |
| Medium | Desharnais | 8 | 77 | 9.63 | 0.1 | 0.3 | 0.7 |
| | Org2 | 3 | 32 | 10.67 | | | |
| | Org5 | 3 | 21 | 7.00 | | | |
| | Org6 | 1 | 22 | 22.00 | | | |
| | Org7 | 1 | 20 | 20.00 | | | |
| Large | Kitchenham | 3 | 145 | 48.33 | 0.04 | 0.08 | 0.7 |
| | Org1 | 3 | 76 | 25.33 | | | |
| | Org3 | 3 | 162 | 54.00 | | | |
| | Org4 | 3 | 122 | 40.67 | | | |

They were grouped into several data sets according to the *organisation type*, and only the groups with at least 20 projects were maintained following ISBSG's data size guidelines. The resulting organisation types are shown in Table 2.

The ISBSG suggests that the most important criteria for estimation purpose are *functional size*, *development type* (new development, enhancement or re-development), *primary programming language* (3GL, 4GL or ApG) and *development platform* (mainframe, midrange or PC). As *development platform* is missing in more than 40% of the projects for two organisation types, the remaining three criteria were used as input features.

Note that all projects of Org6 had the same *development type* and *programming language*, so *functional size* was used as a single feature. In Org7, all projects had the same *development type* and *programming language* with only one exception. Removing the exception, we had 20 projects with a single input feature.

**Data preprocessing.** For each data set in table 1, we apply the logarithm to the numerical features making them less skewed and more Gaussian distributed. Exponential distributions of numeric features are often observed in defect and effort prediction data sets, which are usually composed of many small values combined with a few much larger values [46, 62]. Logarithm preprocessor has shown to be non-harmful to or even sometimes improve the performance of the defect prediction [46, 62]. Our preliminary experiments on SEE have also shown either similar or better performance when using the logarithm scales of the numeric features compared to using their original values.

Using the logarithm preprocessor, all numeric features are replaced with their natural logarithm values. This procedure also minimizes the effects of the occasional very large feature values. Furthermore, each feature was normalized to be zero-mean and unit-variance to avoid scalability problem.

For the dependent outputs, we converted the numerical efforts into their logarithm scales to make the effort distribution more Gaussian. This procedure can also alleviate the prediction problem when treating test sample with very large effort.

**Table 2: ISBSG data sets grouped according to organization type and only the groups with at least 20 projects were maintained following ISBSG's data size guideline.**

| ID | Organisation Type | #Data |
|---|---|---|
| 1 | financial, property & business services | 76 |
| 2 | banking | 32 |
| 3 | communications | 162 |
| 4 | government | 122 |
| 5 | manufacturing, transport & storage | 21 |
| 6 | ordering | 22 |
| 7 | billing | 20 |

## 4.2 Performance Evaluation

There are several performance metrics for SEE evaluation [13, 50]. Popular examples are Mean Absolute Error (MAE), Mean Magnitude of the Relative Error (MMRE), Percentage of estimations within $N$% of actual values (Pred($N$)), Logarithmic Standard Deviation (LSD) [21], and Standardised Accuracy (SA) [57]. Different performance metrics emphasize different factors and can behave differently in effort model evaluation [50]. For instance, MMRE was shown to be biased towards prediction systems that underestimate effort [21, 36, 52, 57]. Underestimation (over-optimism) is the direction of the error that practitioners are more unwilling to see [28, 30], so we did not use MMRE in our investigation.

The performance metric used in this paper is MAE defined as $\sum_{i=1}^{N} |y_i - \widehat{y_i}|/N$, where $y_i/\widehat{y_i}$ is the actual/estimated effort, and $N$ is the number of testing data. MAE was recommended by Shepperd and MacDonell for SEE studies for being symmetric and not bias towards under or overestimation [57]. As the effort is in the logarithm scale, this metric becomes less affected by project size.

We apply *holdout* evaluation to control the training set size deliberately and evaluate the effects of synthetic data when training set size is small, medium, and large respectively. We randomly split the data set into training and testing subsets. Each SEE model is trained from the training set and its performance is evaluated from the testing set. This process is repeated 30 times and the average MAE is reported.

## 4.3 Baseline SEE Predictors Investigated

We investigate 6 SEE models: linear regression (LR), automatically transformed linear model (ATLM), $k$-nearest neighbour ($k$-NN), relevance vector machine (RVM), regression tree (RT) and support vector regression (SVR), since they are among the state-of-the-art SEE predictors [13, 39, 42, 49, 61, 71]. Each of them is used as a baseline model to investigate whether or not the generated synthetic data can improve its prediction performance. These models are implemented in MATLAB and specified if otherwise.

LR and ATLM [72] are chosen because they have been shown to be good baselines after appropriate data transformations [34, 72]. *R.matlab* package [6] was used to configure the R implementation of ATLM into the MATLAB framework.

$K$-NN is chosen for being among the simplest prediction model and due to its intuitive interpretation that mimics the human instinctive decision-making [39, 44, 58, 60]. Some empirical studies have showed that $k$-NN is comparable and sometimes superior to other SEE models [3, 29, 39, 44, 58]. To predict the effort of a testing project, the distances of this data to all training examples are computed in Euclidean metric. Based on them, $k$ nearest neighbours to the testing project are determined, and their median is returned as the estimated effort of this testing project [41].

**Table 3: Parameter values of the SEE models investigated.**

| ID | Method | Parameters |
|---|---|---|
| 1 | LR | No tuning parameter |
| 2 | ATLM | No tuning parameter |
| 3 | $k$-NN | $k$ (#neighbour) = {1,2,3,5} |
| 4 | RVM | $s$ (width) = 0.1 : 0.5 : 10 (#=20) |
| 5 | RTs | L (max tree depth) = {-1, 2, 6}<br>M (min #node per leaf) = {1, 2, 4}<br>E (stopping error) = {0.0001, 0.01, 0.5} |
| 6 | SVR | *kernel* = 'linear'<br>C (regularization) = {0.01, 0.1, **1**, 10 }<br>$\epsilon$ (slack variables) = {0.1, **0.3**, 0.5, 1} |
| 7 | syn.our | $\gamma$ (synRate) = {0.25,0.5,0.75,1}<br>$\tau$ (categorical) = {0,0.2,0.4}<br>$\sigma^2$ (GaussVar) = {0.1,0.2,0.3} |
| 8 | syn.cmp | $k$ (neighbours in SMOTE) = {1,2,3,5} |

RVM is chosen because it has been shown to be very competitive compared to other state-of-the-art SEE models and can be used to provide uncertain effort prediction [20, 61, 63]. In RVM, each training data is associated with one *basis function*, measuring the distance of this training project to the testing project. There are several choices for the basis function. We employ non-normalized Gaussian kernel $\phi_j(\mathbf{x}) = exp\{-(\mathbf{x} - \boldsymbol{\mu_j})^2/(2s^2)\}$ as our basis function for its *locality* property [49], where the $\boldsymbol{\mu}_j$ is the $j$-th training sample and the width $s$ controls their spatial scale.

RT is chosen for being among the most frequently used SEE models which has presented potential advantage for SEE [49, 71]. RT is a rule-based, hierarchical model where software data features are used to split projects into to small groups and this process is recursively repeated to form a regression tree [49].

SVR is designed for small data problems [18], which seems suitable to effort estimation. However, SVR has not been popularly used in SEE community partially because of the contradictory conclusions drawn from previous studies [2, 12, 53, 55]. Some claimed its superior performance in SEE [12, 53, 55], while others claimed inferior performance of SVR compared with other SEE models [2]. There are several choices for SVR kernel. We use linear kernel that has been shown to be a better choice [53].

**Parameter settings.** The parameter values of the SEE models investigated in this paper are listed in Table 3. For RT, the maximum tree depth of -1 means unlimited tree depth. For SVR, we investigate the conventional settings for regularization parameter $C$ and slack variable $\epsilon$ [12, 51]. For the model that has more than one parameters, we investigate its all parameter combinations. Our discussion is based on the performance of the best parameter settings, with which the SEE predictors can achieve their best performance.

# 5 RESULT AND DISCUSSION

This section aims to evaluate our synthetic data generator by comparing the performance of the SEE models with and without using the generated synthetic projects. For simplicity, the performance of the SEE model that does not use synthetic data is represented by *bsl.SEEr*, and the performance of the SEE model that uses the synthetic data generated by our approach is represented by *syn.SEEr*, where *SEEr* is one of the SEE models discussed in Sec. 4.3. For a more

thorough assessment, Sec. 5.3 compares our synthetic generator against its competitor in the SEE literature [32]. The performance of SEE models that uses the synthetic projects generated by this generator is represented by *syn.cmp.SEEr*.

## 5.1 Effect of Synthetic Data on Performance

This subsection aims to answer RQ1. To this aim, we investigate the effect of the synthetic data by comparing the performance of *syn.SEEr* against *bsl.SEEr* across 14 data sets with small, medium and large training set sizes respectively. Table 4 lists the performance comparisons in all settings. We can see that the synthetic data generated by our approach can usually improve the performance.

To investigate whether the improvement is significant, the effect size between *syn.SEEr* and *bsl.SEEr* across 30 runs of each data set is checked. Effect size is a simple way of quantifying the size of the difference between two methods with multiple runs [57, 67]. The Vargha and Delaney's $A_{12}$ is a non-parametric effect size that makes no assumptions about the underlying distribution [4, 67], which is interpreted in terms of Vargha and Delaney's categories: small (≥0.56), medium (≥0.64) and large (≥ 0.71) [67]. In table 4, large/medium/small effect size is highlighted in orange bold/yellow bold/bold indicating the performance improvement of using the synthetic projects generated by our approach.

We perform Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 to judge whether performance difference between *bsl.SEEr* and *syn.SEEr* is statistically significant across all data sets. Wilcoxon signed-rank tests are typically used to compare the performance of two models across multiple data sets [14, 73]. The null hypothesis (H0) states that the two models are equivalent. The alternative hypothesis (H1) states that they differ significantly.

Wilcoxon signed rank tests also provide the average ranks of *bsl.SEEr* vs *syn.SEEr* across 14 data sets calculated as $R_j = \frac{1}{N}\sum_i r_j^{(i)}$, where $r_j^{(i)}$ is the rank of the $j^{th}$ method on the $i^{th}$ data set, $j \in \{bsl.SEEr, syn.SEEr\}$, $i \in \{1, \cdots, N\}$, and $N = 14$ is the number of data sets. The average rank (*aveRank*) provides a reasonable comparison between *bsl.SEEr* vs *syn.SEEr* given rejection of the null hypothesis [14].

*5.1.1 LR and ATLM.* Since ATLM is a variant of LR using the automatic data transformation mechanism, we discuss the effect of our synthetic projects on them together.

**For small training set size**, we can see from table 4(a) that the synthetic projects generated by our approach can drastically improve the performance of LR/ATLM with large effect size in five out of seven SEACRAFT data sets. The synthetic data never hurts the performance of LR/ATLM in any SEE data set investigated. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all SEE data sets detect significantly better performance of *syn.LR*/*syn.ATLM* over *bsl.LR*/*bsl.ATLM*.

It is noteworthy that the performance of LR/ATLM is unstable in some data sets. For example, ATLM performs extremely bad in Org1 with very large MAE (mean MAE of 30 runs) 668.348±3648.420. Further investigation found that ATLM performed extremely bad on one of the 30 runs with MAE 19,985.448. Removing this outlier, the mean MAE across the remaining 29 runs reduced to 0.968±0.355 for *syn.ATLM* vs 2.241±4.303 for *bsl.ATLM*, with $A_{12} = 0.6373$.

The unstable performance of LR/ATLM may be due to the scarcity of training samples. When the few training samples are close to each other, being more likely to happen given inadequate training data, LR/ATLM may suffer from ill-conditional problem when doing matrix inversion in the training process. Another possible reason for ATLM is the incorrect statistic estimate on its automatic transformation mechanism caused by insufficient training samples.

**For medium training set size**, we can see from tables 4(a) vs 4(b) that *bsl.LR*/*bsl.ATLM* can achieve superior and more stable performance using medium compared to small training set sizes, indicating that augmenting the training data from an insufficient number can improve the performance of LR/ATLM. Similar observation can also seen for *syn.LR*/*syn.ATLM*.

We can also see that our synthetic projects can improve the performance of LR/ATLM especially for SEACRAFT data sets: the effect size $A_{12}$ is large in 3 data sets. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all data sets detect significant better performance of *syn.LR*/*syn.ATLM* over *bsl.LR*/*bsl.ATLM*, showing an overall superiority when the training set size is medium.

**For large training set size**, the superiority of *syn.LR*/*syn.ATLM* over *bsl.LR*/*bsl.ATLM* becomes smaller than for medium/small training set sizes. For instance, effect size $A_{12}$ is medium or small in only two SEACRAFT data sets. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all data sets detect significantly better performance of *syn.LR*/*syn.ATLM* over *bsl.LR*/*bsl.ATLM* with $p$-value 0.016255/0.00067.

**Summary.** Our synthetic data can always improve the baseline performance of LR and ATLM, and the improvement magnitude is usually significant having large or medium effect size especially when the training data is insufficient. When the training set size is large, our synthetic projects can hardly have detrimental effect and sometimes significantly improve the baseline performance.

*5.1.2 RVM and RT.* Table 4 shows that our synthetic data can usually improve the baseline performance of RVM and RT.

**For RVM,** our synthetic data can always improve their baseline performance. Their effect sizes are sometimes large or medium, showing substantial performance improvement on RVM. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all data sets detect significant overall superiority of using our synthetic data for all the training set sizes.

**For RT,** our synthetic data can always improve their baseline performance. When they are helpful with RT, the effect size is often large or medium especially when the training set size is not large. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all data sets detect significant overall superiority of using our synthetic data for medium and large training set sizes.

**Summary.** Our synthetic data can enhance the performance of RVM and RT, and the improvement is often significant especially when the training set size is not large.

*5.1.3 K-NN and SVR.* We can see from table 4 that our synthetic projects can usually improve the performance of $k$-NN and SVR, but the improvement is not very large.

**For $k$-NN,** our synthetic data can usually improve the baseline performance. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all data sets detect

**Table 4: Performance comparisons between each pair of *syn.SEEr* vs *bsl.SEEr* across 14 data sets in terms of MAE for small, medium and large training set sizes respectively. The different training set sizes refer to different *holdout* values of table 1. The reported values are the mean of 30 runs followed by their standard deviations (STDs). The comparison is highlighted in orange (dark grey) and bold font for large, in yellow (light grey) and bold font for medium, and in bold font for small effect size values. The last two rows of each sub table list the results of Wilcoxon tests with Bonferroni correction. The overall comparison between *bsl.SEEr* vs *syn.SEEr* can be seen from *aveRank* (average ranks). The first value 1 (or 0) in *Wilcoxon* row means there is (or not) significant difference detected, and its corresponding $p$-value comes the next. Significant difference is highlighted in orange (dark grey) on this row.**

(a) **Small training set size.**

| Data | syn.LR | bsl.LR | syn.ATLM | bsl.ATLM | syn.$k$-NN | bsl.$k$-NN | syn.RVM | bsl.RVM | syn.RT | bsl.RT | syn.SVR | bsl.SVR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Maxwell | **0.645±0.095** | **1.314±0.549** | **0.649±0.101** | **14.470±32.537** | 0.724±0.090 | 0.731±0.085 | **0.584±0.064** | **0.643±0.090** | **0.667±0.100** | **0.693±0.111** | **0.570±0.087** | **0.598±0.075** |
| Cocomo81 | **0.654±0.135** | **8.596±13.643** | **0.668±0.140** | **17.543±39.428** | 1.266±0.142 | 1.297±0.142 | **0.684±0.115** | **0.779±0.143** | **1.100±0.176** | **1.172±0.135** | **0.640±0.118** | **0.703±0.152** |
| Nasa93 | **0.534±0.082** | **0.942±0.877** | **0.540±0.081** | **0.927±0.836** | 0.990±0.111 | 0.984±0.107 | 0.532±0.113 | 0.534±0.121 | **0.728±0.078** | **0.796±0.083** | 0.519±0.079 | 0.544±0.148 |
| Kitchenham | **0.653±0.149** | **0.765±0.243** | **0.657±0.180** | **0.757±0.253** | 0.744±0.168 | 0.748±0.156 | **0.696±0.153** | **0.831±0.225** | **0.802±0.170** | **0.832±0.107** | **0.621±0.119** | **0.676±0.138** |
| Albrecht | **0.823±0.261** | **3.499±4.208** | **0.817±0.267** | **48.975±235.167** | 0.724±0.113 | 0.717±0.121 | **0.673±0.171** | **0.766±0.303** | **0.806±0.182** | **0.920±0.130** | 0.580±0.128 | 0.574±0.110 |
| Kemerer | 1.058±0.573 | 1.712±2.152 | **1.121±1.059** | **5.703±17.881** | 0.643±0.142 | 0.685±0.143 | 0.665±0.151 | 0.615±0.170 | 0.799±0.155 | 0.818±0.150 | **0.526±0.147** | **0.575±0.157** |
| Deshar | **0.695±0.193** | **2.163±4.230** | **0.699±0.188** | **3.235±4.758** | 0.622±0.088 | 0.618±0.084 | **0.583±0.095** | **0.626±0.160** | **0.639±0.094** | **0.692±0.068** | 0.526±0.051 | 0.526±0.067 |
| Org1 | **1.324±1.361** | **2.133±2.337** | **1.004±0.400** | **668.348±3648.420** | **0.895±0.203** | **0.907±0.134** | **0.922±0.228** | **0.988±0.234** | 1.027±0.297 | 1.000±0.256 | **0.853±0.209** | **0.874±0.160** |
| Org2 | 1.092±1.634 | 1.343±2.222 | 0.785±0.325 | 0.975±0.846 | 0.659±0.208 | 0.671±0.185 | 0.645±0.179 | 0.637±0.129 | 0.762±0.209 | 0.747±0.189 | 0.633±0.182 | 0.645±0.201 |
| Org3 | 0.684±0.146 | 0.744±0.229 | **0.682±0.148** | **0.745±0.231** | **0.767±0.132** | **0.782±0.114** | **0.753±0.192** | **0.855±0.188** | **0.835±0.142** | **0.971±0.101** | **0.647±0.124** | **0.701±0.189** |
| Org4 | **0.902±0.258** | **2.341±3.983** | **0.916±0.342** | **5.298±20.085** | 0.860±0.156 | 0.863±0.135 | 0.836±0.096 | 0.846±0.109 | 0.897±0.136 | 0.892±0.116 | **0.800±0.099** | **0.840±0.131** |
| Org5 | **2.177±2.983** | **3.837±4.172** | **1.231±1.287** | **2.413±3.398** | 0.971±0.232 | 1.009±0.239 | 1.042±0.270 | 1.287±1.366 | 1.060±0.195 | 1.036±0.172 | **0.771±0.188** | **0.938±0.265** |
| Org6 | **1.003±0.508** | **2.680±3.728** | **1.111±0.576** | **2.123±2.408** | 0.959±0.255 | 0.961±0.273 | **0.999±0.320** | **1.089±0.260** | 1.159±0.244 | 1.165±0.248 | 0.860±0.236 | 0.888±0.262 |
| Org7 | **1.156±0.651** | **1.868±2.498** | **1.179±0.674** | **1.890±2.494** | 0.917±0.168 | 0.909±0.150 | 0.953±0.235 | 0.946±0.155 | 0.959±0.167 | 0.946±0.137 | 0.923±0.220 | 0.892±0.148 |
| aveRank | 1.00 | 2.00 | 1.00 | 2.00 | 1.29 | 1.71 | 1.21 | 1.79 | 1.36 | 1.64 | 1.14 | 1.86 |
| Wilcoxon | 1 | 0.000122 | 1 | 0.000122 | 0 | 0.056274 | 1 | 0.006714 | 0 | 0.057983 | 1 | 0.005249 |

(b) **Medium training set size.**

| Data | syn.LR | bsl.LR | syn.ATLM | bsl.ATLM | syn.$k$-NN | bsl.$k$-NN | syn.RVM | bsl.RVM | syn.RT | bsl.RT | syn.SVR | bsl.SVR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Maxwell | **0.496±0.089** | **0.656±0.146** | **0.498±0.089** | **0.666±0.160** | **0.662±0.113** | **0.681±0.111** | **0.547±0.085** | **0.589±0.083** | 0.565±0.072 | 0.574±0.094 | **0.495±0.081** | **0.523±0.081** |
| Cocomo81 | 0.447±0.108 | 0.439±0.093 | 0.471±0.103 | 0.475±0.098 | **1.136±0.197** | **1.211±0.179** | 0.502±0.135 | 0.505±0.121 | **0.848±0.106** | **0.928±0.153** | 0.459±0.098 | 0.452±0.068 |
| Nasa93 | 0.444±0.075 | 0.450±0.088 | 0.444±0.075 | 0.450±0.088 | **0.817±0.143** | **0.842±0.097** | 0.448±0.100 | 0.458±0.100 | 0.621±0.138 | 0.638±0.120 | **0.411±0.078** | **0.433±0.073** |
| Kitchenham | 0.552±0.043 | 0.593±0.170 | 0.545±0.046 | 0.586±0.174 | 0.617±0.049 | 0.619±0.072 | **0.571±0.050** | **0.602±0.090** | **0.670±0.090** | **0.688±0.097** | 0.547±0.059 | 0.555±0.078 |
| Albrecht | **0.536±0.141** | **0.586±0.167** | **0.551±0.163** | **0.628±0.234** | 0.559±0.191 | 0.548±0.180 | 0.544±0.145 | 0.544±0.147 | **0.624±0.220** | **0.779±0.172** | 0.486±0.134 | 0.438±0.124 |
| Kemerer | **0.596±0.218** | **1.063±0.827** | **0.553±0.224** | **0.927±0.587** | 0.566±0.191 | 0.608±0.200 | 0.517±0.169 | 0.513±0.219 | **0.623±0.146** | **0.707±0.199** | 0.448±0.156 | 0.458±0.162 |
| Deshar | **0.490±0.068** | **0.561±0.094** | **0.489±0.066** | **0.561±0.094** | 0.531±0.036 | 0.531±0.045 | 0.480±0.053 | 0.486±0.072 | 0.550±0.072 | 0.558±0.078 | 0.447±0.045 | 0.451±0.050 |
| Org1 | **0.759±0.114** | **0.838±0.234** | **0.753±0.118** | **0.842±0.239** | 0.827±0.117 | 0.831±0.083 | **0.809±0.097** | **0.856±0.133** | **0.851±0.118** | **0.903±0.116** | **0.747±0.091** | **0.785±0.133** |
| Org2 | 0.542±0.065 | 0.559±0.080 | 0.538±0.064 | 0.553±0.082 | 0.595±0.083 | 0.597±0.086 | 0.566±0.078 | 0.561±0.074 | **0.590±0.074** | **0.689±0.092** | **0.531±0.091** | **0.547±0.078** |
| Org3 | **0.612±0.098** | **0.632±0.108** | 0.614±0.099 | 0.631±0.109 | 0.688±0.067 | 0.690±0.067 | **0.622±0.068** | **0.725±0.123** | **0.718±0.136** | **0.772±0.078** | 0.587±0.065 | 0.594±0.073 |
| Org4 | **0.706±0.070** | **0.825±0.221** | **0.704±0.068** | **0.815±0.202** | 0.782±0.084 | 0.791±0.073 | **0.726±0.065** | **0.807±0.093** | **0.769±0.073** | **0.851±0.053** | 0.717±0.072 | 0.732±0.100 |
| Org5 | 0.626±0.179 | 0.682±0.296 | 0.661±0.215 | 0.714±0.296 | 0.783±0.157 | 0.798±0.181 | **0.715±0.167** | **0.756±0.158** | **0.774±0.162** | **0.925±0.121** | 0.577±0.173 | 0.580±0.179 |
| Org6 | 0.729±0.134 | 0.806±0.287 | **0.751±0.157** | **0.839±0.298** | 0.746±0.161 | 0.783±0.179 | **0.795±0.269** | **0.900±0.272** | **0.808±0.141** | **0.999±0.145** | 0.688±0.117 | 0.721±0.154 |
| Org7 | 0.798±0.111 | 0.835±0.278 | 0.804±0.113 | 0.841±0.279 | 0.806±0.160 | 0.844±0.141 | 0.814±0.099 | 0.881±0.376 | **0.767±0.174** | **0.884±0.137** | 0.787±0.086 | 0.807±0.156 |
| aveRank | 1.07 | 1.93 | 1.00 | 2.00 | 1.07 | 1.93 | 1.21 | 1.79 | 1.00 | 2.00 | 1.14 | 1.86 |
| Wilcoxon | 1 | 0.000670 | 1 | 0.000091 | 1 | 0.000670 | 1 | 0.016255 | 1 | 0.000091 | 1 | 0.003763 |

(c) **Large training set size.**

| Data | syn.LR | bsl.LR | syn.ATLM | bsl.ATLM | syn.$k$-NN | bsl.$k$-NN | syn.RVM | bsl.RVM | syn.RT | bsl.RT | syn.SVR | bsl.SVR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Maxwell | 0.533±0.292 | 0.536±0.336 | 0.534±0.296 | 0.549±0.337 | 0.693±0.416 | 0.719±0.439 | **0.529±0.426** | **0.616±0.462** | 0.448±0.354 | 0.476±0.400 | 0.499±0.278 | 0.486±0.344 |
| Cocomo81 | 0.478±0.371 | 0.468±0.403 | 0.475±0.444 | 0.513±0.422 | 1.190±0.830 | 1.286±0.826 | 0.411±0.313 | 0.444±0.374 | **0.788±0.614** | **1.047±0.584** | 0.457±0.419 | 0.459±0.376 |
| Nasa93 | 0.394±0.476 | 0.368±0.460 | 0.394±0.458 | 0.368±0.460 | 0.653±0.720 | 0.692±0.761 | 0.409±0.456 | 0.467±0.471 | 0.446±0.394 | 0.426±0.354 | **0.338±0.454** | **0.393±0.450** |
| Kitchenham | 0.462±0.035 | 0.461±0.034 | 0.455±0.037 | 0.456±0.035 | 0.509±0.056 | 0.510±0.050 | 0.456±0.050 | 0.454±0.043 | 0.553±0.054 | 0.564±0.051 | 0.457±0.036 | 0.458±0.034 |
| Albrecht | 0.433±0.312 | 0.447±0.360 | 0.433±0.312 | 0.447±0.360 | 0.407±0.352 | 0.472±0.459 | 0.375±0.319 | 0.379±0.421 | **0.475±0.355** | **0.666±0.497** | 0.344±0.325 | 0.316±0.330 |
| Kemerer | 0.444±0.403 | 0.447±0.354 | 0.403±0.371 | 0.438±0.356 | 0.494±0.483 | 0.499±0.517 | 0.353±0.379 | 0.351±0.513 | 0.533±0.507 | 0.637±0.549 | 0.352±0.370 | 0.376±0.395 |
| Deshar | **0.437±0.049** | **0.464±0.059** | **0.438±0.059** | **0.465±0.059** | 0.502±0.082 | 0.507±0.077 | 0.425±0.062 | 0.435±0.063 | 0.447±0.074 | 0.457±0.088 | 0.426±0.050 | 0.430±0.051 |
| Org1 | 0.631±0.110 | 0.635±0.106 | 0.629±0.106 | 0.636±0.107 | 0.762±0.122 | 0.776±0.136 | 0.663±0.131 | 0.672±0.175 | **0.725±0.122** | **0.798±0.143** | 0.617±0.101 | 0.621±0.110 |
| Org2 | 0.464±0.087 | 0.476±0.097 | 0.460±0.082 | 0.467±0.096 | 0.532±0.105 | 0.532±0.110 | 0.468±0.095 | 0.469±0.107 | 0.514±0.095 | 0.506±0.093 | 0.451±0.074 | 0.459±0.082 |
| Org3 | 0.528±0.064 | 0.532±0.060 | 0.528±0.063 | 0.532±0.060 | **0.609±0.078** | **0.624±0.066** | **0.534±0.065** | **0.558±0.083** | 0.582±0.068 | 0.582±0.064 | 0.518±0.066 | 0.521±0.063 |
| Org4 | 0.644±0.063 | 0.652±0.062 | 0.646±0.064 | 0.655±0.061 | **0.709±0.081** | **0.729±0.072** | **0.638±0.068** | **0.655±0.066** | 0.698±0.077 | 0.721±0.088 | 0.645±0.069 | 0.650±0.066 |
| Org5 | **0.471±0.138** | **0.524±0.180** | **0.482±0.129** | **0.536±0.176** | 0.654±0.216 | 0.663±0.189 | **0.528±0.226** | **0.634±0.182** | 0.592±0.249 | 0.596±0.229 | 0.445±0.144 | 0.464±0.151 |
| Org6 | 0.619±0.138 | 0.653±0.154 | 0.632±0.142 | 0.661±0.153 | 0.677±0.136 | 0.679±0.120 | 0.570±0.121 | 0.563±0.112 | 0.657±0.173 | 0.670±0.154 | 0.586±0.121 | 0.607±0.129 |
| Org7 | 0.748±0.215 | 0.755±0.199 | 0.753±0.219 | 0.760±0.204 | 0.743±0.220 | 0.736±0.243 | 0.735±0.188 | 0.741±0.191 | 0.690±0.186 | 0.686±0.213 | 0.707±0.162 | 0.691±0.152 |
| aveRank | 1.21 | 1.79 | 1.07 | 1.93 | 1.07 | 1.93 | 1.21 | 1.79 | 1.21 | 1.79 | 1.21 | 1.79 |
| Wilcoxon | 1 | 0.016255 | 1 | 0.000670 | 1 | 0.000670 | 1 | 0.016255 | 1 | 0.016255 | 0 | 0.172607 |

significantly better overall performance of using our synthetic data for medium and large training set sizes. However, the effect size shows small or insignificant superiority.

**For SVR,** our synthetic data can usually improve the baseline performance. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all SEE data sets detect significant overall superiority of using our synthetic data for small and medium training set sizes. However, the effect size usually shows insignificant superiority.

**Superior performance of SVR.** We can see from table 4 that SVR usually outperforms other SEE predictors. Friedman tests at the significance level 0.05 across all data sets reject the null hypothesis (H0), which states that all models are equivalent. Nevertheless, our synthetic data can further improve its performance a little when there are insufficient training samples.

**Factors that impact the prediction performance of SEEr.** The superiority of SVR over SEE models is consistent with some previous works [12, 53, 53, 55], but contradicts some others [2]. One of the possible reasons would be the usage of different evaluation approaches that resulted in different training sizes. The performance of SEE models can be affected by the training set size. For instance, RVM performed the second best among all baseline models for small and large training size sets; but when the training set size was medium, it ranked the fourth after SVR, LR and ATLM. Some other factors that may affect the results of SEE model comparisons include the data sets used in the study, the type of preprocessing, the

performance metrics, the model parameter tuning, and the amount of fine tuning of the methods [5, 45, 60].

**Summary.** Our synthetic data can often improve the prediction performance of $k$-NN and SVR, though the improvement is usually small or insignificant. At least, our synthetic data is not detrimental.

*5.1.4 Summary.* Our synthetic projects are particularly helpful for LR and ATLM on small/medium data sizes; moderately helpful for RVM and RT, and not very helpful for $k$-NN and SVR. Nevertheless, they are rarely detrimental to the baseline performance. We have also computed the predictive performance based on MAE applied to actual efforts (not in the logarithm scale). The supplementary material will be available in Leandro Minku's homepage. The key conclusion that syn.SEEr always performed similarly/better than bsl.SEEr remained the same. Therefore, results with MAE applied to actual efforts were omitted due to space constraints.

## 5.2 Underlying Reasons for the Effect of Our Synthetic Data on Prediction Performance

This subsection aims to answer RQ2. Given the results summarised in Sec. 5.1.4, RQ2 can be rephrased as:

RQ2.1 Why do our synthetic projects usually have positive effect to an SEE model?

RQ2.2 Why do our synthetic projects have different improvement magnitude for different SEE models?

When the training set size is large, an SEE predictor can usually achieve relatively good performance, leaving limited improvement space for using our synthetic projects. Thus, our discussion will focus on the case with insufficient training examples.

*5.2.1 Positive Effect of Synthetic Data.* This subsection aims to answer RQ2.1 in view of the augmentation of data set and the enhanced ability to cope with data noise.

The mai=n reason would be the augmentation of the SEE data by encompassing the synthetic projects into the construction of SEE models, which directly tackles the data scarcity problem of SEE.

Another reason would be the enhanced ability to cope with data noise that can lead to large variations from the actual values, i.e., large noise. Effort values are highly likely to contain noise due to the participation of humans in SEE data collection [27, 31, 61]. When training examples are insufficient, such noise is more likely to mislead the construction of SEE models, resulting in less correct and unstable performance. When the training data contains noise and the amount of noise is smaller than the predictive information, the synthetic projects can compensate the possibly negative effect and enhance the prediction robustness. Figure 2 illustrates the positive effect of our synthetic data on a linear SEE model.

Our approach emphasizes the more typical areas of learning space, helping to avoid being misled by large noise. Specifically, the training projects that locate in crowded regions, which are less likely to contain large variations, are more likely to be chosen for generating our synthetic projects. In this way, our synthetic data emphasizes the space with small or no noise, and impacts the neighbourhood of those training projects by encoding more representatives. This would enhance the robustness of this local area when being used to construct an SEE model. On the other hand, our synthetic projects can be rarely generated in sparse regions, where large variations are more likely to exist. In this way, we can circumvent the issue of introducing data noise that can lead to large variations from the actual values. In this sense, the use of our synthetic data can usually have positive effect to an SEE model by enlarging the data set and compensating the noisy data.

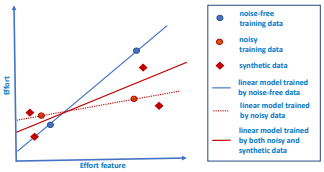



**Figure 2: Illustration that synthetic data can improve the quality of the parameter estimate of a linear SEE model. The synthetic project (square) enhances the robustness of its neighbourhood and alleviates detrimental effect of noisy training examples.**

It is noteworthy that data noise can only be filtered out if ground-truth noise-free projects are known. However, such ground truth is not known in reality. Therefore, coping with noise by filtering would be difficult, and our proposed approach can be a good alternative. Moreover, our synthetic projects may introduce noise but only in the form of small variations in the projects, since our synthetic generator emphasizes the space with smaller variation and generates synthetic data with small change.

*5.2.2 Effect of Synthetic Data on Each SEE Model.* This subsection aims to answer RQ2.2 in view of the *locality*/*globality* property of each SEE model.

***Locality and globality of SEE models.*** SEE approaches that perform estimations based on training examples that are similar to the testing project are referred as *local* approaches [22, 49]. The opposite terminology is referred here as *globality*, where the effort estimation is performed based on all training examples regardless of the similarity to the project to be estimated. Recall that our synthetic data can only impact their neighbourhood, so the *locality*/*globality* property of an SEE model would be a primary avenue to spread the effect of the synthetic data from the neighbourhood to other areas having projects to be estimated.

**LR/ATLM** is an example of SEE models with thorough *globality*. All training examples, regardless of their similarity to the project to be estimated, are used to estimate the model parameters, which are then used to predict the effort of the testing project. Therefore, the effects of our synthetic data in one area will impact the predictions in the entire space, leading to remarkable effect of our synthetic data on the prediction performance. In particular, if synthetic examples are created in an area with several examples where the model can become quite confident in their predictions, this could improve the predictions in the areas with less examples, where the model would originally not be confident about.

**$K$-NN** is an example of SEE models with thorough *locality*, where the prediction of a project is only based on the training examples in its neighbourhood. Therefore, the effect of our synthetic data in one area will not impact the predictions in other area, causing little effect of our synthetic data.

**RT** possesses a hybrid property of *globality* and *locality*. On the one hand, RT has *globality*. To construct RT, all training examples are used to decide the split features and the corresponding thresholds on which the tree branches are formed. On the other hand, RT has *locality*. To predict the effort of the testing project, RT needs to find a branch where the testing project is more similar to the training examples of this branch. The effort prediction is based on

the training subset. Therefore, the effect of our synthetic data in one area will impact the predictions in other area to some extent.

**RVM** is another example of SEE models with a hybrid property of *globality* and *locality*. On the one hand, RVM has *globality*. To construct RVM, all training examples are used to estimate the optimal model parameters. On the other hand, RVM has *locality*. The prediction of RVM is a weighted summation, with each weight being the similarity of the testing project to one training example. In this sense, only a subset of training data is used to predict the effort. Therefore, the effect of our synthetic data in one area can impact the prediction in the other area in some degree.

**SVR** has a *tolerance margin* ($\epsilon$ in table 3), with which data noise is tolerant to some extent. When the synthetic project locates within the *tolerance margin*, it can be seen as a disturbance of its original training example and thus has no effect on the decision of the model parameters. Only when the synthetic projects locates on the *tolerance margin*, namely when it is a *support vector*, it can effect the decision of the model parameters. In this sense, little improvement of using our synthetic projects is probably cased by the much less opportunity for them to be chosen as *support vectors*.

## 5.3 Comparison of Synthetic Generators

This subsection aims to answer RQ3. by comparing the performance of our synthetic generator against its only competitor in SEE [32], denoted by *syn.cmp.SEEr*.

*5.3.1 Syn.SEEr vs Syn.Cmp.SEEr.* Table 5 shows the performance comparisons of the two synthetic generators. We can see that, regardless of the SEE model, the performance using our synthetic generator (*syn.SEEr*) is often better than the performance using the competing synthetic generator (*syn.cmp.SEEr*) especially when the training set size is not large.

The effect size between *syn.SEEr* and *syn.cmp.SEEr* across 30 runs of each SEE data set is checked and exhibited on the cells in the columns of *syn.SEEr*. We can see that when the training set size is large, *syn.SEEr* usually has similar performance to *syn.cmp.SEEr*. When the training set size is not large, the superiority of our synthetic generator over its competitor can be considerable depending on SEE models. The superiority magnitude of *syn.SEEr* over *syn.cmp.SEEr* is often large for LR and ATLM, moderate for RT and RVM, and small for *k*-NN and SVR.

Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 between *syn.SEEr* and *syn.cmp.SEEr* show that when the training set size is not large, our synthetic generator is always superior to its competitor by having significantly better prediction performance.

*5.3.2 Syn.Cmp.SEEr vs bsl.SEEr.* Comparing the performance of *bsl.SEEr* in table 4 and *syn.cmp.SEEr* in table 5, we can see that *syn.cmp.SEEr* cannot outperform *bsl.SEEr* in many cases. Effect size across the 30 runs of each data set between *syn.cmp.SEEr* and *bsl.SEEr* is always small or insignificant, indicating that the synthetic examples generated by the literature do not have considerable impact on the prediction performance. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 show that *syn.cmp.SEEr* is similar to *bsl.SEEr* in most cases.

However, the competing synthetic generator was claimed to be effective in improving the performance of its baseline model [32]. Further examines found that the experiments of [32] were based on Desharnais data set only. The reported superiority of using their synthetic data was small and no statistical test was conducted. We suspect that with more data sets into their experiments and using statistic tests, their conclusions would probably be no significant difference with or without using their synthetic data.

# 6 THREATS TO VALIDITY

**Internal validity.** We did multiple Wilcoxon tests to evaluate the statistical significance of the results, which may induce type I error. For instance, to answer RQ1 we performed Wilcoxon post-hoc tests between *syn.SEEr* and *bsl.SEEr* across 14 data sets for 6 SEE models in 3 training set sizes leading to total 14*6*3=252 comparisons. However, we do not consider it to be very serious to this study, because these *p*-values were usually considerably small indicating very confident difference. Besides, the size of difference was also checked by effect size alleviating the problem of multiple comparisons.

Another potential threat to validity is the three extra parameters when using our synthetic generator. We did not investigate a very large number of possible values for these parameters. Despite that, our synthetic generator showed its effectiveness in improving performance of LR, ATLM, RVM and RT when the training set size is not large. Therefore, we do not consider further parameter tuning as essential for this study. As a future work, we will investigate the impact of parameter settings and present guidelines to tune the model parameter.

**Construct validity.** Our analyses are mainly based on MAE in the logarithm scale for being not biased towards under or over-estimation and for alleviating the dominance of very large effort values. Our preliminary studies showed that using other performance metrics such as median absolute error led to similar results as using MAE. As a future work, other performance measures could be investigated.

**External validity.** This study has not explored a full range of SEE models to be used with our synthetic generator in all SEE data sets. We may not be able to generalize the obtained findings to other SEE models or other SEE data sets. Nevertheless, since the chosen SEE models have been shown to be the state-of-the-art and the data sets covering a wide range of SEE data, this paper offers good support in the effectiveness of our synthetic generator in addressing the data scarcity problem of SEE.

# 7 CONCLUSIONS

We proposed a novel synthetic data generator to address data scarcity problem of SEE. Our approach produces a similar synthetic project by displacing a training example that is chosen randomly. The generated synthetic projects are then added to the training data set and used to train SEE models. Experimental results show positive effect of our approach in improving the baseline performance of SEE models and its superiority over the only synthetic generator of SEE literature [32]. We validate our data generator by answering the three research questions a follows.

**Ans1.** Experiments show that our synthetic projects always have positive effect on and are rarely detrimental to the performance of all SEE models investigated. They are particularly helpful for small and medium data set sizes for LR and ATLM, moderately helpful for RVM and RT, and not very helpful for *k*-NN and SVR. Nevertheless, they are hardly detrimental to the baseline performance.

**Table 5: Performance comparisons between *syn.SEEr* and *syn.cmp.SEEr* across 14 data sets in terms of MAE with small, medium and large training set sizes. The reported values are the mean of 30 runs followed by their STDs. The effect size across 30 runs of each SEE data set is used to measure the performance difference between *syn.SEEr* vs *syn.cmp.SEEr* and between *syn.cmp.SEEr* vs *bsl.SEEr*, which is exhibited in the cells associated with *syn.SEEr* and *syn.cmp.SEEr* respectively. The orange (dark grey) bold/yellow (light grey) bold/bold font indicates large/medium/small effects size. The last two rows list the results of Wilcoxon tests with Bonferroni correction across all data sets: the rows associated with *syn.SEEr* list the Wilcoxon results between *syn.SEEr* vs *syn.cmp.SEEr*, and the rows associated with *syn.cmp.SEEr* list the Wilcoxon results between *syn.cmp.SEEr* vs *bsl.SEEr*. Significant difference of Wilcoxon tests is highlighted in orange (dark grey).**

(a) **Small training set size.**

| Data | syn.LR | syn.Cmp.LR | syn.ATLM | syn.Cmp.ATLM | syn.$k$-NN | syn.Cmp.$k$-NN | syn.RVM | syn.Cmp.RVM | syn.RT | syn.Cmp.RT | syn.SVR | syn.Cmp.SVR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Maxwell | **0.645±0.095** | 1.336±0.487 | **0.649±0.101** | 14.682±32.480 | **0.724±0.090** | **0.777±0.071** | **0.584±0.064** | **0.603±0.066** | 0.667±0.100 | 0.684±0.091 | **0.570±0.087** | 0.601±0.070 |
| Cocomo81 | **0.654±0.135** | 8.596±13.643 | **0.668±0.140** | 12.827±28.833 | 1.266±0.142 | 1.294±0.148 | **0.684±0.115** | **0.747±0.147** | **1.100±0.176** | 1.145±0.141 | **0.640±0.118** | 0.703±0.152 |
| Nasa93 | **0.534±0.082** | 0.958±0.913 | **0.540±0.081** | 0.949±0.863 | 0.990±0.111 | 0.987±0.102 | 0.532±0.113 | 0.533±0.114 | **0.728±0.078** | **0.758±0.059** | **0.519±0.079** | **0.559±0.091** |
| Kitchenham | **0.653±0.149** | 0.772±0.259 | **0.657±0.180** | 0.767±0.263 | **0.744±0.168** | 0.765±0.163 | **0.696±0.153** | **0.962±0.360** | **0.802±0.170** | 0.869±0.170 | **0.621±0.119** | 0.688±0.149 |
| Albrecht | **0.823±0.261** | 3.499±4.208 | **0.817±0.267** | 4.835±6.721 | 0.724±0.113 | 0.720±0.150 | **0.673±0.171** | 0.785±0.242 | **0.806±0.182** | 0.940±0.218 | **0.580±0.128** | **0.609±0.126** |
| Kemerer | 1.058±0.573 | 1.306±1.506 | **1.121±1.059** | 19.447±92.795 | 0.643±0.142 | 0.679±0.131 | **0.665±0.151** | **0.777±0.348** | **0.799±0.155** | **0.875±0.203** | **0.526±0.147** | 0.584±0.148 |
| Deshar | **0.695±0.193** | 2.189±4.224 | **0.699±0.188** | 2.787±3.025 | 0.622±0.088 | 0.630±0.102 | 0.583±0.095 | **0.587±0.122** | **0.639±0.094** | **0.669±0.088** | 0.526±0.051 | 0.523±0.069 |
| Org1 | **1.324±1.361** | 2.133±2.337 | **1.004±0.400** | 668.348±3648.420 | **0.895±0.203** | 0.907±0.134 | **0.922±0.228** | 0.989±0.239 | 1.027±0.297 | 1.000±0.256 | **0.853±0.209** | 0.874±0.160 |
| Org2 | 1.092±1.634 | 1.343±2.222 | 0.785±0.325 | 0.975±0.846 | 0.659±0.208 | 0.671±0.185 | 0.645±0.179 | 0.635±0.130 | 0.762±0.209 | 0.747±0.189 | 0.633±0.182 | 0.645±0.201 |
| Org3 | **0.684±0.146** | 0.751±0.234 | **0.682±0.148** | 0.750±0.238 | **0.767±0.132** | 0.804±0.148 | **0.753±0.192** | 0.920±0.326 | **0.835±0.142** | **0.974±0.178** | **0.647±0.124** | 0.701±0.190 |
| Org4 | **0.902±0.258** | 2.346±3.980 | **0.916±0.342** | 5.305±20.084 | 0.860±0.156 | 0.841±0.110 | **0.836±0.096** | **1.061±0.409** | **0.897±0.136** | **0.882±0.126** | 0.800±0.099 | 0.820±0.111 |
| Org5 | **2.177±2.983** | 3.837±4.172 | **1.231±1.287** | 2.413±3.398 | 0.971±0.232 | 1.009±0.239 | 1.042±0.270 | 1.287±1.366 | 1.060±0.195 | 1.036±0.172 | **0.771±0.188** | 0.938±0.265 |
| Org6 | **1.003±0.508** | 2.680±3.728 | **1.111±0.576** | 2.123±2.408 | 0.959±0.255 | 0.961±0.273 | **0.999±0.320** | 1.089±0.260 | 1.159±0.244 | 1.165±0.248 | 0.860±0.236 | 0.888±0.262 |
| Org7 | **1.156±0.651** | 1.868±2.498 | **1.179±0.674** | 1.890±2.494 | 0.917±0.168 | 0.909±0.150 | 0.953±0.235 | 0.946±0.155 | 0.959±0.167 | 0.946±0.137 | 0.923±0.220 | 0.892±0.148 |
| Wilcoxon | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| $p$-value | 0.000091 | 0.250000 | 0.000091 | 1.000000 | 0.024536 | 0.359375 | 0.003763 | 0.322266 | 0.057983 | 0.910156 | 0.003763 | 0.203125 |

(b) **Medium training set size.**

| Data | syn.LR | syn.Cmp.LR | syn.ATLM | syn.Cmp.ATLM | syn.$k$-NN | syn.Cmp.$k$-NN | syn.RVM | syn.Cmp.RVM | syn.RT | syn.Cmp.RT | syn.SVR | syn.Cmp.SVR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Maxwell | **0.496±0.089** | 0.646±0.151 | **0.498±0.089** | 0.659±0.168 | **0.662±0.113** | 0.699±0.106 | **0.547±0.085** | **0.564±0.074** | 0.565±0.072 | 0.582±0.094 | **0.495±0.081** | 0.514±0.076 |
| Cocomo81 | 0.447±0.108 | 0.449±0.087 | 0.471±0.103 | 0.461±0.087 | 1.136±0.197 | **1.150±0.149** | 0.502±0.134 | 0.507±0.111 | **0.848±0.106** | **0.893±0.159** | 0.459±0.098 | 0.465±0.059 |
| Nasa93 | **0.444±0.076** | **0.487±0.098** | **0.444±0.076** | **0.499±0.106** | **0.817±0.143** | 0.880±0.172 | **0.448±0.100** | **0.516±0.130** | 0.621±0.138 | 0.612±0.082 | **0.411±0.078** | 0.449±0.087 |
| Kitchenham | 0.552±0.043 | 0.582±0.108 | 0.545±0.046 | 0.576±0.111 | 0.617±0.049 | 0.621±0.068 | **0.571±0.050** | 0.645±0.282 | 0.670±0.090 | **0.655±0.055** | 0.547±0.059 | 0.552±0.067 |
| Albrecht | **0.536±0.140** | 0.576±0.147 | **0.551±0.163** | 0.592±0.169 | 0.559±0.191 | 0.530±0.158 | 0.544±0.146 | 0.539±0.160 | **0.624±0.220** | **0.692±0.164** | 0.486±0.134 | **0.464±0.103** |
| Kemerer | **0.596±0.218** | 1.056±0.878 | **0.553±0.223** | 0.903±0.642 | **0.566±0.191** | 0.607±0.187 | **0.517±0.169** | 0.547±0.177 | **0.623±0.146** | 0.736±0.194 | 0.448±0.156 | 0.458±0.159 |
| Deshar | **0.490±0.068** | 0.573±0.092 | **0.489±0.066** | **0.573±0.091** | **0.531±0.036** | **0.549±0.043** | **0.480±0.053** | **0.506±0.074** | 0.550±0.073 | 0.542±0.073 | 0.447±0.045 | 0.451±0.050 |
| Org1 | **0.759±0.114** | 0.869±0.345 | **0.753±0.118** | 0.870±0.348 | 0.826±0.117 | 0.858±0.142 | **0.809±0.097** | **0.986±0.383** | **0.851±0.118** | 0.900±0.116 | **0.747±0.091** | 0.786±0.101 |
| Org2 | **0.542±0.065** | 0.570±0.076 | **0.538±0.064** | 0.565±0.078 | 0.595±0.083 | 0.596±0.074 | 0.566±0.078 | 0.566±0.069 | 0.590±0.074 | **0.594±0.104** | **0.531±0.091** | 0.555±0.073 |
| Org3 | **0.612±0.098** | 0.643±0.124 | **0.614±0.099** | 0.643±0.124 | **0.688±0.067** | 0.719±0.113 | **0.622±0.068** | **0.740±0.322** | **0.718±0.136** | **0.746±0.110** | **0.587±0.065** | 0.620±0.109 |
| Org4 | **0.706±0.070** | 0.841±0.244 | **0.704±0.068** | 0.836±0.244 | 0.782±0.084 | 0.786±0.069 | **0.726±0.065** | 0.837±0.153 | 0.769±0.073 | **0.772±0.067** | **0.717±0.072** | **0.746±0.080** |
| Org5 | 0.626±0.179 | 0.687±0.305 | **0.661±0.216** | 0.743±0.333 | 0.783±0.157 | 0.798±0.181 | **0.715±0.167** | 0.824±0.318 | **0.774±0.162** | **0.981±0.144** | 0.577±0.173 | 0.599±0.179 |
| Org6 | 0.729±0.134 | 0.771±0.226 | 0.751±0.157 | 0.808±0.268 | 0.746±0.161 | 0.787±0.185 | **0.795±0.269** | **1.047±0.407** | **0.808±0.141** | 0.999±0.145 | **0.688±0.117** | 0.725±0.128 |
| Org7 | 0.798±0.110 | **0.862±0.304** | 0.804±0.113 | 0.865±0.304 | 0.807±0.160 | 0.844±0.165 | **0.814±0.099** | **1.017±0.536** | **0.767±0.174** | **0.859±0.141** | 0.787±0.086 | 0.795±0.150 |
| Wilcoxon | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| $p$-value | 0.000091 | 0.216553 | 0.000670 | 0.541626 | 0.000670 | 0.463135 | 0.003763 | 0.003763 | 0.016255 | 0.067627 | 0.000670 | 0.041870 |

(c) **Large training set size.**

| Data | syn.LR | syn.Cmp.LR | syn.ATLM | syn.Cmp.ATLM | syn.$k$-NN | syn.Cmp.$k$-NN | syn.RVM | syn.Cmp.RVM | syn.RT | syn.Cmp.RT | syn.SVR | syn.Cmp.SVR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Maxwell | 0.533±0.292 | 0.497±0.353 | 0.534±0.296 | 0.519±0.388 | 0.693±0.417 | 0.713±0.464 | 0.529±0.427 | 0.501±0.380 | 0.448±0.354 | 0.510±0.443 | 0.499±0.278 | 0.459±0.344 |
| Cocomo81 | 0.478±0.371 | 0.449±0.430 | 0.475±0.444 | **0.449±0.430** | 1.190±0.830 | 1.219±0.830 | 0.411±0.313 | 0.446±0.339 | 0.788±0.614 | **0.892±0.587** | 0.457±0.419 | 0.444±0.437 |
| Nasa93 | 0.394±0.476 | 0.385±0.492 | **0.394±0.458** | 0.386±0.492 | 0.653±0.720 | 0.648±0.760 | **0.409±0.456** | 0.596±0.797 | 0.446±0.394 | **0.435±0.478** | 0.338±0.454 | 0.378±0.443 |
| Kitchenham | 0.462±0.035 | 0.464±0.040 | 0.455±0.037 | 0.453±0.036 | 0.509±0.056 | 0.512±0.053 | **0.456±0.050** | **0.484±0.049** | **0.553±0.054** | 0.575±0.058 | 0.457±0.036 | 0.462±0.035 |
| Albrecht | 0.433±0.312 | 0.422±0.333 | 0.433±0.312 | 0.422±0.333 | 0.407±0.352 | 0.429±0.395 | 0.375±0.319 | 0.354±0.345 | 0.475±0.355 | **0.547±0.419** | **0.344±0.325** | **0.363±0.250** |
| Kemerer | 0.444±0.403 | 0.476±0.384 | 0.403±0.371 | 0.477±0.402 | 0.494±0.483 | 0.490±0.495 | 0.353±0.379 | **0.361±0.394** | 0.533±0.507 | 0.557±0.516 | 0.352±0.370 | 0.364±0.407 |
| Deshar | **0.437±0.049** | 0.470±0.067 | **0.438±0.059** | 0.470±0.066 | **0.502±0.082** | 0.518±0.080 | 0.425±0.062 | 0.434±0.067 | 0.447±0.074 | 0.462±0.070 | 0.426±0.050 | 0.430±0.051 |
| Org1 | **0.631±0.110** | **0.671±0.126** | **0.629±0.106** | 0.651±0.117 | 0.762±0.122 | **0.746±0.132** | 0.663±0.131 | 0.682±0.149 | 0.725±0.122 | **0.754±0.103** | **0.617±0.101** | **0.666±0.133** |
| Org2 | 0.464±0.088 | 0.475±0.102 | 0.460±0.082 | 0.467±0.101 | 0.532±0.105 | 0.536±0.112 | 0.468±0.095 | 0.468±0.118 | 0.514±0.094 | 0.507±0.081 | 0.451±0.074 | 0.459±0.083 |
| Org3 | 0.528±0.063 | 0.528±0.066 | 0.528±0.063 | 0.529±0.066 | 0.609±0.078 | **0.608±0.069** | **0.534±0.065** | 0.572±0.089 | 0.582±0.068 | 0.581±0.064 | 0.518±0.066 | 0.520±0.065 |
| Org4 | **0.644±0.063** | 0.665±0.061 | **0.646±0.064** | 0.667±0.061 | 0.709±0.081 | 0.728±0.079 | **0.638±0.068** | 0.658±0.064 | 0.698±0.077 | 0.716±0.070 | **0.645±0.069** | **0.666±0.063** |
| Org5 | **0.471±0.138** | 0.511±0.154 | **0.482±0.129** | 0.529±0.149 | 0.654±0.216 | 0.647±0.199 | **0.528±0.226** | 0.614±0.184 | 0.592±0.249 | 0.617±0.193 | 0.445±0.144 | 0.468±0.137 |
| Org6 | **0.619±0.138** | 0.656±0.164 | **0.632±0.142** | 0.678±0.188 | 0.677±0.136 | 0.687±0.131 | 0.570±0.121 | 0.679±0.370 | 0.657±0.173 | 0.675±0.169 | 0.586±0.122 | 0.606±0.126 |
| Org7 | 0.748±0.215 | 0.761±0.197 | 0.753±0.219 | 0.761±0.198 | 0.743±0.220 | 0.710±0.246 | 0.735±0.188 | 0.769±0.267 | 0.690±0.186 | 0.727±0.233 | 0.707±0.162 | 0.690±0.152 |
| Wilcoxon | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| $p$-value | 0.104004 | 0.807739 | 0.172607 | 0.903198 | 0.426270 | 0.024536 | 0.003763 | 0.216553 | 0.016255 | 0.463135 | 0.104004 | 0.951538 |

**Ans2.** The positive effect of our synthetic data is mainly due to the data augmentation and the robustness enhancement in the areas that the noise of SEE projects may injure the quality of SEE model training. Different SEE models have different improvement magnitude that can be usually affected by their *locality*/*globality*.

**Ans3.** Studies show that our synthetic generator is significantly superior to or has no significant difference from its only competitor of SEE literature [32]. Besides, the competing generator probably brings no significant improvement over the baseline SEE models.

Future work includes further investigation of the impact of parameter settings and tuning guidelines, investigation of more performance metrics, and comparisons against the random strategy that assigns synthetic effort values with the outputs of randomly chosen training projects.

## ACKNOWLEDGMENTS

This work was supported by the Ministry of Science and Technology of China (Grant No. 2017YFC0804003), the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284), Shenzhen Peacock Plan (Grant No. KQTD2016112514355531), Shenzhen Key Laboratory of Computational Intelligence in Southern University of Science and Technology, and EPSRC (Grant Nos. EP/J017515/1, EP/R006660/1 and EP/P005578/1).

## REFERENCES


[1] A. J. Albrecht and J. E. Gaffney. 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering (TSE)* SE-9, 6 (1983), 639–648.

[2] S. Aljahdali, A. F. Sheta, and N. C. Debnath. 2015. Estimating Software Effort and Function Point Using Regression, Support Vector Machine and Artificial Neural Networks models. In *IEEE/ACS International Conference of Computer Systems and Applications (AICCSA)*. 1–8.

[3] L. Angelis and I. Stamelos. 2000. A Simulation Tool for Efficient Analogy Based Cost Estimation. *Empirical Software Engineering* 5, 1 (2000), 35–68.

[4] Andrea Arcuri and Lionel Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *International Conference on Software Engineering (ICSE)*. 1–10.

[5] A. Arcuri and G. Fraser. 2011. On Parameter Tuning in Search Based Software Engineering. In *Symposium on Search-Based Software Engineering (SSBSE)*. Szeged, Hungary, 33–47.

[6] Henrik Bengtsson. 2016. R.matlab: Read and Write MAT Files and Call MATLAB from Within R. R package version 3.6.0-9000. https://github.com/HenrikBengtsson/R.matlab.

[7] Barry W. Boehm. 1981. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ.

[8] D. Dennis Boos and L.A. Stefanski. 2013. *Essential Statistical Inference: Theory and Methods*. Springer New York.

[9] Paula Branco, Luís Torgo, and Rita P. Ribeiro. 2017. SMOGN: a Pre-processing Approach for Imbalanced Regression. In *First International Workshop on Learning with Imbalanced Domains: Theory and Applications (Machine Learning Research (MLR))*, Vol. 74. ECML-PKDD, Skopje, Macedonia, 36–50.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 6 (2002), 321–357.

[11] Zhihao Chen, Tim Menzies, Dan Port, and Barry Boehm. 2005. Feature Subset Selection Can Improve Software Cost Estimation Accuracy. In *International Conference on Predictor Models in Software Engineering (PROMISE)*. 1–6.

[12] Vladimir Cherkassky and Yunqian Ma. 2004. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks* 17, 1 (2004), 113 – 126.

[13] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens. 2012. Data Mining Techniques for Software Effort Estimation: A Comparative Study. *IEEE Transactions on Software Engineering (TSE)* 38, 2 (March 2012), 375–397.

[14] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research (JMLR)* 7 (2006), 1–30.

[15] Kitchenham doi. 2017. https://doi.org/10.5281/zenodo.268457.

[16] Maxwell doi. 2009. https://doi.org/10.5281/zenodo.268461.

[17] Dennis. J. Drown, Taghi. M. Khoshgoftaar, and Naeem Seliya. 2009. Evolutionary Sampling and Software Quality Modeling of High-Assurance Systems. *IEEE Transactions on Systems, Man, and Cybernetics* 39, 5 (2009), 1097–1107.

[18] Harris Drucker, Chris J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. 1996. Support Vector Regression Machines. In *Neural Information Processing Systems (NIPS)*. 155–161.

[19] Andrew Estabrooks. 2004. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence* (2004), 18–36.

[20] A. Faul and M. Tipping. 2001. Analysis of Sparse Bayesian Learning. In *Advances in Neural Information Processing Systems 14*. MIT Press, 383–389.

[21] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. 2003. A Simulation Study of the Model Evaluation Criterion MMRE. *IEEE Transactions on Software Engineering (TSE)* 29 (2003), 985–995.

[22] J. Cuadrado Gallego, D. Rodriguez, M. Sicilia, M. Rubio, and A. Cresp. 2007. Software Project Effort Estimation based on Multiple Parametric Models Generated through Data Clustering. *Journal of Computer Science and Technology* 22, 3 (2007).

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NIPS)*. 2672–2680.

[24] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

[25] Haibo He and Edwardo A. Garcia. 2009. Learning from Imbalanced Data. *IEEE Transaction on Knowledge and Data Engineering* 21, 9 (2009), 1263–1284.

[26] ISBSG. 2011. The International Software Benchmarking Standards Group. http://www.isbsg.org.

[27] M. Jorgensen. 2005. Evidence-Based Guidelines for Assessment of Software Development Cost Uncertainty. *IEEE Transactions on Software Engineering (TSE)* 32, 11 (2005), 942–954.

[28] Magne Jorgensen. 2013. The Influence of Selection Bias on Effort Overruns in Software Development Projects. *Information and Software Technology (IST)* 55, 9 (2013), 1640–1650.

[29] Magne Jorgensen, Ulf Indahl, and Dag Sjoberg. 2003. Software Effort Estimation by Analogy and 'Regression Toward the Mean'. *Journal of Systems and Software (JSS)* 68, 3 (2003), 253–262.

[30] Magne Jorgensen and Kjetil Molokken-Ostvold. 2006. How Large are Software Cost Overruns? A Review of the 1994 CHAOS Report. *Information and Software Technology (IST)* 48, 4 (2006), 297–301.

[31] M. Jorgensen, K. H. Teigen, and K. Molokken. 2004. Better Sure than Safe? Overconfidence in Judgement Based Software Development Effort Prediction Intervals. *Journal of Systems and Software* 70 (2004), 79–93.

[32] Yasutaka Kamei, Jacky Wai Keung, Akito Monden, and Ken-ichi Matsumoto. 2008. An over-sampling method for analogy-based software effort estimation. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 312–314.

[33] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. i. Matsumoto. 2007. The Effects of Over and Under Sampling on Fault-prone Module Detection. In *International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. 196–204.

[34] Barbara Kitchenham and Emilia Mendes. 2009. Why Comparative Effort Prediction Studies May Be Invalid. In *International Conference on Predictor Models in Software Engineering (PROMISE)*. New York, USA, 4:1–4:5.

[35] B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan. 2002. An Empirical Study of Maintenance and Development Estimation Accuracy. *Journal of Systems and Software (JSS)* 64, 1 (2002), 57–77.

[36] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd. 2001. What Accuracy Statistics Really Measure. *IEE Proceedings - Software Engineering* 148, 3 (2001), 81–85.

[37] E. Kocaguneli, B. Cukic, and H. Lu. 2013. Predicting More from Less: Synergies of Learning. In *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. San Francisco, CA, USA.

[38] E. Kocaguneli, B. Cukic, T. Menzies, and H. Lu. 2013. Building a Second Opinion: Learning Cross-Company Data. In *International Conference on Predictor Models in Software Engineering (PROMISE)*. Baltimore, USA.

[39] E. Kocaguneli and T.J. Menzies. 2012. Exploiting the Essential Assumptions of Analogy-based Effort Estimation. *IEEE Transactions on Software Engineering (TSE)* 38, 2 (2012), 425–438.

[40] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung. 2012. Exploiting the Essential Assumptions of Analogy-Based Effort Estimation. *IEEE Transactions on Software Engineering (TSE)* 38, 2 (2012), 425–438.

[41] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, and R. Madachy. 2013. Active Learning and Effort Estimation: Finding the Essential Content of Software Effort Estimation Data. *IEEE Transactions on Software Engineering (TSE)* 39, 8 (2013), 1040–1053.

[42] E. Kocaguneli, T. Menzies, and J. W. Keung. 2012. On the Value of Ensemble Effort Estimation. *IEEE Transactions on Software Engineering (TSE)* 38, 6 (2012), 1403–1416.

[43] Yigit Kultur, Burak Turhan, and Ayse Basar Bener. 2008. ENNA: Software Effort Estimation Using Ensemble of Neural Networks with Associative Memory. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 330–338.

[44] Y. Li, M. Xie, and T. Goh. 2009. A Study of Project Selection and Feature Weighting for Analogy Based Software Cost Estimation. *Journal of Systems and Software (JSS)* 82, 2 (2009), 241 – 252.

[45] C. Mair and M. Shepperd. 2005. The Consistency of Empirical Comparisons of Regression and Analogy-based Software Project Cost Prediction. In *International Symposium on Empirical Software Engineering*. 491 – 500.

[46] T. Menzies, J. Greenwald, and A. Frank. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering (TSE)* 33, 1 (2007), 2–13.

[47] T. Menzies, R. Krishna, and D. Pryor. 2015. The PROMISE Repository of Empirical Software Engineering Data. http://openscience.us/repo. North Carolina State University, Department of Computer Science.

[48] T. Menzies, R. Krishna, and D. Pryor. 2017. The SEACRAFT Repository of Empirical Software Engineering Data. https://zenodo.org/communities/seacraft.

[49] Leandro L. Minku and Xin Yao. 2012. Ensembles and Locality: Insight on Improving Software Effort Estimation. *Information and Software Technology (IST)* 55, 8 (2012), 1512–1528.

[50] Leandro L. Minku and Xin Yao. 2013. Software Effort Estimation as a Multiobjective Learning Problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22 (2013).

[51] Michinari Momma and Kristin P. Bennett. 2002. A Pattern Search Method for Model Selection of Support Vector Regression. In *International Conference on Data Mining*. 261–274.

[52] I. Myrtveit, E. Stensrud, and M. Shepperd. 2005. Reliability and Validity in Comparative Studies of Software Prediction Models. *IEEE Transactions on Software Engineering (TSE)* 31, 5 (2005), 380–391.

[53] Adriano L.I. Oliveira. 2006. Estimation of Software Project Effort with Support Vector Regression. *Neurocomputing* 69, 13 (2006), 1749 – 1753.

[54] L. Pelayo and S. Dick. 2007. Applying Novel Resampling Strategies To Software Defect Prediction. In *Annual Meeting of the North American Fuzzy Information Processing Society*. 69–72.

[55] Shashank Mouli Satapathy and Santanu Kumar Rath. 2014. Use Case Point Approach Based Software Effort Estimation using Various Support Vector Regression Kernel Methods. *CoRR* abs/1401.3069 (2014). http://arxiv.org/abs/1401.3069

[56] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris. 2005. Software Productivity and Effort Prediction with Ordinal Regression. *Information and Software Technology (IST)* 47, 1 (2005), 17 – 29.

[57] M. Shepperd and S. McDonell. 2012. Evaluating Prediction Systems in Software Project Estimation. *Information and Software Technology (IST)* 54 (2012), 820–827.

[58] M. Shepperd and C. Schofield. 1997. Estimating Software Project Effort Using Analogies. *IEEE Transactions on Software Engineering (TSE)* 23, 12 (1997), 736–743.

[59] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. 2013. *Real-Time Human Pose Recognition in Parts from Single Depth Images.* Springer Berlin Heidelberg, Berlin, Heidelberg, 119–135.

[60] Liyan Song, Leandro L. Minku, and Xin Yao. 2013. The Impact of Parameter Tuning on Software Effort Estimation Using Learning Machines. In *International Conference on Predictor Models in Software Engineering (PROMISE)*. Baltimore, USA, 9:1–9:10.

[61] Liyan Song, Leandro L. Minku, and Xin Yao. 2014. The Potential Benefit of Relevance Vector Machine to Software Effort Estimation. In *International Conference on Predictor Models in Software Engineering (PROMISE)*. Turin, Italy, 52–61.

[62] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. 2011. A General Software Defect-Proneness Prediction Framework. *IEEE Transaction on Software Engineering (TSE)* 37, 3 (2011), 356–370.

[63] M. Tipping. 2001. Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning* 1, June (2001), 211–244.

[64] Luis Torgo, Paula Branco, Rita P. Ribeiro, and Bernhard Pfahringer. 2015. Resampling Strategies for Regression. *Expert Systems* 32, 3 (2015), 465–476.

[65] Luis Torgo, Rita P. Ribeiro, Bernhard Pfahringer, and Paula Branco. 2013. SMOTE for Regression. In *Progress in Artificial Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 378–389.

[66] Vladimir N. Vapnik. 1998. *Statistical Learning Theory.* Wiley.

[67] Andras Vargha and Harold D. Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (6 2000), 101–132.

[68] George Wadsworth and Joseph Bryan. 1960. *Introduction to Probability and Random Variables.* McGraw-Hill. https://books.google.com.hk/books?id=NNtQAAAAMAAJ

[69] Shuo Wang and Xin Yao. 2013. Using Class Imbalance Learning for Software Defect Prediction. *IEEE Transactions on Reliability* 62 (2013), 434–443.

[70] Gary M. Weiss. 2004. Mining with Rarity: A Unifying Framework. *SIGKDD Explor. Newsl.* 6, 1 (2004), 7–19.

[71] 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 (IST)* 54, 1 (2012), 41–59.

[72] Peter A. Whigham, Caitlin A. Owen, and Stephen G. Macdonell. 2015. A Baseline Model for Software Effort Estimation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 3 (2015), 20:1–20:11.

[73] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968